# INTELLIGENT SYSTEMS AND SERVICES FOR IMAGE AND VIDEO ANALYSIS

Dimitrios E. Diamantis

Department of Computer Science and Biomedical Informatics

University of Thessaly

A thesis submitted for the degree of

*Doctor of Philosophy*

July 2021

## SUPERVISOR

**Dimitris Iakovidis**, *Professor*
*University of Thessaly*

## ADVISORY COMMITTEE MEMBERS

**Dimitris Iakovidis**, *Professor*
*University of Thessaly*

**Vassilis Plagianakos**, *Professor*
*University of Thessaly*

**Konstantinos Delimpasis**, *Associate Professor*
*University of Thessaly*

## REVIEWING COMMITTEE MEMBERS

**Dimitris Iakovidis**, *Professor*
*University of Thessaly*

**Vassilis Plagianakos**, *Professor*
*University of Thessaly*

**Konstantinos Delimpasis**, *Associate Professor*
*University of Thessaly*

**Savelonas Michalis**, *Assistant Professor*
*University of Thessaly*

**Tasoulis Sotirios**, *Assistant Professor*
*University of Thessaly*

**Maglogiannis Ilias**, *Professor*
*University of Piraeus*

**Tsipouras Markos**, *Associate Professor*
*University of Western Macedonia*

# ABSTRACT

This doctoral dissertation explores intelligent systems and services for image and video analysis. In view of scientific challenges for developing innovative solutions with a broad social impact, it investigates applications in biomedicine and computer-assisted navigation of visually impaired individuals. In this context, it focuses on machine learning, particularly the investigation of methods to improve the efficiency and the effectiveness of deep artificial neural network architectures, such as the Convolutional Neural Networks (CNNs).

In Convolutional Neural Networks (CNNs) the input data can contain uncertainties, such as noise, color and geometric ubiquities, that is naturally propagated from the input layer to the convolution layers of the network affecting the quality of the extracted features. To cope with this problem, a novel pooling operation based on (type-1) fuzzy sets is proposed, named Fuzzy Pooling, which can be used as a drop-in replacement of the current, crisp, pooling layers of CNN architectures. Several experiments using publicly available datasets show that the proposed approach can enhance the classification performance of a CNN.

Aiming to improve the effectiveness of CNNs, especially in the context of medical image analysis, a novel architecture named Look Behind Fully Convolutional Neural Network (LB-FCN) is proposed. The architecture is capable of extracting multi-scale image features by using blocks of parallel convolutional layers with different filter sizes. These blocks are connected by look-behind connections, so that the features they produce are combined with features extracted from behind layers, thus preserving the respective information. Furthermore, it has a smaller number of free parameters than conventional CNN architectures, which makes it suitable for training with smaller datasets. This is particularly useful in medical image analysis, since data availability is usually limited, due to ethicolegal constraints. Experiments on publicly available gastrointestinal image datasets show higher classification performance compared to state-of-the-art machine and deep learning methodologies. The architecture is capable of generalizing well even when the training dataset is different than the one on which it is tested. To investigate that, a novel cross-dataset experimental study was performed on various publicly available gastrointestinal tract image datasets, containing images from different modalities, including Wireless Capsule Endoscopy (WCE) and flexible endoscopy.

The number of training samples in CNN training is directly linked to their generalization performance. When the training samples are limited, such as in the case of medical images, the generalization performance is negatively affected. A typical approach to mediate this problem is to use data augmentation techniques, which image rotation and translation. While effective, this technique still requires a substantial amount of training samples to be available. To battle this problem, in the context of inflammatory conditions detection in WCE images, a novel approach is presented that uses Generative Adversarial Networks (GANs) to generate artificial images. More

specifically the study trained two GANs, one to generate healthy small bowel images and another, images with inflammatory conditions. The images are then used to train a CNN architecture and validate its performance in real images. The results from this study show that the substitution of real with artificially generated endoscopic images for CNN training can be a viable option.

While CNNs have a remarkable performance in computer vision problems, usually, they are computationally expensive. This limits their usage in high-end expensive devices with multiple graphical processing units (GPUs). To mediate the problem, a typical approach is to reduce the number of floating-point operations (FLOPs) required for inference, at the expense of generalization performance. In this context, a novel LB-FCN inspired CNN architecture was proposed, named LB-FCN *light*. The architecture features a relatively low number of free parameters and FLOPs, while managing to maintain high generalization performance. The performance of the network is validated in the problem of staircase detection in indoor and outdoor environments, with application on assisted navigation of visually impaired individuals. The results from the experimental evaluation of LB-FCN *light* indicate its advantageous performance over the relevant state-of-the-art architectures.

The development of easy-to-use machine learning (ML) application frameworks has enabled the development of advanced artificial intelligence (AI) applications with only a few lines of self-explanatory code. However, the deployment of ML algorithms as a service for remote high throughput ML task execution, involving complex data-processing pipelines can still be challenging, especially with respect to production ML use cases. To cope with this issue, a novel system architecture is presented, which enables Algorithm-agnostic, Scalable ML (ASML) task execution for high throughput applications. It aims to provide an answer to the research question of how to design and implement an abstraction framework, suitable for the deployment of end-to-end ML pipelines in a generic and standard way. The architecture manages horizontal scaling, task scheduling, reporting, monitoring and execution of multi-client ML tasks using modular, extensible components that abstract the execution details of the underlying algorithms. Applications of ASML are investigated for the analysis of image streams in the context of medical image analysis and assisted navigation of visually impaired individuals. The results of the experiments performed demonstrate its capacity for parallel, mission critical, task execution.

Assistive navigation systems require the development, assessment, and optimization of different algorithms for obstacle detection, recognition, and avoidance, as well as path planning. This is a painstaking and costly process that requires repetitive measurements under stable conditions, which is usually difficult to achieve. To this end, a novel digital twin framework for the simulation and evaluation of assistive navigation systems is presented. The framework can replicate relevant real-life situations, enabling the evaluation and optimization of algorithms through adjustable and cost-effective simulations. The utility and the effectiveness of the framework are demonstrated with an indicative simulation study in the context of a camera-based wearable system for the navigation of visually impaired individuals in an outdoor cultural space.

The work presented in this dissertation includes methods with both theoretical and practical impact, that can be used as the basis for further research, and the applications presented can be used as paradigms for applications on different domains, such as telemedicine, robotics, and intelligent transportation systems.

# ΠΕΡΙΛΗΨΗ

Η παρούσα διδακτορική διατριβή διερευνά πρωτότυπα έξυπνα συστήματα και υπηρεσίες ανάλυσης εικόνας και βίντεο. Λαμβάνοντας υπόψη τις επιστημονικές προκλήσεις για την ανάπτυξη καινοτόμων λύσεων με ευρύ κοινωνικό αντίκτυπο, διερευνά εφαρμογές στη βιοϊατρική και την καθοδήγηση ατόμων με προβλήματα όρασης. Σε αυτό το πλαίσιο, επικεντρώνεται στη μηχανική μάθηση, εστιάζοντας στη διερεύνηση μεθόδων για τη βελτίωση της αποδοτικότητας και αποτελεσματικότητας των αρχιτεκτονικών βαθέων τεχνητών νευρικών δικτύων, όπως τα Συνελικτικά Νευρωνικά Δίκτυα (Convolutional Neural Networks, CNN).

Τα δεδομένα εισόδου των CNN μπορούν να περιέχουν αβεβαιότητες, όπως θόρυβος, χρώμα και γεωμετρική απροσδιοριστία, που μεταδίδονται από το επίπεδο εισόδου στα συνελικτικά επίπεδα του δικτύου επηρεάζοντας την ποιότητα των εξαγόμενων χαρακτηριστικών. Προκειμένου να αντιμετωπιστεί αυτό το πρόβλημα, προτείνεται μια νέα λειτουργία συγκέντρωσης (pooling) βασισμένη σε ασαφή σύνολα (τύπου-1), με όνομα Fuzzy Pooling, η οποία μπορεί να χρησιμοποιηθεί για την αντικατάσταση των υπαρχόντων επιπέδων pooling των CNN αρχιτεκτονικών. Πειράματα σε δημοσίως διαθέσιμα δεδομένα έδειξαν ότι η χρήση της προτεινόμενη προσέγγισης μπορεί να χρησιμοποιηθεί για την βελτίωση της απόδοσης ταξινόμησης των CNN.

Με στόχο τη βελτίωση της αποτελεσματικότητας των CNN, και ειδικότερα στο πλαίσιο της ανάλυσης ιατρικών εικόνων, προτάθηκε μια νέα αρχιτεκτονική CNN που ονομάζεται Look Behind Fully Convolutional Neural Network (LB-FCN). Η αρχιτεκτονική είναι ικανή να εξαγάγει χαρακτηριστικά πολλαπλών κλιμάκων χρησιμοποιώντας σύνολα (μπλοκ) παράλληλων συνελικτικών στρωμάτων με διαφορετικά μεγέθη φίλτρου. Τα σύνολα αυτά, συνδέονται με οπίσθιες συνδέσεις, με στόχο τον συνδυασμό των παραγόμενων χαρακτηριστικών με τα χαρακτηριστικά εισόδου, διατηρώντας έτσι τις αντίστοιχες πληροφορίες. Επιπλέον, η αρχιτεκτονική έχει μικρότερο πλήθος ελεύθερων παραμέτρων σε σχέση με συμβατικές αρχιτεκτονικές CNN, γεγονός που επιτρέπει την εκπαίδευσή της με μικρό πλήθος δεδομένων εκπαίδευσης. Αυτό είναι ιδιαίτερα χρήσιμο στην ανάλυση ιατρικών εικόνας, δεδομένου ότι η διαθεσιμότητα δεδομένων εκπαίδευσης είναι συνήθως περιορισμένη, λόγω βιοηθικών και νομικών περιορισμών. Πειράματα σε δημοσίως διαθέσιμα δεδομένα εικόνων του γαστρεντερικού συστήματος, παρουσιάζουν υψηλή απόδοση ταξινόμησης σε σύγκριση με άλλες σύγχρονες προσεγγίσεις. Η αρχιτεκτονική είναι ικανή να γενικεύει καλά ακόμη και όταν το δεδομένα εκπαίδευσης προέρχονται από διαφορετικά σύνολα δεδομένων από αυτά στα οποίο δοκιμάζεται. Σε αυτό το πλαίσιο, πραγματοποιήθηκε πειραματική μελέτη σε πληθώρα δημοσίων διαθέσιμων συνόλων δεδομένων γαστρεντερικού συστήματος, απαρτιζόμενα από εικόνες που έχουν ληφθεί κάνοντας χρήση διαφορετικών ιατρικών οργάνων, όπως ενδοσκοπικής κάψουλας και εύκαμπτου ενδοσκοπίου.

Η δυνατότητα γενίκευσης των CNN συνδέεται άμεσα με το διαθέσιμο πλήθος δειγμάτων εκπαίδευσης. Όταν τα δείγματα εκπαίδευσης είναι περιορισμένα, όπως στην περίπτωση ιατρικών εικόνων, η δυνατότητα γενίκευσης επηρεάζεται αρνητικά. Μια τυπική προσέγγιση για τον περιορισμό αυτού του προβλήματος είναι η χρήση τεχνικών επαύξησης δεδομένων, τροποποιώντας τα υπάρχοντα δεδομένα. Αν και η τεχνική αυτή είναι αποτελεσματική και πάλι απαιτείται σημαντικό πλήθος δεδομένων εκπαίδευσης. Για την καταπολέμηση αυτού του προβλήματος, στο πλαίσιο της ανίχνευσης φλεγμονών σε εικόνες που προέρχονται από ενδοσκοπική κάψουλα, παρουσιάζεται μια προσέγγιση που χρησιμοποιεί Παραγωγικά Αντιπαραθετικά Δίκτυα (Generative Adversarial Networks, GAN) για τη δημιουργία συνθετικών εικόνων. Πιο συγκεκριμένα, η μελέτη βασίζεται στην εκπαίδευση δύο GAN, ένα για να την παραγωγή υγιών εικόνων του λεπτού εντέρου και ένα άλλο, για την παραγωγή εικόνων με φλεγμονές. Οι παραγόμενες εικόνες στη συνέχεια χρησιμοποιούνται για την εκπαίδευση ενός CNN με στόχο την αξιολόγηση της αποδοτικότητάς του σε πραγματικές εικόνες. Τα αποτελέσματα αυτής της μελέτης δείχνουν ότι η αντικατάσταση πραγματικών με τεχνητά παραγόμενων ενδοσκοπικών εικόνων για εκπαίδευση στο CNN μπορεί να είναι μια βιώσιμη επιλογή.

Η αξιοσημείωτη απόδοση των CNN στον τομέα της υπολογιστικής όρασης, συνήθως, συνοδεύεται από αυξημένο υπολογιστικό κόστος. Αυτό περιορίζει τη χρήση τους σε συσκευές υψηλών υπολογιστικών προδιαγραφών εξοπλισμένες με πολλαπλές κάρτες γραφικών. Για την αντιμετώπιση αυτού του προβλήματος, μια τυπική προσέγγιση είναι η μείωση των απαιτούμενων αριθμητικών πράξεων, σε βάρος της απόδοσης γενίκευσης. Σε αυτό το πλαίσιο, προτάθηκε μια νέα αρχιτεκτονική CNN, εμπνευσμένη από την LB-FCN, με όνομα LB-FCN *light*. Η αρχιτεκτονική διαθέτει χαμηλό αριθμό ελεύθερων παραμέτρων και πράξεων, ενώ παράλληλα διατηρεί υψηλή απόδοση γενίκευσης. Η απόδοση του δικτύου διερευνήθηκε στο πρόβλημα της ανίχνευσης σκαλών σε εσωτερικούς και εξωτερικούς χώρους, με εφαρμογές στην υποβοηθούμενη πλοήγηση ατόμων με προβλήματα όρασης. Τα αποτελέσματα από την πειραματική αξιολόγηση του LB-FCN *light* δείχνουν πως απόδοσή του είναι υψηλότερη σε σύγκριση με άλλες, σύγχρονες αρχιτεκτονικές CNNs.

Η ανάπτυξη εύχρηστων πλαισίων εφαρμογών μηχανικής μάθησης, δίνει την δυνατότητα ανάπτυξης προηγμένων εφαρμογών τεχνητής νοημοσύνης με μόνο λίγες γραμμές κώδικα. Ωστόσο, η εγκατάσταση αλγορίθμων μηχανικής μάθησης σε απομακρυσμένο περιβάλλον υψηλής απόδοσης, που περιλαμβάνει περίπλοκα επίπεδα επεξεργασίας δεδομένων, εξακολουθεί να είναι δύσκολη, ειδικά όταν τα περιβάλλοντα αυτά προορίζονται για χρήση από επιχειρήσεις. Για την αντιμετώπιση αυτού του προβλήματος, παρουσιάζεται μια νέα αρχιτεκτονική συστήματος, η οποία επιτρέπει την εκτέλεση εργασιών μηχανικής μάθησης για εφαρμογές υψηλής απόδοσης, με όνομα Algorithm-agnostic, Scalable Machine Learning (ASML). Στόχος της αρχιτεκτονικής είναι να δώσει μια απάντηση στο ερευνητικό πρόβλημα της σχεδίας και ανάπτυξης πλαισίου εφαρμογής, κατάλληλο για την ανάπτυξη διεργασιών μηχανικής μάθησης με γενικό και

τυποποιημένο τρόπο, ανεξάρτητο του αλγορίθμου μηχανικής μάθησης. Η αρχιτεκτονική διαχειρίζεται την οριζόντια κλιμάκωση, τον προγραμματισμό εργασιών, την αναφορά, την παρακολούθηση και την εκτέλεση εργασιών μηχανικής μάθησης, με δυνατότητα χρήσης από πολλαπλούς χρήστες, χρησιμοποιώντας ανεξάρτητα και επεκτάσιμα στοιχεία που αποκρύπτουν τις λεπτομέρειες εκτέλεσης των υποκείμενων αλγορίθμων. Η δυνατότητες της αρχιτεκτονικής διερευνήθηκαν σε εφαρμογές ανάλυσης ροών εικόνων από ιατρικά δεδομένα και στα πλαίσια της υποβοηθούμενης πλοήγηση ατόμων με προβλήματα όρασης. Τα αποτελέσματα των πειραμάτων που πραγματοποιήθηκαν δείχνουν ότι η αρχιτεκτονική είναι κατάλληλη για παράλληλη χρήση και σε κρίσιμα συστήματα.

Τα συστήματα υποβοηθούμενης πλοήγησης απαιτούν την ανάπτυξη, αξιολόγηση και βελτιστοποίηση διαφορετικών αλγορίθμων για την ανίχνευση εμποδίων, την αναγνώριση και την αποφυγή τους, καθώς και τον σχεδιασμό διαδρομών. Η διαδικασία αυτή είναι ιδιαιτέρως επίπονη και δαπανηρή και απαιτεί επαναλαμβανόμενες μετρήσεις υπό σταθερές συνθήκες, κάτι που συνήθως είναι δύσκολο να επιτευχθεί. Για το σκοπό αυτό, παρουσιάζεται ένα πρωτότυπο πλαίσιο εφαρμογής για την προσομοίωση και την αξιολόγηση συστημάτων υποβοήθησης πλοήγησης. Το πλαίσιο αυτό μπορεί να αναπαράγει πραγματικές καταστάσεις, επιτρέποντας την αξιολόγηση και βελτιστοποίηση αλγορίθμων μέσω ρυθμιζόμενων και οικονομικά αποδοτικών προσομοιώσεων. Η χρησιμότητα και η αποτελεσματικότητα του πλαισίου αποδεικνύονται με μια ενδεικτική μελέτη προσομοίωσης στο πλαίσιο ενός φορητού συστήματος που βασίζεται σε κάμερα για την πλοήγηση ατόμων με προβλήματα όρασης σε έναν υπαίθριο χώρο πολιτιστικού ενδιαφέροντος.

Το έργο που παρουσιάστηκε στην παρούσα διατριβή περιλαμβάνει μεθόδους με θεωρητικό και πρακτικό αντίκτυπο, οι οποίες μπορούν να χρησιμοποιηθούν ως βάση για περαιτέρω έρευνα. Οι εφαρμογές που παρουσιάζονται μπορούν να χρησιμοποιηθούν ως πρότυπα για εφαρμογές σε διαφορετικούς τομείς, όπως τηλεϊατρική, ρομποτική και έξυπνα συστήματα μετακίνησης.

*This thesis is dedicated to my family and friends.*

# ACKNOWLEDGMENTS

The pathway for a doctoral degree is full of challenges and obstacles. In 1911, Constantine Peter Cavafy wrote the poem “Ithaca”. One can say that the quote “When you depart for Ithaca, wish for the road to be long, full of adventure, full of knowledge…”, closely resembles the doctoral degree pathway. Crossing it I was lucky enough to meet great people, colleagues, who I now consider friends. What a great outcome it is.

Firstly, and foremost I would like to thank my parents, who were there for me throughout my studies, supporting and endorsing me to pursue my dreams, undercoating me with their protective umbrella.

I would also like to thank the supervisor of this dissertation, Dr. Dimitris K. Iakovidis, for his valuable contribution to my studies. The fruitful discussions, advice and optimism greatly contributed throughout this doctoral dissertation. He is an excellent example of a knowledgeable, hardworking researcher, who I was honored to meet and work alongside him.

I would like to thank my colleagues in the lab. Especially I would like to thank Dr. Michael Vasilakakis, Mr. George Dimas, Ms. Dimitra-Christina Koutsiou, Dr. Panagiotis Kalozoumis and Dr. Charis Ntakolia, who throughout this thesis, they were always there to help and encourage me. I am honored to now consider them good friends.

Finally, I would like to express my gratitude to the Onassis Foundation for supporting this doctoral dissertation through a scholarship.

# TABLE OF CONTENTS

# LIST OF FIGURES

# LIST OF TABLES

# CHAPTER 1

# INTRODUCTION

This chapter introduces the notions investigated in this dissertation, it describes the research context of the performed investigation, its aims, and its novel scientific contributions. It includes references to the publications produced, as well as a structure of this document.

## 1.1 Introduction

Nowadays, intelligent systems based on Artificial Neural Networks (ANNs) are flourishing in the context of multidimensional signal analysis, especially for the analysis of images and videos. ANNs are Machine Learning (ML) systems that simulate the biological neural networks of living organisms. They are used in a variety of applications that include pattern recognition, *e.g.*, in the identification of persons, objects, etc., and the solution of forecasting problems based on previous observations, *e.g.*, risk prediction from time series data. The increased computational power of modern computer systems, especially the use of Graphical Processing Units (GPUs), enabled Deep Learning (DL), a contemporary machine learning paradigm based on Deep Neural Network (DNN) architectures, which are ANNs with multiple hidden layers of artificial neurons. DNNs are increasingly becoming more efficient in recognizing patterns in large volume data (big data), but also in solving complex prediction problems (Gu et al. 2018).

Conventional machine learning systems, such as Support Vector Machines (SVMs) (Vapnik 2013) and ANNs, require a data pre-processing step in which typically features are extracted from the data, or selected from existing features, and used as input to the model. This pre-processing step typically requires a domain expert and aim to represent specific characteristics of the input that are of interest in the context of an application. For this reason, these features are typically referred as "hand-crafted". In computer vision, such features mainly include color, shape, and textural information. In deep learning, this data pre-processing step is eliminated, as the feature extraction is automated through training, which removes some of the dependency on domain expert. Such models ingest the entire data, instead of features, from which multiple levels of features are extracted. Convolutional Neural Networks (CNNs) are a representative example of DL with high efficiency (LeCun et al. 2015; Ravi et al. 2016).

Since 2012 (Krizhevsky et al. 2012), CNNs have revolutionized the domain of Computer Vision (CV) and nowadays are considered the de-facto choice for image analysis tasks. Applications of CNNs are numerous (Liu et al. 2017) . In this dissertation, motivated from the challenges posed by the projects funding the respective research, more attention has been given to applications related to the recognition and detection of objects in natural, assistive navigation systems for visually impaired people, and medical image analysis **(Appendix A)**.

Another area significantly impacted by DL are the generative models. Generative Adversarial Networks (GANs) (Goodfellow et al. 2014) and Adversarial Auto Encoders (AE) (Makhzani et al. 2015) are examples of generative models with a remarkable performance in generating realistic images. Applications of such models include, image generation from random noise (Goodfellow et al. 2014), image-to-image translations (Isola et al. 2017), super-resolution (Ledig et al. 2017) and realistic medical image generation (Kazeminia et al. 2020).

Nowadays most of the population of developed countries use cameras on a daily basis, *e.g.*, through a mobile smartphone device. The increase in computational power of such portable devices enabled the development of smart applications implementing elements of artificial intelligence. Since 2014 (Jin et al. 2014), there has been an increased research interest towards the reduction of the computational resource requirements of DL models, such as CNNs, to fit their limited hardware requirements. To achieve that, most "mobile-oriented" models, primarily focus on ways to reduce the free parameters of these models and consequently the number of floating-point operations (FLOPs) required for inference. In effect, such models (Howard et al. 2017) usually sacrifice the generalization capabilities of the model in favor of computational performance.

This dissertation investigates DL models with enhanced generalization performance, even when the samples available for training these models are limited, methodologies that can reduce the computational requirements of such models to enable their use in embedded devices, and methodologies to provide ML remotely as a service. The results of this dissertation show that the proposed methodologies can be successfully used in a variety of different domains, including medical image analysis for clinical decision support, navigation systems to assist visually impaired people navigate in unknown environments, and software as a service deployment of complex ML pipelines, such as the pipelines required in such applications, which enable the use of ML in a cost-efficient and production-ready way.

## 1.2 Aims of this Dissertation

This doctoral research investigates novel approaches to ML with focus in deep learning and their applications in computer-aided assistive technologies. The directions in which the research focuses have broad social impact and can be summarized as:

- Investigate novel deep learning methodologies and develop DNN architectures and methods offering improved generalization and computational performance.

- Investigate image generation methodologies with applications in biomedical image synthesis.
- Investigate methodologies that enable effective DNN training even when the data availability is limited.
- Investigate methods that enable ML algorithms to be deployed in scalable production environments, and offered through remote intelligent services.
- Investigate applications in the context of medical image analysis and computer-assisted navigation systems.
- Investigate frameworks that enable *in silico* testing and evaluation of assistive navigation systems.

## 1.3 Thesis Contributions

The effort invested for the accomplishment of the aforementioned aims, resulted in the development of novel DNN architectures, methodologies, and applications:

- An image pooling methodology based on fuzzy logic, named Fuzzy Pooling, which aims to cope with the local imprecision of the feature maps produced by the CNNs.
- A CNN architecture, named Look-Behind Fully Convolutional Neural Network (LB-FCN), that can generalize well, even when the availability of training data is limited.
- A lightweight CNN architecture, named LB-FCN *light*, with relatively low computational footprint and high generalization capabilities, designed for mobile and embedded applications.
- An algorithm-agnostic architecture for scalable machine learning (ASML), enabling the implementation of, even real-time, remote ML services.
- A GAN-based image synthesis methodology that enables substitution of real with artificially generated endoscopic images for CNN training.
- Applications of LB-FCN and other CNN architectures in the context of medical image analysis, including cross-dataset abnormality detection experiments and multi-label classification on wireless capsule endoscopy (WCE) images.
- Applications of LB-FCN *light* in the context of obstacle detection for computer-assisted navigation.
- Applications of ASML in the context of computer-aided endoscopy and computer-assisted navigation.
- A digital twin framework, designed for the simulation and evaluation of assistive navigation systems.

The research and development in the scope of this doctoral thesis has been accepted for publication in *six (6) international journals,* and *two (2) book chapters*, and have been presented in *three (3) international conferences*. The list of publications is provided in **Appendix A**.

## 1.4 Thesis Outline

The rest of this thesis is organized in six (6) chapters:

- Chapter 2 provides the necessary theoretical background to artificial neural networks, focusing on CNNs and GANs. It includes a detailed literature review.
- Chapter 3 presents the proposed fuzzy pooling methodology.
- Chapter 4 is dedicated to machine learning in the context of computer-aided endoscopy. It includes literature review along with novel contributions of CNN and GAN architectures that contribute to the aims of this study.
- Chapter 5 investigates machine learning in the context of computer-assisted navigation and includes literature review of assistive navigation methodologies along with contributions that enable the development of such methods.
- Chapter 6 presents a novel algorithm-agnostic system architecture that enables scalable machine learning.
- Chapter 7 is the last chapter, where the conclusions and future prospects of further research are summarized.

# CHAPTER 2

# ARTIFICIAL NEURAL NETWORKS

This chapter aims to introduce the reader to the concepts of ML with primary focus on Deep Learning techniques. It provides a brief introduction in the concept of Artificial Neural Networks (ANNs), their usage and where their source of inspiration came from. Deep Learning is examined in detail with primary focus on a special kind of deep ANN architecture named Convolutional Neural Networks (CNNs) and their application in CV problems. The chapter also includes an introduction to Generative Adversarial Networks (GANs) along with a review of the state-of-the-art methods in the context of artificial image generation.

## 2.1 Introduction

It is well known that the human brain contains billions of neurons **(Figure 2.1)** that are connected between each other via synapses and that these neurons are acting together in parallel and are responsible for our perception.

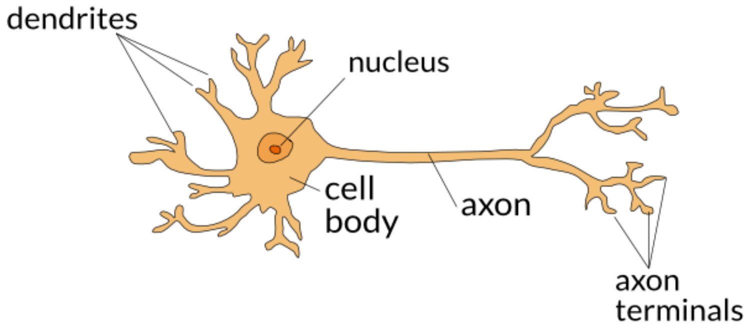


**Figure 2.1** A stripped down human brain neuron. Dendrites reassemble the input of the cell body (neuron), nucleus the computational unit and axon the output of the neuron.

ANNs are inspired by the biological neurons and try to mimic the way human brain works. ANNs are a simplified simulation of the human brain neurons connected with synapses forming a graph. The basic computational unit firstly presented in late 50's is called perceptron (Kubat 1999) and it was inspired by the earlier work of (McCulloch & Pitts 1943). A basic perceptron is illustrated in **(Figure 2.2)**. It can be noticed that a perceptron is a direct translation of brain neuron into a computational unit, with dendrites replaced by weights $W_i$, nucleus with an input and a weight sum, known as transfer function, and the axon with an activation function $\varphi$.

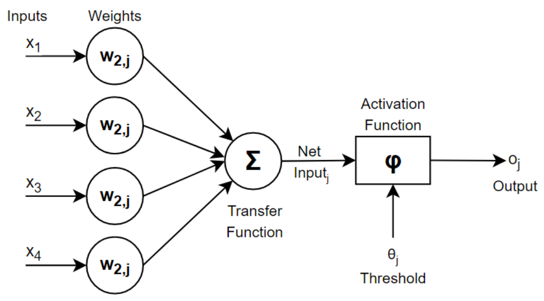


**Figure 2.2** A graphical representation of a perceptron.

Essentially a perceptron computes the weight sum of its inputs which are then passed by an activation function. This produces a signal if a threshold value is reached. This can be expressed as:

$$y = \varphi\left(\sum_i x_i w_i\right) \tag{2.1}$$

Examining (2.1), one can notice that a single perceptron is a basic linear binary classifier (**Figure 2.3**) that can be expressed as:

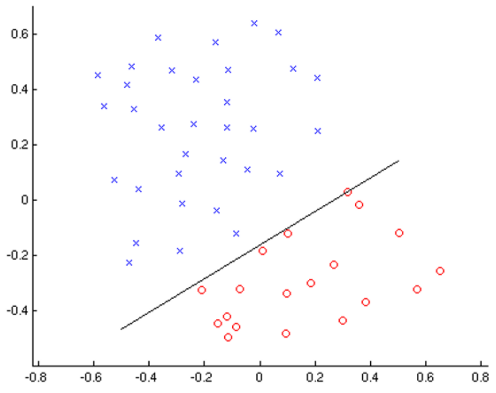
$$f(x) = \begin{cases} 1, if\ w \cdot x + b > 0 \\ 0, otherwise \end{cases} \tag{2.2}$$

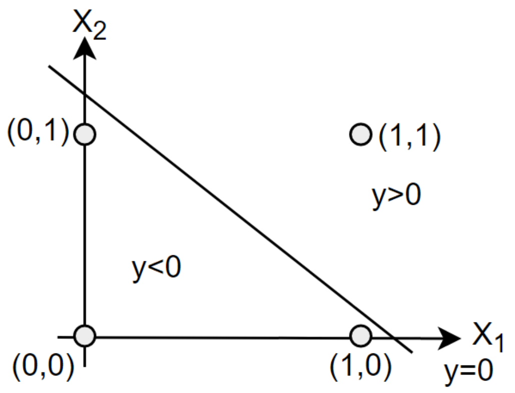
The $w \cdot x$ is the dot product of the input $x$ and weights $w$ matrices, respectively. The $b$, known as bias, is an independent parameter which helps the decision boundary to move away from the origin. Neurons with large bias, can be activated easily while neurons with negative bias can impact the perceptron to activate harder.

An illustration of the effectiveness of a single perceptron model can be observed by modeling a NAND gate **(Figure 2.3b)**. The behavior of NAND logical gate is shown in **Table 2.1** and can be expressed as perceptron with two inputs, a bias and a threshold:

$$f(x_1, x_2) = \begin{cases} 1, if\ -2 \cdot x_1 \cdot -2 \cdot x_2 + 3 > 0 \\ 0, otherwise \end{cases} \tag{2.3}$$

where $w_1 = w_2 = -2, b = 3$.


(a)



(b)

**Figure 2.3** (a) A generic linear classifier following $f(x) = w \cdot x + b$. (b) The NAND gate input space.

**Table 2.1** The NAND gate behavior

| $x_1$ | $x_2$ | $y$ |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

### 2.1.1 Perceptron Training

In simple cases, where the function is linearly separable, it is possible to tune the weights and the threshold of the perceptron to output the desired values. In more complex scenarios, like the XOR logical gate, more perceptrons are required to achieve the desired output. The question that arises, is, when there are multiple perceptrons, how to compute, algorithmically those free parameters?

A simple yet computationally wasteful approach, to calculate the weights and thresholds of a perceptron would be to follow an exhaustive search approach. As there is no guarantee that this approach would lead to a desirable output within reasonable time, the need for an algorithmic automation arises. There are multiple ways of computing these free parameters. The process of finding these parameters, is called "training" of a neural network.

There are several methodologies to train a neural network such as the error back-propagation (Rumelhart et al. 1986), which will be discussed in the following subsection, yet a simple example for perceptron training  (Minsky & Papert 1988) is beneficial as it presents the basic principles behind training without the complications of a more complex approach.  In the training phase of a perceptron, a set of known examples called "training dataset" is presented to the network, multiple times. Each time the training dataset is presented, the free parameters are adjusted, according to a training rule. This process is called supervised training, as for all the samples included in the training dataset, the input parameters and the desired output is known.

Let $x_i$ be the input parameters and $w_i$ the weights. Let $T$ be the training dataset, defined as a set $T = \{t_1, t_2 \dots t_i\}$ which contain sample vectors $t_i = \{x_1, x_2 \dots x_i, y\}$ where $y$ is desired output. For simplicity, the weights are combined with bias and the inputs of the model can be combined as a matrix $W$ and $X$ respectively (2.4).

$$W = \begin{bmatrix} w_1 \\ w_2 \\ \dots \\ w_i \\ b \end{bmatrix}, X = \begin{bmatrix} x_1 \\ x_2 \\ \dots \\ x_i \end{bmatrix} \tag{2.4}$$

The result of the dot product between the two matrices (2.4) is the output of the neuron $z$:

$$z = w \cdot x \tag{2.5}$$

Training is achieved by adjusting the weight vector $W$ according to the distance between the output $z$ of the perceptron and the desired output $y$ presented in the training vector $t_i$:

$$w' = w \pm n \cdot t \tag{2.6}$$

Parameter $n$ is called learning rate, and is a small number, typically within $[0,1]$, and it defines the speed in which the weights will be adjusted in each iteration. Having a large learning rate, can result into unstable learning while having too small, can lead to increased learning time. When training, if $z$ is approaches the desired $y$, the term $n \cdot y$ will be a positive value, and thus $w' = w + n \cdot y$. On the other hand, if the desired $y$ is moving away then $w' = w - n \cdot y$. The learning rule derived from this can be expressed as:

$$w' = w + n(y - z) \cdot w \tag{2.7}$$

The Eq. (2.7) can now be rewritten as a desired weight change $\Delta w = w' - w$

$$\Delta w = n(y - z) \cdot t \tag{2.8}$$

This equation originally introduced by (Van Der Malsburg 1986) was historically the first used to train logical unit and is called the "Perceptron Rule". The perceptron rule can be algorithmically expressed as:

**Algorithm 2.1** The perceptron rule algorithm (Van Der Malsburg 1986)

```
1:  while output != $y$ do
2:     foreach $t_i$ in $T$:
3:        output = evaluate model using $t_i$
4:        if output != $y$ then
5:           $w = w'$ according to (2.7)
6:        end if
7:     end foreach
8:  end while
```

Although variations of the perceptron rule were introduced by (Minsky & Papert 1988) and later by (Kubat 1999), the main obstacle of how the above algorithm could generalize into training multiple perceptrons remained unsolved, effectively halting the neuroscience research for nearly 15 years, placing neural networks power in question.

### 2.1.2 The Activation Function

An important component of an artificial neuron is the activation function, as it defines the way that the function behaves based on different input. Without an activation function the network would not be able to approximate complex, non-linear functional mapping between the input and the output. The purpose of an activation function is to translate an input signal of a neuron in a network to an output signal, usually within a specified range of values.

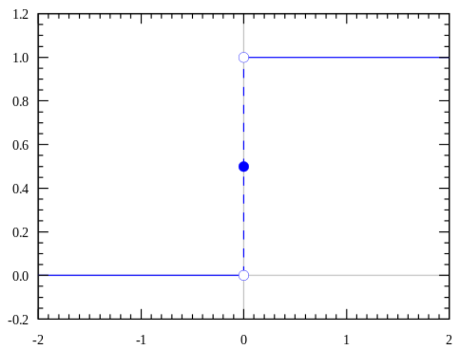
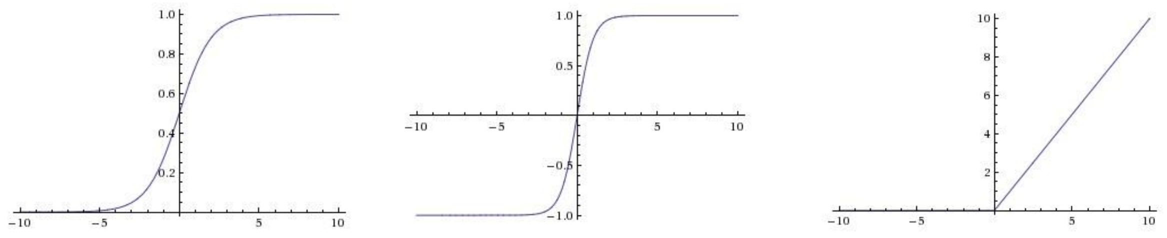


**Figure 2.4** The step activation function

In the example of the pervious section, the step activation function **(Figure 2.4)** was used, which activates whenever the input signal is greater than 0. Although computationally simple, it has major disadvantage when used in a perceptron; small changes to the input space can significantly alter the output of the neuron. An alternative function that enables smaller changes to the output of the neuron according to the input value is desired. There are many functions that fulfill this property, with the most commonly used one been the Logistic function **(Figure 2.5a)**.



(a) Logistic $a(x) = 1/\left(1 + e^{-\beta x}\right)$

(b) Tanh $a(x) = \dfrac{1 - e^{-2\beta x}}{1 + e^{-2\beta x}}$

(c) Rectified Linear Units (ReLU) $a(x) = \max(0, x)$

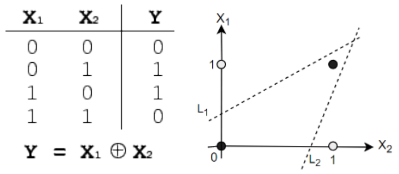
**Figure 2.5** Commonly used activation functions.

The logistic non-linear function squashes the input value between 0 and 1 and has the advantage of smoothing the output according to the change of the provided input. While effective, it can also lead to computational mathematical loss as small real numbers are produced. When $\beta \rightarrow \infty$ the logistic function becomes has the same properties as the step function. Similarly, the tanh function **(Figure 2.5b)**, squashes the values between -1 and 1, having the advantage of overcoming the problem of non zero-centered values. Unfortunately, tanh is usually avoided as it can lead to saturation problems while training, especially on large neural networks. Both logistic and tanh function belong to the same family of activation functions commonly known as “Sigmoid”. Recently, the Rectified Linear Units (ReLU) activation **(Figure 2.5c)** has been proposed (Nair & Hinton 2010). The ReLU function battles the problem of neuron saturation and is relatively computationally simple. These two properties are especially useful in deep neural networks, where the problem vanishing gradient is more apparent. Another benefit of ReLU activations, is the sparsity that arises when $x \leq 0$, while sigmoid functions tend to generate small non-zero values leading to dense representations. While ReLU activation can lead to numeric explosions, newer versions of the function have been proposed, such as capped ReLU (Howard et al. 2017) $a(x) = \max(0, \min(\mathrm{x}, \mathrm{c}))$, where c is the max value, deal with this problem. Although there are not enough mathematical evidence to prove that ReLU activation function performs better the typical Sigmoid functions, the non-saturating properties of the function have empirically proven that can improve the training performance (Krizhevsky et al. 2012).

Neurons with sigmoid functions as activations are called “sigmoid neuron” and, generally, neurons are typically named after the activation function that is used. In literature though, the perceptron term has been widely adopted to name any sort of artificial neuron. For historical reasons, neurons are named by the by the widely used naming convention “perceptron” and the terms “sigmoid” or “ReLU” neurons, or other, are only used when emphasis is required on their activation function.

## 2.1.2 Feed Forward Networks and Multilayer Perceptron

Although a single perceptron can be used to model a simple, linear, binary classifier it is unable to model more complex functions such as the XOR logical function. The XOR logical function is non-linear and computes the eXclusive OR logical operation. The non-linearity of the XOR function can be observed in **(Figure 2.6)**.



| $X_1$ | $X_2$ | $Y$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

$Y = X_1 \oplus X_2$

**Figure 2.6** Visual representation of the XOR logical function with the truth table on the left and the function plot on the right.

Observing **(Figure 2.6)**, it can be noticed that no single line is able separate the XOR two-dimensional space. As a result, the need of extending the single-neural model arises. By including more neurons effectively, we can incorporate more binary classifiers; forming a network of neurons, that can approximate a desired function.

The Multilayer Perceptron networks architecture (MLPs) is an architecture that extends the perceptron theory by incorporating multiple neuron (perceptron) units in layers, each one connected with the next layer, in a fully connected feed forward neuron connection architecture. An MLP network has a set of inputs, forming the so-called input layer (unlike other layers, it does not include any neurons), at least one intermediate layer and an output layer. The intermediate layer is also called “hidden” layer. A typical MLP architecture with two hidden layers is illustrated in **(Figure 2.7)**. It can be noticed that the neurons of each layer of the architecture is connected with all the neurons of the next layer, thus the term fully-connected network. This means that in an MLP architecture with 10 inputs, 1 hidden layer of 20 neurons and 3 output neurons will result into $10 \cdot 20 + 20 \cdot 3 = 260$ weights and 23 biases that need to be optimized.

MLPs belong to a wider range of neural network architectures known as “Multi-layer Feed-Forward Neural Networks” (MFNNs). The naming of feed forward networks reflects their design, which enables the signal flowing towards one direction, from the input neurons throughout the network to the output without any backward connections. This signal propagation ensures a synchronous behavior in which the signal is propagated by one neuron at a time interval, without “delays” or “accumulation” of signals, which although it contradicts the complex connectivity of the human brain neurons, it has proved relatively effective of solving both classification and regression problems (Cybenko 1989; Hornik et al. 1989).

It has been shown (Cybenko 1989; Hornik et al. 1989) that an MLP with at least one hidden layer is able to simulate any kind of continue function. This is also known as the Universal Approximation Theorem. The universality of MLP architectures can also be extended to discontinuous functions, if the nature of the problem can accept an approximated solution. Although based on this theorem, any MLP with a single hidden layer and enough neurons can be used to approximate any function, in practice, architectures with multiple hidden layers are more common. This is because, by limiting an MLP architecture to a single hidden layer, results into increased number of neurons, which introduce more free parameters to the system and thus, increase the overall complexity. Using more hidden layers, the approximation load is shared along among the layers resulting into reduced weights and biases that have to be computed.

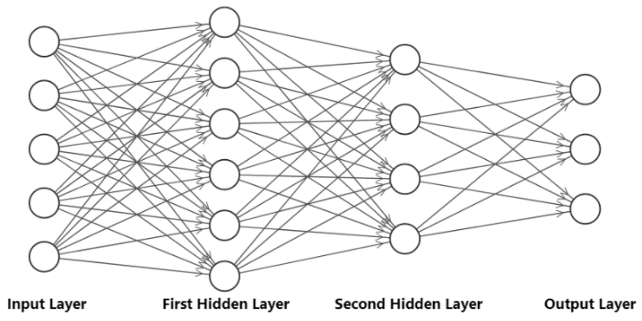


**Figure 2.7** A visual representation of an MLP architecture with two hidden layers, each one of arbitrary number of neurons, and three output neurons.

The number of hidden layers is another hyper-parameter that needs to be considered, when designing an MLP architecture. Unfortunately, there is no mathematical way of computing the number of hidden layers and their corresponding number of neurons. For this reason, heuristic approaches are usually employed, which are based on trial and error, such as training the network and validating its performance in unknown data. If the network is not capable of generalizing, different hyper-parameters are tested. The changes include both the number of hidden layers and the corresponding number of neurons of each layer (Murata et al. 1994; Bengio et al. 2007).

A simple example of an MLP architecture that is capable of modeling the XOR gate **(Figure 2.6)** is illustrated in **(Figure 2.8)**. The network consists of one input layer with two units, $x_1$ and $x_2$ which are connected to two-neuron, $z_1$ and $z_2$, hidden layer the output of which is connected to a single output neuron $Y$.

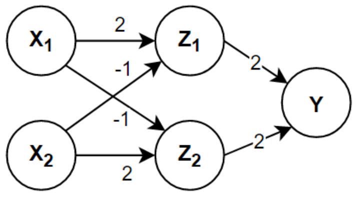


**Figure 2.8** A neural network capable of modeling the XOR logical function.

The input layer with the $x_1$ and $x_2$ neurons is considered fixed in terms that thee output of its neurons is the same value as their input which explains their naming as “units”. The architecture contains six weights and 3 biases. These parameters are also known as “free-parameters”, which need to be fine-tuned, through training in order for the network to estimate the desired function. Adding more neurons, the number of free-parameters also increases along with the computational complexity of the overall model. In return, the network can estimate more complex functions. For the XOR example the parameters can be computed by hand, yet for more complex scenarios, with thousands or even millions of neurons, the need of an automated parameter computation becomes a necessity.

## 2.2 Neural Network Training

Training of neural networks is an actively researched subject. Many methodologies have been proposed over the years, which can be grouped into two main categories; evolutionary algorithms (Jones 1993; Montana 1995) and  error back-propagation (Hecht-Nielsen 1992). Both approaches can be used to effectively train neural networks (Gupta & Sexton 1999), yet the error back-propagation based algorithms have been the de-facto choice in literature, mainly due to its effectiveness on training large networks.

### 2.2.1 Gradient Descent Optimization Algorithms

In literature, numerus gradient descent optimization algorithms have been proposed over the years (Ruder 2016). The goal of these algorithms is to minimize the error of the output of the neural network by adjusting the weights and biases of the model. A visual representation of gradient descent in a three-dimensional space of an upside-down cliff containing a ball is illustrated in **(Figure 2.9)**, where the goal of the algorithm is to guide the ball towards the lowest point of the cliff, which is known as global minima.

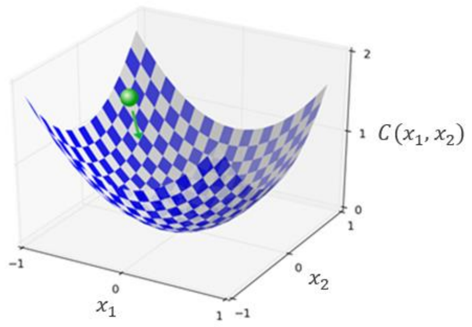


**Figure 2.9** A 3D representation of upside-down cliff and a ball following a down-hill direction towards the cliff global minimum.

### 2.2.1.1 The Loss Function

To train an ANN, there is a need to quantify the output of performance. For this reason, a function is used, named loss function, to quantify the error produced by the network comparing to the expected output. In literature the name "loss function" can also be found as "cost", "optimization" or "error" function. The goal of this function is to be minimized through a training algorithm by adjusting the free parameters of the network. A commonly used loss function, especially on classification problems is the mean square error (MSE), which is also known as "quadratic function" and is defined as:

$$C(w,b,x,y) = \frac{1}{2n}\sum_{x}|y(x) - y'|^2 \tag{2.4}$$

The $w$ and $b$ parameters are the free-parameters of the network, weights and biases respectively, $x$ the input of the network, $y(x)$ the expected output (target) of the network and $y'$ the output of the network. The $n$ parameter represents the total number of samples presented to the model. It can be noticed that the sum of the errors is always a positive real number, and it approximates $C(w,b) \cong 0$ only when $y(x) \cong y'$. This observation shows that the function can be used to minimize the error, between the output of the network and the expected value. There are multiple loss functions used in the literature (Rosasco et al. 2004), including cross entropy, root mean square error etc. which can be used depending on the goal and the optimization algorithm that is used to optimize them.

### 2.2.1.2 The Gradient Descent Algorithm

One of the simplest yet powerful optimization algorithms used to train an ANN, is the gradient descent algorithm. This iterative algorithm tries to minimize the error of a loss function $C$ by adjusting the free parameters of the network in small steps, upon every iteration of all training samples. As it is the basis of many optimization algorithms it is worth understanding how it operates. Let $f$ be a function with $n$ parameters that is minimized through a loss function $C$. Let $\Delta x_i$ be a small change in the direction of the function that is applied to $x_i$ original directions. Following the mathematical calculus, the change of $C$ can be obtained following Eq. (2.5) which has as a goal to obtain a negative $\Delta C$ and thus minimize $f$.

$$\Delta C \cong \frac{\vartheta C}{\vartheta x_1}\Delta x_1 + \cdots + \frac{\vartheta C}{\vartheta x_i}\Delta x_i \tag{2.5}$$

Let $\Delta x$ be the transposed matrix of the variables and thus $\Delta x = (\Delta x_1, \dots, \Delta x_i)^T$ with the gradient of $C$ defined as:

$$\nabla C = \left(\frac{\vartheta C}{\vartheta x_1}, \dots, \frac{\vartheta C}{\vartheta x_i}\right)^T \tag{2.6}$$

We can rewrite Eq. (2.6) with respect of $\Delta x$ and $\nabla C$:

$$\Delta C \cong \nabla C \cdot \Delta x \tag{2.7}$$

To minimize $\Delta C$ of Eq. (2.7) is possible by introducing a parameter $n$, which is called learning rate:

$$\Delta x = -n\nabla C \tag{2.8}$$

The learning rate $n$, is a small positive real number. Based on Eq. (2.8) this can be written as:

$$\Delta C \cong -n\nabla C^2 \tag{2.9}$$

As $\nabla C^2$ is always positive number, $\Delta C \leq 0$ is guaranteed. The vector $x$ update can be expressed as Eq. (2.10). Repeating this update for certain number of iterations, the change in the parameters of the function $f$ will reach to global minimum.

$$x \rightarrow x' = x - n\nabla C \tag{2.10}$$

Gradient descent can be used to train a neural network directly, by estimating the best parameters which the selected loss function $C$ is dependent on. Considering the neural network as a function with parameters been the weights and biases of the neurons, we can rewrite Eq. (2.10) as:

$$w \rightarrow w' = w - n\frac{\vartheta C}{\vartheta w} \tag{2.11}$$

$$b \rightarrow b' = b - n\nabla C = b - \frac{\vartheta C}{\vartheta b} \tag{2.12}$$

There are challenges of using gradient descent on large number of free parameters, as the goal of it is to minimize the quadratic loss function $C$ Eq. (2.4). As a result, to compute the gradient $\nabla C$ we also need to compute the $\nabla C_x$ for each training sample as the function is an average computed by $C_x = \frac{|y(x)-a|^2}{2}$ and $\nabla C = \frac{1}{n}\sum_x \nabla C_x$. This is a computationally expensive task, both in terms of time for large number of training examples and the learning, as the minimization of $C$, will occur slowly.

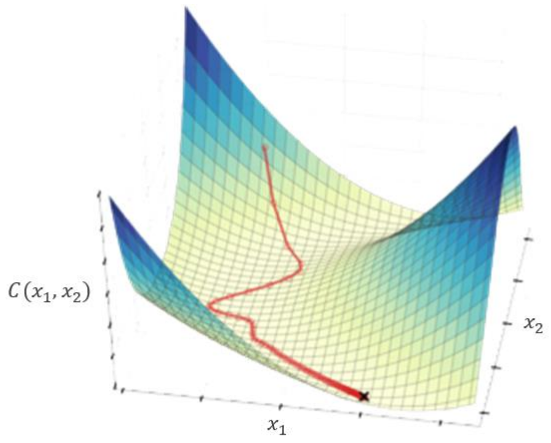


**Figure 2.10** A 3D representation of a loss function minimization. The red line represents the value been minimized over a series of iterations, reaching the global minima of the function (denoted with black ×).

Although the goal of optimizing the loss function is to reach a global minimum, in most cases, and especially in complex optimization problems such as, training an ANN with million free parameters, that is not always the case. Local minima and vanishing gradient are some of the most common problems found in such optimizations, that can lead to poor generalization performance or the network to never converge. State-of-the-art optimization algorithms have been proposed that mitigate this problem to a large extend, but without guarantees that global minima will be found.

#### 2.2.1.3 The Stochastic Gradient Descent Algorithm

To mitigate the slow learning rate of the conventional gradient descent algorithm, stochastic gradient descent (SGD) was introduced. The algorithm operates by randomly selecting a subset $m$ of training samples from the training set $T$ and uses that to perform the normal gradient descent. This subset is also known as “batch”. The principle behind the SGD algorithm is that instead of computing the parameter change based on the entire dataset it estimates the change based on smaller sample of data. As a result, more changes in the free parameters of the network are applied per iteration (epoch) of the entire dataset which in return speeds up the converge time significantly.

The SGD algorithm is very similar to conventional gradient descent. Let mini-batch $M = \{X_1, X_2 \dots, X_m\}$ of the training dataset $T$, where $M \subseteq T$. The gradient descent is estimated by averaging $\nabla C_{X_i}$:

$$\nabla C = \frac{\sum_x \nabla C_x}{n} \cong \frac{\sum_i^m \nabla C_{X_i}}{m} \tag{2.13}$$

To apply Eq. (2.13) in neural network training, an estimation of $\nabla C_w$ and $\nabla C_b$ for the weights and biases of the network is obtained as:

$$w \to w' = w - \frac{n}{m} \sum_i \frac{\vartheta C X_i}{\vartheta w} \tag{2.14}$$

$$b \to b' = b - \frac{n}{m} \sum_i \frac{\vartheta C X_i}{\vartheta b} \tag{2.15}$$

Symbol $\sum_i$ represents the summing of all the mini-batch $M$ presented samples. To compute the changes of the free-parameters of the network, the process is applied over all training samples. A full iteration over all the training samples is called “epoch”. Repeating the same process for multiple epochs the loss function is minimized.

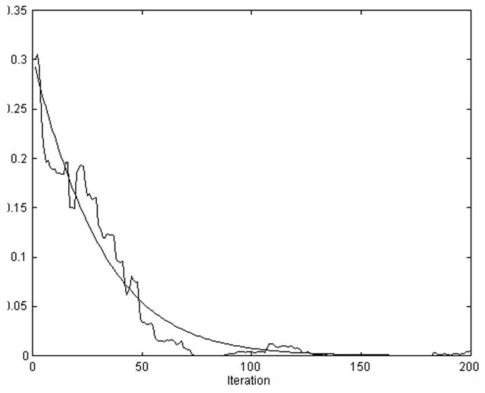


**Figure 2.11** Visual representation of SGD (noisy line) and Gradient Descent (smooth line) over a period of 200 epochs.

SGD is characterized by the noisy convergence compared to conventional gradient descent approach **(Figure 2.11)**. This can wrongly lead to the assumption of model not performing well based on chosen hyper-parameters, especially on the first epochs. For this reason, it is preferably to wait for at least few epochs until conclusions about the performance of the model are drawn. A special case of SGD called “on-line” learning can be achieved by choosing a batch size of 1. This can be used when the training data are not available at the beginning of training, or if they are made available to the model from a stream of data. Although powerful, on-line learning does not always converge and training might end up getting “stuck” on a local minima, greatly affecting the generalization performance of the model.

### 2.2.1.4 The Mini-Batch Gradient Descent

A similar algorithm to SGD, is the mini-batch gradient descent algorithm (MBGD). The algorithm relies too on splitting the training dataset into multiple batches, but instead of updating the free-parameters of the network once every epoch, it does on every batch. This reduces a lot the noisy training behavior of conventional SGD and thus, keeps the best of both worlds.

The advancements in computer hardware and most importantly the Graphical Processing Units (GPU) enables MBGD algorithm to take advantage of performing matrix multiplications right on the GPU memory without having to wait for each mini-batch to get back from RAM which involves CPU wait. As a result, this allows faster gradient computation, significantly increasing the training performance. The batch size is an important hyper-parameter of the algorithm as it has a direct correlation with the training speed. The size of the batch is determined mainly based on the available resources (memory) of the computational unit.

Although MBGD can significantly increase the training speed of a network, the algorithm does not guarantee good convergence of the network, as it can easily fell for a local minima. The learning rate with which the algorithm will update the free-parameters is really important. A low learning rate can significantly increase the training time, effectively eliminating the benefits of the algorithm, while using a large one can create unstable learning. To deal with the problem of learning rate selection, ad-hoc solutions have been developed including learning rate scheduling (Robbins & Monro 1951; Moreira & Fiesler 1995), which adjust the learning rate based on the fluctuation of the cost function. One down side of these approaches is that they have to be defined before training and are not able to adapt with all the idiomorphic characteristics of a dataset (Darken et al. 1992).

### 2.2.1.5 Newton’s Optimization Algorithm

Newton’s optimization (Kelley 1999) is an approach that uses second order derivates and it can lead, in theory, to quicker converge times compared training with standard gradient descent. According to this approach a loss function $C(w)$, where $w = \{w_1, w_2 \dots w_n\}$, can be approximated using the Taylor’s theorem:

$$C(w + \varDelta w) = C(w) + \sum_j \frac{\vartheta C}{\vartheta X w_j} \varDelta w + \frac{1}{2} \sum_{jk} \varDelta w_j \frac{\vartheta^2 C}{\vartheta w_j \vartheta w_k} + \cdots \tag{2.16}$$

This can be rewritten with respect to the gradient vector $\nabla C$ as:

$$C(w+\Delta w)=C(w)+\nabla C\cdot\Delta w+\frac{1}{2}\Delta w^T H\Delta w+\cdots \tag{2.17}$$

where $H$ is the Hessian matrix, with $j,k_{th}$ entries set as $\frac{\vartheta^2 C}{\vartheta w_j \vartheta w_k}$. The $C$ can be estimated by calculating:

$$C(w+\Delta w)\cong C(w)+\nabla C\cdot\Delta w+\frac{1}{2}\Delta w^T H\Delta w \tag{2.18}$$

Using calculus Eq. (2.17) can be minimized by Eq. (2.18):

$$\Delta w=-H^{-1}\nabla C \tag{2.19}$$

and thus, an estimation that decreases $C$ can be obtained as follows:

$$w\rightarrow w'=w-H^{-1}\nabla C \tag{2.20}$$

Similarly, this can be extended for biases and any other free-parameter of the network. This methodology can be expressed as an iterative algorithm with the following three steps:

1. Randomly initialize the weights $w$ and any other free-parameter of the network
2. Calculate the first order derivative of $w$ by using Eq. (2.19)
3. Update the calculated weights $w'$ using the second order derivative of $w''$ such as $w''=w'-H'^{-1}\nabla' C$

Learning rate can also be introduced to the Eq. (2.19), which can control the learning speed of training:

$$\Delta w=-nH^{-1}\nabla C \tag{2.21}$$

While Newton’s approximation converges faster than conventional gradient descent, it suffers from increased complexity due to the second order derivatives used in the optimization process. As a result, in a network with millions free-parameters this method can be considered not feasible even with the today’s modern GPUs.

#### 2.2.1.6 Introducing Momentum in Stochastic Gradient Descent

Local minima create relatively steep curves when optimizing a loss function using the conventional SGD algorithm (Sutton 1986), leading to hesitant movement towards the local minima slope **(Figure 2.12a)**.

A parameter that controls the velocity of the minimization rate, called "momentum" (Qian 1999) can be introduced to the conventional SGD algorithm to deal with this problem. Effectively momentum can be compared with the Newton's optimization, Hessian matrix-based technique, yet without the performance implications of computing second order derivatives. Introducing the momentum $m$ parameter to the original SGD algorithm is relatively straight forward. Initially let $v$ be the velocity in which the weights are changed:

$$v \rightarrow v' = mv - n\nabla C \tag{2.22}$$

where $m$ is a constant parameter (momentum), and $n$ the learning rate. Using momentum, the update rule of SGD can be rewritten as:

$$w \rightarrow w' = w + v' \tag{2.23}$$

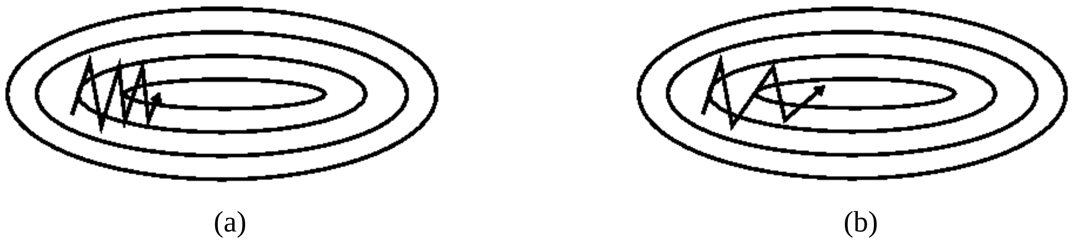


**Figure 2.12** A visual representation optimizing a loss function using the (a) SGD and (b) SGD with Momentum algorithms.

Typically, the momentum constant is a real number, ranging between 0 and 1 and it controls the accumulation of speed of change towards the direction of the global minimum. Having m close to 1 can lead to velocity that builds up on every iteration of the algorithm which in return increases the training speed, yet it creates fluctuations. Having relatively low momentum, or even 0, leads to very small, or any, velocity build up, leading to behavior similar to the original gradient descent.

### 2.2.1.7 Nesterov Accelerated Gradient

Although momentum approach is capable of controlling the acceleration of training, it might accelerate towards a local minimum. Aiming to solve this problem Nesterov Accelerated Gradient (NAG) (Nesterov 1983) was introduced. NAG attempts to approximate the future direction of the gradient, instead of blindly searching for the global minima, which results in forcing the gradient direction to move towards the correct direction.

$$v \rightarrow v' = mv - n\nabla C(w - mv) \tag{2.24}$$

In the illustration of **(Figure 2.13)** it can be noticed that momentum gradient descent initially with a small step towards the minima (blue line) accumulating velocity which creates a large "jump" away from the global minima (long blue line). On the other hand, NAG, represented in green, makes a big jump towards the direction of the previous gradient (brown line) and by measuring the gradient in which it ends up, makes small corrections (small red lines)

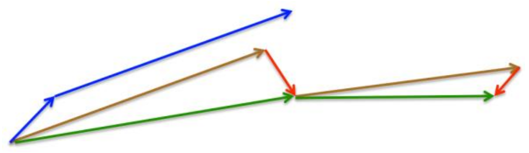


**Figure 2.13** An illustration of momentum (blue line) gradient descent compared with the NAG optimization (green line) approach. The brown and red lines represent the jumps and the corrections made by the NAG optimization approach.

### 2.2.1.8 Adagrad optimization

Adagrad is an adaptive optimization algorithm proposed by (Duchi et al. 2011) which adapts the learning rate of the gradient descent based on the free-parameters state of the network. The algorithm does large updates on the infrequently used free-parameters and smaller ones on the ones used a lot, primarily suited for sparce data. As in Adagrad case the learning rate is varying on every free-parameter update, the classic SGD algorithm can be written as:

$$w_t \rightarrow w_{t+1} = w_t - n\nabla C(w_t) \tag{2.25}$$

where $w_t$ are the free-parameters updated on every $t$ time step and $\nabla C(w_t)$ the gradient. Adagrad learning rate is adjusted on every next step such as:

$$w_t \rightarrow w_{t+1} = w_t - \frac{n}{D_t}\nabla C(w_t) \tag{2.26}$$

where $D_t = \sqrt{G_t + \varepsilon}$. The $\varepsilon$ is a constant parameter, called “smoothing term”, usually set to $1e - 8$ and $G_t$ a diagonal matrix with values the sum of squires of the gradients of all the previous time steps. An element-wise matrix ($\odot$) multiplication can be performed between $G_t$ and the $\nabla C(w_t)$:

$$w_t \rightarrow w_{t+1} = w_t - \frac{n}{D_t}\odot\nabla C(w_t) \tag{2.27}$$

The benefits of Adagrad compared to conventional SGD have been seen in several cases, such when Google (Dean et al. 2012) used Adagrad to train a neural network on finding cats in YouTube videos. Furthermore, Adagrad was used by (Pennington et al. 2014) to successfully train a neural network on the “Glove Word Embeddings”, in which naturally frequent words require much smaller updates than the infrequent ones.

The main benefit of Adagrad algorithm is that the learning rate $n$, does not require pre-training selection and thus reducing the number of hyper-parameters selection. A small learning rate is typically selected in the initialization phase, such as $\mathrm{n} = 0.001$. Unfortunately, Adagrad suffers from exploding sums, as the accumulative terms are always positive which leads to gradually vanishing learning rate, which in some cases can halt completely the training process.

### 2.2.1.9 Adadelta optimization

The problem of the vanishing learning rate of Adagrad is solved by the Adadelta optimization approach (Zeiler 2012). The algorithm solves the problem of exploding sums by using fixed size $w$ of accumulated gradients. The accumulated gradients are stored efficiently by recursively defining them as a decaying average of all previously computed squared gradients. The decaying gradient can be defined as:

$$E[\nabla C(w)^2]_t = \gamma E[\nabla C(w)^2]_{t-1} + (1-\gamma)\nabla C(w)_t^2 \tag{2.28}$$

where $E[\nabla C(w)^2]_t$ is the running average at the timestep $t$ and $\gamma$ the momentum constant which is typically 0.9. According to this Eq. (2.29) the SGD can be re-written as:

$$\varDelta w_t = \frac{-n}{D_t} \odot \nabla C(w_t) \tag{2.29}$$

$$w_{t+1} = w_t + \varDelta w_t \tag{2.30}$$

where $\odot$ represents element-wise matrix multiplication. The $D_t = \sqrt{G_t + \varepsilon}$ which is the same as Adagrad, while in Adadelta case, $G_t$ is a vector of decaying average. Rewriting Eq. (2.30) we get:

$$\varDelta w_t = \frac{-n}{\sqrt{E[\nabla C(w)^2]_t + \varepsilon}} \odot \nabla C(w_t) \tag{2.31}$$

Replacing the $E[\nabla C(w)^2]_t$ parameter with the root mean square error criteria (RMS) of the gradient, it can be noticed that the need for an initial learning rate is eliminated from the equation:

$$\varDelta w_t = \frac{-RMS[\varDelta w]_{t-1}}{RMS[\nabla C(w)]_t} \nabla C(w_t) \tag{2.32}$$

This new update rule relies on $E[\varDelta w^2]_t = \gamma E[\nabla C(w)^2]_{t-1} + (1-\gamma)\varDelta w_t^2$. As a result, the RMS parameter updates are:

$$RMS[\varDelta w]_t = \sqrt{E[\varDelta w^2]_t + \varepsilon} \tag{2.33}$$

in which the $RMS[\Delta w]_t$ value is unknown and an approximation of it can be achieved by calculating the RMS until $RMS[\Delta w]_{t-1}$.

### 2.2.1.10 RMSProp optimization

The RMSProp (Hinton et al. 2012) optimization algorithm, is a widely used optimizer based on Adadelta. The algorithm, as Adadelta, it too tries to solve the problem of vanishing learning rate. The equations behind RMSProp, closely resemble the Adadelta optimizer:

$$E[\nabla C(w)^2]_t = 0.9E[\nabla C(w)^2]_{t-1} + 0.1\nabla C(w)_t^2 \tag{2.34}$$

$$w_{t+1} = w_t - \frac{n}{\sqrt{E[\nabla C(w)]_t^2 + \varepsilon}} \nabla C(w)_t^2 \tag{2.35}$$

where $n$ is the learning rate, and $\gamma$ the momentum parameter, with initial values 0.001 and 0.9, respectively. The difference between the two algorithms, is that in the case of RMSProp, exponentially decaying average of the squared gradients are used to divide the learning rate. The main drawback of RMSProp compared to Adadelta, is that the learning rate is a hyper parameter that needs to be selected before training.

### 2.2.1.11 Adam optimization

Similarly to RMSProp and Adadelta, another adaptive learning rate method to compute gradient descent is called Adaptive Moment Estimation (Adam) (Kingma & Ba 2014). At the time of writing, the Adam optimizer is one of the most used optimizers in the field of Deep Learning. The algorithm keeps the exponentially decaying gradient average factor $v_t$ and extends it by incorporating an exponentially decaying average of the previous gradients $m_t$, which resembles a momentum parameter. These two vectors are estimates of the first moment (mean) and second moment (uncentered variance) of gradients, respectively:

$$m_t = p_1 m_{t-1} + (1 - p_1)\nabla C(w)_t \tag{2.36}$$

$$v_t = p_2 v_{t-1} + (1 - p_2)\nabla C(w)_t^2 \tag{2.37}$$

The $p_1$ and $p_2$ are constant parameters, representing the decay rates with values, according to (Kingma & Ba 2014), 0.9 and 0.999, respectively. The update rule of Adam optimizer can be expressed as:

$$w_{t+1} = w_t - \frac{n}{\sqrt{v_t} + 10^{-8}} m_t \tag{2.38}$$

### 2.2.1.12 AdaMax optimization

Along with the Adam optimizer, (Kingma & Ba 2014) proposed another optimizer named AdaMax. The authors noticed that the update rule of Adam, is inversely proportional to the $l_2$ norm of the previous gradients as the rule contains the $v_{t-1}$ term and current gradient $|\nabla C(w)_t|^2$. AdaMax takes advantage of this by extending the $l_2$ norm to $l_\infty$:

$$v_t = p_1^n v_{t-1} + (1 - p_1^n)|\nabla C(w)_t|^n \tag{2.39}$$

It can be noticed that having relatively high value for the $n$ parameter can result into numerically unstable problems. For this reason, a typical value for the parameters is either $l_1$ or $l_2$. An interesting case, where the $n = \infty$ also results into stable learning:

$$u_t = p_2^\infty v_{t-1} + (1 - p_2^\infty)|\nabla C(w)_t|^\infty = \max(p_2 v_{t-1}, |\nabla C(w)_t|) \tag{2.40}$$

The $v_t$ vector is denoted as $u_t$ to avoid mixing of the two equations. Using $u_t$ in Adam update rule (2.39) we get:

$$w_{t+1} = w_t - \frac{n}{u_t} m_t \tag{2.41}$$

Similar to Adam, the authors suggest $p_1, p_2$ constant decay rates as 0.9 and 0.999 respectively and $n = 0.002$.

### 2.2.1.13 Nadam optimization

Nesterov-accelerated Adaptive Moment Estimation (Nadam) (Dozat 2016) optimization algorithm is an incorporation of Adam and Nesterov Acceleration Gradient (NAG) algorithms. The algorithm modifies the Adam's $m_t$ vector with a momentum like parameter ($\gamma$):

$$m_t = \gamma m_{t-1} + n\nabla C(w_t - \gamma m_{t-1}) \tag{2.42}$$

$$w_{t+1} = w_t - m_t \tag{2.43}$$

Nadam alters the original NAG algorithm and instead of double computation of the momentum step, the look-ahead momentum is performed directly on the update rule such that the Eq. (2.42) and Eq. (2.43) become:

$$m_t = \gamma m_{t-1} + n\nabla C(w_t) \tag{2.44}$$

$$w_{t+1} = w_t - \left(\gamma m_t + n\nabla C(w_t)\right) \tag{2.45}$$

It can be noticed that instead of using the previous $m_{t-1}$ for the update, the algorithm uses only the current step $m_t$ momentum vector. Similarly to the NAG incorporation, Nesterov momentum in Adam optimizer is incorporated as:

$$m_t = p_1 m_{t-1} + (1 - p_1)\nabla C(w)_t \tag{2.46}$$

$$w_{t+1} = w_t - \frac{n}{\sqrt{v_t} + \varepsilon}\left(\frac{p_1 m_{t-1} + (1 - p_1)\nabla C(w)_t}{1 - p_1^t}\right) \tag{2.47}$$

The $p_1$ parameter represents the decay rate and $\frac{p_1 m_{t-1}}{1-p_1^t}$ is a bias corrected estimation of the momentum vector ($m_{t-1}$) of the previous step, expressed as an estimation of current momentum vector $m_t$. The final update rule of Nadam can be expressed as:

$$w_{t+1} = w_t - \frac{n}{\sqrt{v_t} + \varepsilon}\left(p_1 m_t \frac{(1 - p_1)\nabla C(w)_t}{1 - p_1^t}\right) \tag{2.48}$$

## 2.2.2 The Error Backpropagation

Although in the previous section, a large variety of optimization algorithms have been examined, the incorporation of them in neural network training has not been presented. One of the most widely used approaches in training neural networks is the error backpropagation. The algorithm has been originally proposed in mid 70s, year it did not receive enough attention until it was used in neural network training (Rumelhart et al. 1986). The authors showed that using back propagation method, the training of neural networks was significantly faster compared to older training approaches. To understand error backpropagation, a simple MLP is illustrated in **(Figure 2.14)**, where $w_{jk}^l$ is the weight of the $k^{th}$ neuron in the $(l-1)^{th}$ layer to the $j^{th}$ neuron on $l^{th}$ layer. Similar to the weights, let $b_j^l$ be the bias and $a_j^l$ the activation function, of the $j^{th}$ neuron in the $l^{th}$ layer, respectively. The activation function of a neuron can now be expressed as sums of all the neurons on the $k^{th}$neurons the (previous) layer $l-1$:

$$a_j^l = \sigma\left(\sum_k w_{jk}^l \, a_j^{l-1} + b_j^l\right) \tag{2.49}$$

The computational efficiency of Eq. (2.49) can be improved by using matrix multiplications, and thus enable modern GPU acceleration techniques to be used:

$$a^l = \sigma\left(w^l a^{l-1} + b^l\right) \tag{2.50}$$

where $a^l$ notates a matrix of the activation function outputs of the $l^{th}$ layer, $w^l$ and $b^l$ matrices with all the weights and biases of the neurons in $l^{th}$ layer and $a^{l-1}$ a matrix with the activation function output of each neuron of the previous layer. The intermediate $l-1$ activation function matrices are computed, when Eq. (2.50) is applied on a forward pass of the network to get the output of the last layer. These matrices, which are denoted as $z^l$, are kept as they are used to reduce the computational complexity of the next steps of backpropagation algorithm.

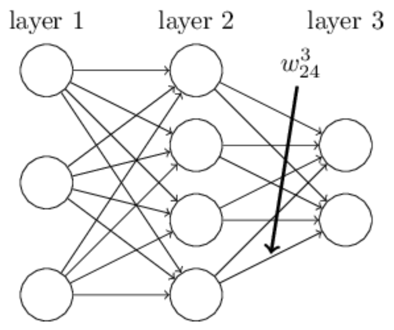


**Figure 2.14** An illustration of a simple MLP with one hidden layer of neurons. The $w^3_{24}$ detonating connection from 4th node of the second layer to the 3rd layer.

### 2.2.2.1 Loss Function Characteristics

The target of backpropagation algorithms is to compute the partial derivatives $\frac{\vartheta C}{\vartheta w}, \frac{\vartheta C}{\vartheta b}$ of a loss function $C$ with respect of all the weights, the biases and in general the free-parameters of the network. To do that, backpropagation mandates the loss function to meet two requirements. The first, is that loss function $C$ shall be expressible as an average sum of all $C_t$ of individual all training examples and thus $C = \frac{1}{n}\sum_x C_x$. This ensures that the algorithm can compute the $\frac{\vartheta C_t}{\vartheta w}, \frac{\vartheta C_t}{\vartheta b}$ derivatives and later or average them in order to compute the actual $\frac{\vartheta C}{\vartheta w}, \frac{\vartheta C}{\vartheta b}$. The second requirement mandates the loss function to be expressed as a function of the output of each neuron of the last layer, such that $C = C(a^l)$. To demonstrate a function which relies on the output of all the previous activations we can think the quadratic loss function. In this case, for a single training example $t$, the loss can be expressed as:

$$C = \frac{1}{2}(|y - a^L|)^2 = \frac{1}{2}\sum_j \left(y_j - a^L_j\right)^2 \tag{2.51}$$

where $y$ is the desired output (target) of the training example $t$.

### 2.2.2.2 The Fundamental Equations of the Backpropagation Algorithm

Ultimately, the backpropagation algorithm computes the $\frac{\vartheta C}{\vartheta w_{jk}^{l}}, \frac{\vartheta C}{\vartheta b_{j}^{l}}$ partial derivatives using the error of a loss function $C$. To do that, an intermediate quantity is used, that corresponds to the "error of the neuron $j^{th}$ neuron in the $l^{th}$ layer" notated as $\delta_{j}^{l}$. Using the backpropagation algorithm, the aim is to compute the $\delta_{j}^{l}$ matrices and later associate them with $\frac{\vartheta C}{\vartheta w_{jk}^{l}}$ and $\frac{\vartheta C}{\vartheta b_{j}^{l}}$ partial derivatives. In effect, the error $\delta_{j}^{l}$ is considered as a small noise introduced when the input passes through the neurons of the network in the feed-forward pass. Let $\Delta z_{j}^{l}$ be the noise to the input weights of the neuron, which in return produce $\sigma\left(z_{j}^{l} + \Delta z_{j}^{l}\right)$ instead of $\sigma\left(z_{j}^{l}\right)$. This small error propagates through the neurons of the network, affecting the overall performance. This can be expressed as $\frac{\vartheta C}{\vartheta z_{j}^{l}} \Delta z_{j}^{l}$. The aim of the algorithm is to compute the $\Delta z_{j}^{l}$ values that reduces the overall loss function $C$ output. When the $\frac{\vartheta C}{\vartheta z_{j}^{l}}$ has a large value, we can lower down the loss by choosing a $\Delta z_{j}^{l}$ with opposite sign, while when the $\frac{\vartheta C}{\vartheta z_{j}^{l}}$ is close to zero, the $\Delta z_{j}^{l}$ has also to be close to zero. Having a small $\frac{\vartheta C}{\vartheta z_{j}^{l}}$ means that the neurons are already optimized. The quantity $\delta_{j}^{l}$ can be defined as Eq. (2.52) where $\delta^{l}$ is the error of the $l^{th}$ layer in a vectorized form.

$$\delta_{j}^{l} \equiv \frac{\vartheta C}{\vartheta z_{j}^{l}}, \delta^{l} \equiv \frac{\vartheta C}{\vartheta z^{l}} \tag{2.52}$$

In the output layer of a neural network the components $\delta^{l}$ can be computed as:

$$\delta_{j}^{L} \equiv \frac{\vartheta C}{\vartheta a_{j}^{L}} \sigma'^{\left(z_{j}^{L}\right)} \tag{2.53}$$

The rate of change of the loss function in respect to activation of the $j^{th}$neuron of the last layer is expressed in the first part of the equation, *i.e.* $\frac{\vartheta C}{\vartheta a_{j}^{L}}$, while the last part $\sigma'^{\left(z_{j}^{L}\right)}$, measures the rate of change of the activation function $\sigma$ at $z_{j}^{L}$. The Eq. (2.53) is relatively computationally inexpensive to calculate. The only computational overhead is to the $\sigma'^{\left(z_{j}^{L}\right)}$. For the quadratic loss function, the computation is as simple as:

$$\frac{\vartheta C}{\vartheta a_{j}^{L}} = a_{j}^{L} - y_{j} \tag{2.54}$$

As the Eq. (2.53) is expressing the rate of change of each neuron, it can be rewritten using matrices, speed up the computations:

$$\delta^L = \nabla_a C \sigma'^{(z^L)} \tag{2.55}$$

where $\nabla_a C$ is a matrix whom components are the partial derivatives $\frac{\vartheta C}{\vartheta a_j^L}$

The second equation on which backpropagation relies on, calculates the error $\delta^l$ in respect to the errors of the next layer:

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'^{(z^l)} \tag{2.56}$$

where $(w^{l+1})^T$ is the transposed weight matrix of the $(l+1)^{th}$ layer. Notice that the transposed weight matrix is multiplied by the error $\delta^{(l+1)}$. This expresses that the error is passed backward through the network. The Hadamard product $\odot$, on which the second component of the equation relies on, is similar to the first component but in this case, it uses the backward error propagation through the activation functions of the previous layer $l^{th}$layer. Using Eq. (2.53) and Eq. (2.56) the error $\delta^l$ can be calculated for any layer of the network.

The rate of change of the loss function with respect to the biases of the network, is controlled by the third equation of the backpropagation algorithm:

$$\frac{\vartheta C}{\vartheta b_j^l} = \delta_j^l \tag{2.57}$$

The vectorized equivalent of this equation can be expressed as $\frac{\vartheta C}{\vartheta b} = \delta$. Using these equations, it is possible to compute the $\delta^l$ error as the quantity $\frac{\vartheta C}{\vartheta b}$ is already known by the previous steps.

The fourth and last equation of the algorithm computes the rate of change of the loss function with respect of any weight in neural network from which, it is already known how to compute the error $\delta^l$and the $a^{l-1}$:

$$\frac{\vartheta C}{\vartheta w} = a_{\text{in}} \delta_{\text{out}} \tag{2.58}$$

The $a_{\text{in}}$ parameter of the equation is the activation of the neural input to the weight $w$ and the $\delta_{\text{out}}$ is the error of the output with respect to the weights $w$.

An important characteristic of the backpropagation algorithm is that based on Eq. (2.51) if $\sigma(z^l) \approx 0$ or $\sigma(z^l) \approx 1$ then the $\sigma'^{(z^l)} \approx 0$. This is because the output of the sigmoid function is nearly constant on regions close to zero or one. This has a direct consequence to the weights of the final layer which will learn slower when the output value is relatively high or low. This is also known as "saturated neuron" as the neurons stop learning. This extends to all free parameters of the network, included the biases. Following the second equation of the backpropagation algorithm, this is extended to all the neurons of the network. To battle this problem, other activation functions such as the ReLU activation function are commonly used, especially in deep neural networks.

The error backpropagation algorithm can be expressed as five iterative steps:

1) Input: training examples
   Set the activations $a^1$ for the input layer.

2) Feed-Forward
   For each layer $l$ in $\{2,3,\ldots,L\}$ compute the $z^l = w^l a^{(l-1)} + b^l$ and $a^l = \sigma(z^l)$.

3) Output error $\delta^L$
   Calculate the matrix $\delta^L = \nabla_a C \odot \sigma'^{(z^L)}$.

4) Backpropagate the error
   For each layer $l$ in $\{L-1, L-2, \ldots, 2\}$ compute $\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'^{(z^l)}$.

5) Output
   Compute the $\frac{\vartheta C}{\vartheta w} = a_{\text{in}} \delta_{\text{out}}$ and $\frac{\vartheta C}{\vartheta b_j^l} = \delta_j^l$ which are the gradient of the cost function $C$.

The algorithm name comes from the fact that the error is backpropagated through the network. The reason that the algorithm starts with the initial feed-forward phase, is that the overall error of the network is a result of previous errors accumulated by the neurons of each layer. By performing the feed-forward pass it is becoming possible to use the chain rule, from mathematical calculus, working backwards throughout the previous layers to obtain the final expression.

To use mini-batch gradient descent, the algorithm is slightly modified:

1) Input: training examples

2) For each of the training examples $t$
   Set the activations $a^1$ for the input layer.

   1) Feed-Forward
      For each $l$ in $\{2,3,\ldots,L\}$ compute $z^{t,l} = w^l a^{(t,l-1)} + b^l$ and $a^{t,l} = \sigma(z^{t,l})$

2) Output error $\delta^{t,L}$
   Calculate the matrix $\delta^{t,L} = \nabla_a C \odot \sigma'^{(z^L)}$

3) Backpropagate the error
   For each layer $l$ in $\{L-1, L-2, \dots, 2\}$ compute $\delta^{t,l} = \left(\left(w^{l+1}\right)^T \delta^{t,l+1}\right) \odot \sigma'^{(z^{t,l})}$.

3) Gradient descent
   For each layer $l$ in $\{L-1, L-2, \dots, 2\}$ update the weights according to the rule $w^l \to w^l - n/m \sum_t \delta^{t,l} \, \alpha^{t,l-1}$ and the biases $b^l \to b^l - n/m \sum_t \delta^{t,l}$

The same algorithm can also be used in the case of SGD, in which the only modification needed is the 2nd step, in which an outer loop needs to be added that selects the subset of the training examples training in multiple epochs.

## 2.3 Tuning Neural Networks

Training a neural network that generalizes well on unknown data is a challenging task (Goodfellow et al. 2016). When training a network, the goal is to generalize well from the training data to any unknown data from the problem domain. The number of free-parameters of a model define its learning capacity and thus affect its the generalization performance. A model with small number of free-parameters might not be able to learn and thus generalize (underfitting), whereas a model with too many free-parameters, is prone to overfitting. Overfitting describes a situation where the trained model becomes really good on predicting on data that has already been trained with, yet behaves poorly when tested on unknown data. Having an “ideal” learning capacity in a network, does not guarantee good generalization performance. This is because the free-parameters of the network are optimized according to the data available upon training and thus, having enough and diverse training data is a requirement for good generalization performance. Finding the balance between the learning capacity and the number of training samples required for training is a challenging task in neural network tuning. The rest of this section describes techniques to cope with neural network tuning.

### 2.3.1 The Early-Stopping Technique

In the process of training an ANN, a training dataset is used to evaluate the performance of the model by computing the output of the loss function. This process prone to overfitting as the model free-parameters are calibrated based on that training data. Furthermore, as the training data are used to evaluate the performance of the network, the hyper parameters of the model are biased towards the training dataset. A more accurate approach of evaluating the performance of the network, while training, is to use a validation dataset. More specifically, this dataset is a subset of the training dataset, that is excluded from training, and is used solely for the model generalization performance evaluation, typically after every epoch.

Using the validation dataset, it is possible to evaluate the performance of the network while training independently of the training dataset. Monitoring the generalization performance of the network on the validation dataset in comparison with the training data generalization performance, it is possible to detect and prevent overfitting. This process is also called early stopping (Girosi et al. 1995), which as the name suggest, when the network begins to generalize better on the training data compared to unknown, validation data, the training process should be halted. This is visually illustrated in **(Figure 2.15)**. Although in theory this works well, when SGD or other "noisy" optimization algorithms are used, the stopping point can become hard to identify. To deal with this problem, the model is allowed to be training for a fixed number of epochs, beyond the early stopping point, while keeping track of the generalization performance degradation. If the degradation continues, early stopping is applied, otherwise the model continues to train until the next early stopping point is found. Nowadays, and because of the increase storage capacity of modern computers, it is common to keep a history of every epoch in a form of model snapshot, in order to find the optimal accuracy between validation and training dataset.

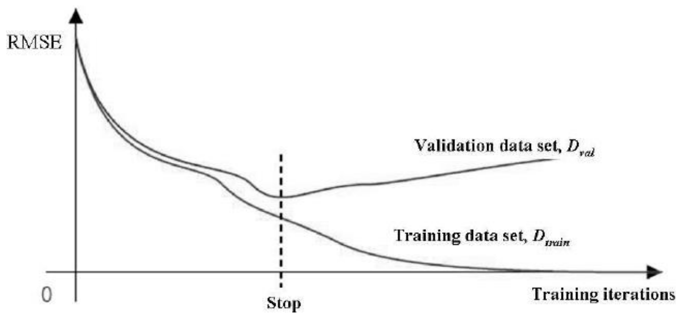


**Figure 2.15** Validation and training dataset early stopping point.

## 2.3.2 Weight Initialization

Weight and biases initialization is an important decision when training any kind of ANN. Initializing with random weights might degrade the performance of the entire training process or even prevent it entirely. The later usually occurs when weights or biases have zero value and thus the neurons are already saturated before the training process even begins. In practice a common approach is to use random independent values that follow the Gaussian distribution, normalized to have 0 mean and 1 standard deviation. Let $z$ be the sum of all the weights and biases of a single neuron and thus:

$$z = \sum_{i=1}^{n} w_i x_i + b \tag{2.59}$$

That means that $z$ is a sum of random Gaussian variables an extra bias term. Assuming that half of $x_i$ will be turned off, or in other words they will be set to 0, and the rest are activated, if the number of weights connected to a single neuron is large, for example 500 weights, $z$ will have a value of 501 considering 500 weights and 1 bias. The standard deviation of $z$ will be $\sqrt{501} \cong 22.4$ with zero mean **(Figure 2.16)** which shows that the $abs(z)$ will be a large value and thus $z \gg 1$ $or$ $z \ll -1$. In case of a sigmoid neuron, this will lead to a pre-saturated neuron as $\sigma(z)$ will be either strongly 0 or 1. As backpropagation works by applying small weights in order to explore the surface of the cost function of the network, the changes will have little to no effect to the neuron which leads to slow learning and thus damages the whole training process.

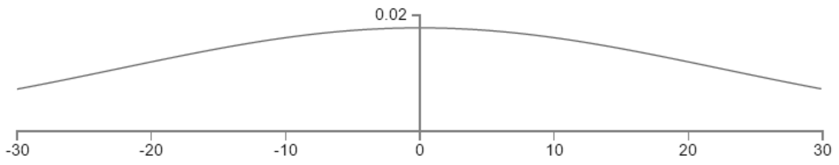


**Figure 2.16** Gaussian distribution of the values of a 500 weight and 1 bias neuron.

To prevent this, an alternative approach is use Gaussian normalized free parameters with mean zero, yet change the standard deviation to be around $1/\sqrt{n}$, where $n$ the number of free-parameters connected to the neuron. This causes the Gaussian distribution to be squashed down and thus have much less possibility for the neuron to be saturated upon initialization. By following that in the previous example, the standard deviation of $z$ will be $\sqrt{\frac{2}{3}} = 1.22$ with a much sharper Gaussian distribution **(Figure 2.17)**.

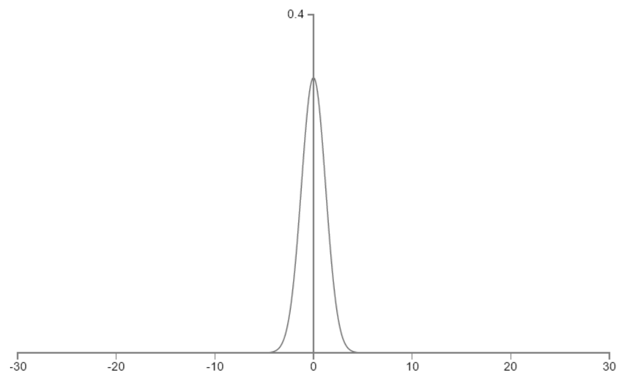


**Figure 2.17** Gaussian distribution with standard deviation 1.22 and mean zero.

## 2.3.3 The L2 Regularization

Weight decay is a regularization technique, also known as L2 regularization. To incorporate weight decay, a modification of the loss function is required in which a regularization term is added. Examining the original cross-entropy loss function, the regularization term can be added as:

$$\mathrm{C} = -\frac{1}{n}\sum_t (yln(a) + (1-y)\ln(1-a)) + \frac{\lambda}{2n}\sum_w w^2 \qquad (2.60)$$

in which the first term is the original cross-entropy loss function, followed by the sum of the squares of all the weights. The scaling factor $\boldsymbol{\frac{\lambda}{2n}}$ , where $\boldsymbol{\lambda}$>0 and $\boldsymbol{n}$ the number of training examples, is added in order to regularize the parameters. Similarly for the quadratic loss function, weight decay can be incorporated which as:

$$C = \frac{1}{2}(|y - a^L|)^2 = \frac{1}{2}\sum_j \left(y_j - a_j^L\right)^2 + \frac{\lambda}{2n}\sum_w w^2 \tag{2.61}$$

In fact, the L2 regularization term, can be added to any loss function by following:

$$C = C_0 + \frac{\lambda}{2n}\sum_w w^2 \tag{2.62}$$

where $\boldsymbol{C_0}$ is the original lost function. The regularization parameter addition guides the network to learn small weights while larger weights are allowed only if they result in a considerable change of the original loss function output. The term that regulates the weight scale balance is $\boldsymbol{\lambda}$ where a small value swifts the attention to minimize the original cost function $\boldsymbol{C_0}$ while a larger value swifts the preference to the small weights.

To incorporate the regularization factor $\lambda$ in SGD algorithm, the partial derivatives of gradient descent are expressed as:

$$\frac{\vartheta C}{\vartheta w} = \frac{\vartheta C_0}{\vartheta w} + \frac{\lambda}{n}w \tag{2.63}$$

$$\frac{\vartheta C}{\vartheta b} = \frac{\vartheta C_0}{\vartheta b} \tag{2.64}$$

Notice that the partial derivatives of biased remain unchanged, as the regularization The is applied only on the weights of the network. In that sense the update rule of the backpropagation algorithm can be re-written as:

$$w \rightarrow w' = w - \left(1 - \frac{n\lambda}{n}\right)w - n\frac{\vartheta C_0}{\vartheta w} \tag{2.65}$$

$$b \rightarrow b' = b - n\frac{\vartheta C_0}{\vartheta b} \tag{2.66}$$

The only to the original update rule is the rescaling of the weights, which is also called weight decay. According to these equations the update rules of the SGD algorithm ca be rewritten as:

$$w \rightarrow w' = w - \left(1 - \frac{n\lambda}{n}\right)w - \frac{n}{m}\sum_t \frac{\vartheta C_t}{\vartheta w} \tag{2.67}$$

$$b \rightarrow b' = b - \frac{n}{m}\sum_t \frac{\vartheta C_t}{\vartheta b} \tag{2.68}$$

The term $C_t$ is the unregularized cost of each training example of each mini-batch as computed by the original backpropagation algorithm.

## 2.3.4 The L1 Regularization

L1 regularization is similar to L2 regularization, with the only difference been on the regularization term of the loss function, which in L1 case, calculates the absolute sum of the weights of the network multiplied by the regularization factor $\lambda$:

$$C = C_0 + \frac{\lambda}{n} \sum_w abs(w) \tag{2.69}$$

The partial derivative of Eq. (2.69) with respect to the weights of the network can be written as:

$$\frac{\vartheta C}{\vartheta w} = \frac{\vartheta C_0}{\vartheta w} + \frac{\lambda}{n} sgn(w) \tag{2.70}$$

in which the $sgn(w)$ represents the sign of the weight $w$. The update rule of the regularized model is expressed as:

$$w \to w' = w\left(1 - \frac{n\lambda}{n}\right) - n\frac{\vartheta C_0}{\vartheta w} \tag{2.71}$$

$$b \to b' = b - n\frac{\vartheta C_0}{\vartheta b} \tag{2.72}$$

Both L1 and L2 regularization are affecting the weights of the network, yet the first shrinks the weights by a constant amount towards zero, while the second the weight shrinkage is proportional to the weight. The result of this, is that when the magnitude of $|w|$ is large, the L1 regularized network shrinks the weight less than L2. In contrast, when the magnitude of the weight is small L1 regularization will affect more the weight than an L2. Concluding, an L1 regularized network focuses the weights on a relatively smaller number of high importance connections, while the rest are led towards zero. Although the difference between the two regularization forms is definite, it is still unclear which performs the better and in fact, both are widely used in literature.

## 2.3.5 Hyperparameter Selection

In the context of neural networks, the parameters that have to be decided prior the training process are known as "hyper-parameters". These parameters depend on the different set of algorithms that are used while training which includes, the learning rate $\boldsymbol{n}$, the regularization parameter $\boldsymbol{\lambda}$, the batch size $\boldsymbol{m}$ in case of stochastic gradient descent like algorithms. Unfortunately, there is no definite answer to what values they should get and thus, most of the researchers are following heuristic approaches. Similarly, the same heuristic approaches are followed for the rest of hyper-parameters of the network, including the number of hidden layers that the network should have, the number of neurons of each hidden layer and even the activation function of the neurons. In most of the cases the most effective way is the trial and error, yet there are some basic principles that can be followed as short paths.

To speed up the learning process, and quickly evaluate the effectiveness of a hyper-parameter, such as the learning rate, a common practice is to keep the training dataset small thus on each epoch, the generalization performance can be monitored and if the model performance does not follow an upwards direction alter the learning rate accordingly. Depending on the gradient descent algorithm that is used, the learning rate and the regularization parameter are usually provided by the creator of the algorithm, yet if that is not the case, a common approach is to follow a learning rate reduction by a factor of 10 on each trial. As an example, a usual starting point for learning rate, can be $n = 10^{-1}$, if the accuracy is unstable, then a lower learning rate should be applied. On the other hand, if the classification accuracy grows steadily but slowly, learning rate should increase, usually again at the same rate as the reduction. To determine the number of epochs that the model should be used, it is a good practice to follow early stopping approach and thus avoid overfitting too. For the regularization parameter, a common approach is first to start the training without regularization at all and thus $\lambda = 0$, after learning rate adjustments, the regularization parameter can be increased by a steady factor of 10, following the same principles with learning rate, yet instead of starting from high values, usually the initial value is as low as $\lambda = 10^{-5}$.

Various automated techniques have been proposed to help on hyper-parameter selection for neural networks. A common approach is the “grid search” which systematically searches through grid in hyper-parameter space to find the optimal values. A Bayesian optimization approach of parameter selection has been proposed in (Snoek et al. 2012) in which a learning algorithm’s generalization performance is modeled as a sample from a Gaussian process. A review of existing algorithms along with practical ways to implement them can be found in (Bergstra & Bengio 2012) and a recent review that covers essential topics of hyper-parameter optimization is presented in (Yu & Zhu 2020).

### 2.3.6 Training Dataset Expansion

The artificial increase of the training samples is commonly employed in the field of ANN training. This is because, in order to train a model with thousands or million free-parameters using a small dataset often leads to overfitting. An example that is commonly used in the field of CV and is to apply affine transformations such as rotations, translations and rescaling of the training samples while keeping the same class. That enables the model to adjust the free parameters accordingly and thus increase the generalization capabilities of the whole network. Another technique that has been recently employed is to use artificially generated images, typically from a generative model, such as a GAN, to create visually and semantically new images from existing dataset. This can be extended to train a neural network solely on artificially generated images (Diamantis et al. 2019).

### 2.3.7 The Neuron Dropout Technique

Neuron dropout (Srivastava et al. 2014) is another technique that is commonly used against overfitting. The technique involves randomly switching off neurons while training in order to prevent "strong connections" to be formed between neurons. An example of this can be seen in **(Figure 2.18)** where temporarily a fixed percentage of neurons are disabled. By repeating this process over every epoch, the neurons are getting trained in a way that resembles using multiple neural network architectures and thus, they learn to generalize better. Furthermore, the complexity of using multiple neural network architectures and then choosing the best is minimized as with this technique, the trials are based on every epoch which would have been computed on each network individually. This technique was used successfully in (Krizhevsky et al. 2012) were it was described as a technique that reduces the complex co-adaptations of neurons.

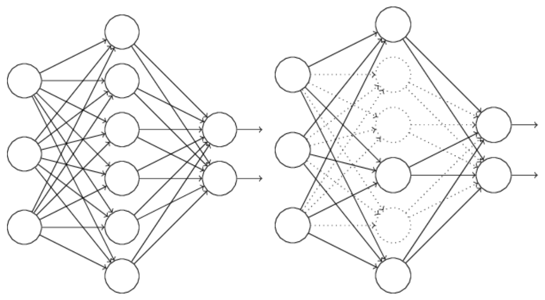


**Figure 2.18** On the left a neural network before dropout and on the right a neural network after dropout process.

## 2.4 Deep Learning

Deep learning is a subset of ML which uses ANNs with multiple cascade neural layers to progressively extract higher level features from the data. The main difference between conventional ML algorithms and deep learning is the type of data used for training the model. Conventional ML algorithms require a data pre-processing step in which typically features are extracted from the data, or selected from existing features, and used as input to the model. This pre-processing step typically requires a domain expert and aim to represent specific characteristics of the input that are of interest in the context of an application. For this reason, these features are typically referred as "hand-crafted". In CV, such features mainly include color, shape, and textural information. In deep learning models, this data pre-processing step is eliminated, as the feature extraction is automated through training, thus removing some of the dependency on domain expert. Such models ingest the entire data, instead of features from which multiple levels of features are extracted. As an example, considering that we have a dataset containing images of pets, such as cats, dogs, and parrots. In deep learning the model ingest entire images as and, through training, determines which features are more important in distinguishing each animal. In ML this hierarchy is established manually from a domain expert. Deep neural networks are usually trained using the well-known back propagation algorithm.

### 2.4.1 Convolutional Neural Networks

In the last decade and especially after the work of (Krizhevsky et al. 2012), a lot of attention has been drawn to a special type of deep neural network architecture called CNNs. CNNs are a type of multi-layer feed-forward neural network architecture that at its core, contains at least one layer of neurons, with connections that perform a special kind of operation known as convolution. Their architecture is inspired by the natural biological process of animal visual cortex, in which neurons are individually responding to small regions of the visual field. This biological arrangement was discovered by examining the visual system of cats and monkeys from the biologists Hubel and Wiesel in 50s and 60s. Later on (Hubel & Wiesel 1968) the authors identified that there are two basic types of visual cells in the brains. The first type is called "single cells" whose output is maximized by edges with particular orientation within their receptive field, which is effectively the portion of the visual image that the cell is able to view. The second type of cells referred as "complex cells" have a relatively larger receptive field compared to the first and their output is insensitive to the exact position of the edges presented into that field.

In early 80s, the work of (Hubel & Wiesel 1968), inspired an adoption of this in the field of ANNs (Fukushima & Miyake 1982) with the name "Neocognitron". The big difference between Neocognitron and the previously used architectures, was that the neurons did not require to share the same trainable parameters (weights). As a result, the architecture, instead of relying on neurons in fully connected arrangement, it was able use neurons with connection similar to ones found in the biological visual context. Due to the increased computational complexity of training such networks, the idea was effectively abandoned for almost a decade. In 1998, and mainly due to the computational power of modern computers the Neocognitron architecture revisited and improved (LeCun et al. 1998). More specifically the so called, LeNet architecture (LeCun et al. 1998), proposed a CNN architecture featuring seven layers, which was successfully trained to recognize handwritten digits in grayscale images of size 32×32 pixels. In 2003 a generalized approach was proposed (Behnke 2003), which was simplified and standardized by (Simard et al. 2003). This opened the path to the scientific community to leverage the power of these type of networks in the field of CV.

Maybe the most well-known modern CNN architecture was proposed by (Krizhevsky et al. 2012) in 2012 with the name "AlexNet". The network was trained, using modern Graphical Processing

Units (GPUs), significantly speeding up the training process. Training on GPUs was a relatively complex task, that was made possible by using the a framework provided by NVIDIA known as Compute Unified Device Architecture (CUDA) (Nickolls et al. 2008), which nowadays the de-facto backbone framework of modern neural network frameworks, such as Tensorflow (Abadi et al. 2016) and Pytorch (Paszke et al. 2019). AlexNet was trained on the ImageNet (Deng et al. 2009) dataset and competed in the "ImageNet Large Scale Visual Recognition Challenge" (ILSVRC) (Russakovsky et al. 2015), which is an annual CV competition began in 2010 and follows the principles set by PASCAL VOC challenge (Everingham et al. 2010).

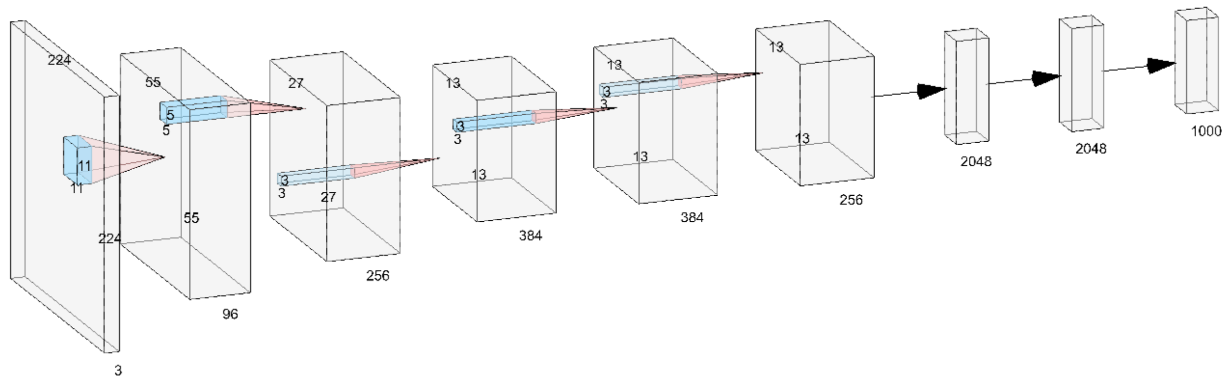


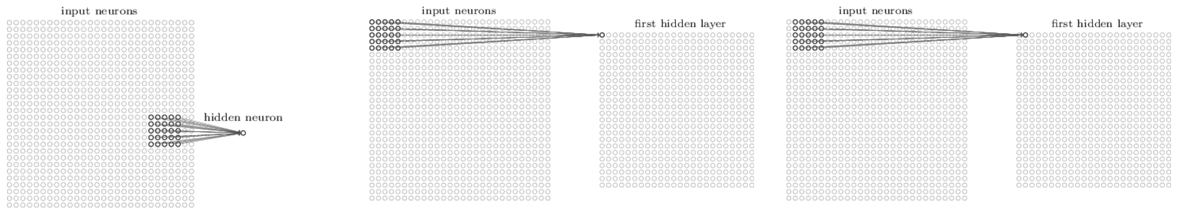
**Figure 2.19** Visual representation of the AlexNet (Krizhevsky et al. 2012) CNN architecture.

The AlexNet architecture was trained on the ILSVRC-2010 ImageNet dataset, which contains 1.3 million high resolution images of various sizes, categorized in 1000 classes. The network achieved 39.7% and 18.9% error rates on top-1 and top-5 scales[1], respectively, largely outperforming machine-learning based approaches, winning the competition, sparking the research "frenzy" in the field of CNNs. AlexNet architecture consists of five convolution layers, some of which followed by max-pooling and normalization layers. The last layers of the network are two fully connected layers followed by one output layer of 1000 neurons with softmax activations. The architecture had more than 60 million free-parameters and 500.000 neurons which was a considerably big number compared to previously proposed networks. An illustration of the architecture is included in **(Figure 2.19)**.

[1] In ImageNet classification challenge the error rates of the model are reported based on the predictions of the top 5 most likely classes. The "top-5" error rate refers to the fraction of the test images for which the correct class is amongst this top 5, while the "top-1" error rate refers to the fraction of test images for which the correct class is the one judged most likely by the model.

### 2.4.1.1 The Convolution Layer

At the core of any CNN architecture there is a special type of layer, called convolution layer, from which this family of networks takes its name. A convolution layer is similar to a classic hidden layer that is found in conventional MLP architectures. The difference between a fully connected layer and a convolution layer lies on the neuron connection arrangement. While in a typical MLP architecture all neurons of the previous layer are connected with all the neurons of the next layer, in a convolution layer, each neuron is connected with a specific set of neurons from the previous layer, which is called "receptive field" **(Figure 2.20)**. The input of a convolution layer is commonly referred as "input volume". The receptive field is defined as the region in the input volume that a particular neuron is looking. The region has a fixed size of $\boldsymbol{w} \times \boldsymbol{h}$, where $\boldsymbol{w}$ and $\boldsymbol{h}$ are the width and height, respectively. Similarly to a conventional sliding window algorithm, each neuron is connected to the next set of neurons, until the entire surface of neurons from the previous layer is covered. It is not mandatory for the regions to overlap each other. The distance, or step, by which the region shifts on (filters) the input volume is called is called stride $\boldsymbol{s}$.



(a) (b)

**Figure 2.20** (a) The receptive field of a single neuron. (b) Overlapping receptive fields.

Depending on the filter size and stride used by the convolution layer, the outermost filters might fell outside of the input volume. To overcome this problem, padding $\boldsymbol{p}$ is used, which effectively expands the input volume so that all filters can fit. An example of padding with $\boldsymbol{p} = \boldsymbol{1}$ is illustrated in **(Figure 2.21),** where the surrounding pixels of the input volume are padded with zeros.

The receptive field of each neuron in a convolution layer is extended across the depth of the input volume. To understand this, we can consider a 3D input volume, such as an RGB image, in which each channel can be considered as another slice added for the creation of a cube with a width and height equal to the width and height of the image, and depth equal to the number of channels. In such cases, the receptive field of neurons, which is also known as filter or kernel, raster scans the input volume, covering their whole width and height. A convolution layer can raster scan the input volume with many filters. The filters of the same type are sharing the same weights, increasing the computational efficiency while training. This is also known as "parameter sharing".

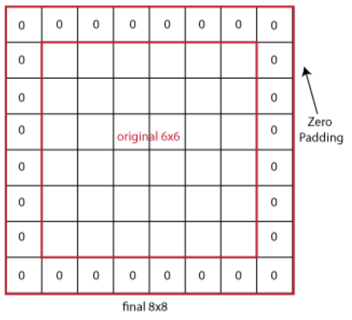


**Figure 2.21** A visual representation of padding $p = 1$ been used to surround and input volume of size 6×6 with zeros, resulting in volume of size 8×8.

Parameter sharing is the main difference of the convolution layers from the conventional fully connected layers. In a convolution layer, each neuron connection represents a receptive field, on which weights and biases need to be computed. Instead of using different parameters for each of neuron connection, the parameters are shared between them:

$$\sigma(b + \sum_{l=0}^{w}\sum_{m=0}^{h} w_{l,m}a_{j+l,k+m}) \tag{2.73}$$

where $\sigma$ is the activation function of a neuron, $b$ the shared bias and $w_{l,m}$ the array of $w \times h$ shared weights. The $a_{j+l,k+m}$ detonates the activation function output of the previous layer neuron at position $j + l, k + m$. This equation is also known as mathematical convolution and is the one from which the name of the layer derives. In literature, this equation can also be found as:

$$a^{l,m} = \sigma(b + w \cdot a^{l-1}) \tag{2.74}$$

where $a^{l,m}$ is the set of output activations of feature map $m$ in layer $l$ and $a^{l-1}$ a set of inputs from the previous layer. The "·" represents the convolution operation between the input and the shared free-parameters of the feature map.

Due to the parameter sharing properties, the same filter is computed across all the input volume, forming a feature detector, which is translation invariant. This derives from the fact that all neurons look for the same feature across the input volume and as a result high activations will be achieved wherever that feature is found. This mapping between the input and the filters of a convolution layers is named as "feature map". A convolution layer can have multiple feature maps, forming multiple layers of feature detectors. A visual representation of such feature maps is depicted in **(Figure 2.22)**, in which 20 feature maps formed by training a CNN using the MNIST dataset (Deng 2012).

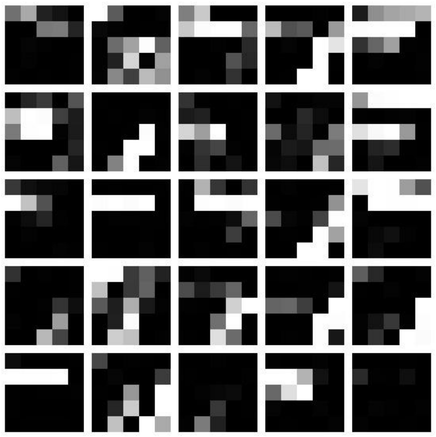


**Figure 2.22** Visualized feature maps formed by training a convolution layer using the MNIST dataset (Deng 2012).

From the visualization of the feature maps in **(Figure 2.22)** it can be noticed that the features have a spatial structure with lighter and darker regions, sensitive to corners, which resemble the conventional approaches of feature extraction such as the Gabor filters. The difference between these approaches and the feature maps is that the later, are learned based on the training samples and they do not follow a specific mathematical procedure. The parameter sharing of feature maps, is extended across the depth of the input volume. A benefit from the parameter sharing is that it reduces the number of free parameters of the convolution layer, especially when compared with conventional fully connected layers. As this extends along the whole network, the training and inference process become considerably faster, even for deeper architectures. The lower computational requirements in conjunction with modern computer hardware, mainly GPUs, enabled deep learning to become a reality.

#### 2.4.1.2 Pooling Layer

Convolution layers are usually followed by a pooling layer. A pooling layer can be considered as a summarization layer of the input volume. The units of this layer have the same hyper-parameters with the convolution layer, such that they too have a receptive field, stride, and padding, yet instead of computing convolution operation, depending on the type of the pooling, they summarize the receptive field of each unit into a single scalar value. Typically, a pooling layer has three hyper parameters; the size of their receptive field, stride and padding. The benefit of using pooling layers is that they reduce the number free-parameters in the network and thus lower the overall computation cost of the model. The reduction of the spatial size of the model, also help to mitigate the problem of overfitting, which will be discussed in the following sections.

Common pooling types include the max and average pooling, which compute the maximum and average value from the receptive field of each unit, respectively. Commonly used hyper-parameters are filter of size 2×2 and stride of 2 which in down sample the input volume by discarding 75% of the activations. Higher pooling filter size, result into larger information loss that, in most cases is not desirable. Max-pooling aim to keep the most important features of the feature map, such us edges, while the average-pooling summarizes the input volume acting like low pass filter. Although max and average pooling **(Figure 2.23)** are widely used, mainly due to their simplicity, they "blindly" discard or mix the values of the input volume. For this reason, more advanced pooling operations have been proposed, such as a fuzzy-set based pooling operation, named "Fuzzy Pooling" (Diamantis & Iakovidis 2020), which aim to cope with the local imprecision of the feature maps. It has been shown that similar benefits with the pooling layer can be achieved by replacing them with larger filter size of convolution layers (Springenberg et al. 2014). Finally in some cases, discarding completely pooling layers can be beneficial in training, especially in the case generative models such as variational autoencoders (VAEs) (Kingma & Welling 2013) or GANs (Goodfellow et al. 2014).

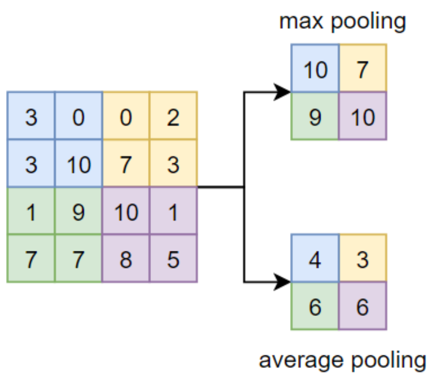


**Figure 2.23** Visual demonstration of max and average pooling with filter size 2×2 and stride 2, when applied on an input volume of spatial size 4×4.

### 2.4.1.3 Fully Connected Layer

A commonly used layer, especially in older CNN architectures, such as the AlexNet (Krizhevsky et al. 2012), fully connected layers are used as the last layers of the architecture. The reasoning behind that is that the convolution layers are used to exploit the local associations between the input signals yet and not to access whether a signal is strong enough to be considered significant. For this reason, fully connected layers, which are the classic layers of neurons found in MLPs, are used to classify the responses of the convolution layers. On the other hand, using fully connected layers in network can significantly increase the free-parameters of the overall architecture, and thus the overall computational complexity. Furthermore, as the layer is fully connected, spatial information that is embedded in the responses of the previous layers is lost. For these reasons newer architecture, such as in (Springenberg et al. 2014), discard completely the use of fully connected layers and opt for convolution layers with small filter size or by global average pooling.

To replace a fully connected layer using the global average pooling technique, a convolution layer is used in combination with a global average pooling layer. The number of feature maps of this convolution layer matches the number of neurons that would have been used with a fully connected layer. The global average pooling layer receives as input the output the convolution layer and computes the average value for each feature map. The resulting vector is then used directly as input for the output layer of the network.

#### 2.4.1.4 Normalization

Signal normalization is commonly used between the layers of CNN architectures. It consists of computational units that receives as an input the output signals of a previous layer and adjust their values according to a normalization procedure. For this reason, it is also commonly referred as "Normalization Layer". A variety of normalization operations have been proposed, including Local Response Normalization (LRN), Mean Variance Normalization (MVN) (Krizhevsky et al. 2012) and Batch Normalization (BN) (Ioffe & Szegedy 2015), which are inspired by the biological normalizations of signals that happen in brain. LRN performs a "lateral inhibition" by normalizing over input regions, which useful when ReLU type activations are used in the previous layer. This is because the ReLU neurons have unbound activations and thus, LRN aims to detect frequency features with large response. That means that if normalization is done around local neighborhood of an explicit neuron, it becomes more sensitive as compared to its neighbors.

A drawback of LRN is that it discriminates the responses that are uniformly large in any given local neighborhood, which in return the normalization diminishes them. The goal of the LRN is to encourage some kind of inhibition and boost the neurons with relatively large activations (Krizhevsky et al. 2012). The normalization can be done either to a specific channel or depth of the previous layer, or it can be extended across all depth. In both cases the size of normalization filter follows the same principles as any other layer and thus it can be configured based on the size of the receptive field that normalization is desired. MVN is working similarly to LRN layer; yet it handles the normalization differently, as it normalizes the input volume so that its values will have 0 mean and a variance of 1.

#### 2.4.1.5 The Output Layer

The last layer of a CNN architecture is typically called "output layer". The activation function used for the neurons of this layer depends on the type of prediction problem. Commonly used activation functions include the linear, logistic and softmax activation functions.

The linear activation function is also known as “identity” function, as it does not have any effect on the output of the neuron. This function is typically used in regression problems. The logistic function was discussed in Section 2.1.2 and when used as the activation function for the neurons of the output layer, each element of the resulting vector can be interpreted as a confidence score. When the softmax function is used as the activation for the neurons of the output layer, the algebraic sum of each element of the resulting vector would be 1. The element with the highest probability corresponds to the predicted class.

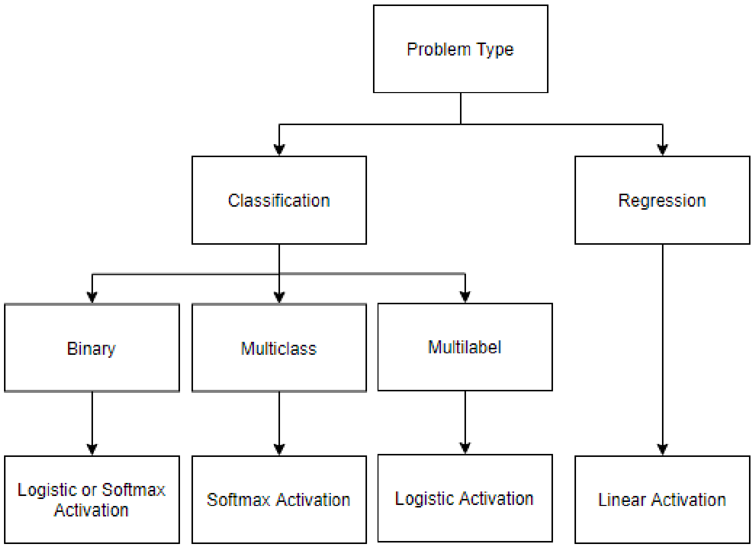


**Figure 2.24** Commonly used activation functions for the output layer of a CNN architecture.

Depending on the type of the classification problem **(Figure 2.24)**, both logistic and softmax activation functions are used. In binary classification problems the output layer can consists of one or two neurons. In the first case, the logistic activation is used, while when two neurons are used, the softmax activation is preferred. When there are more than two mutually exclusive classes (multiclass classification) the output layer consists of one neuron per class with softmax activations. When there are two or more mutually inclusive classes (multilabel classification), then the output layer has one neuron per class and the logistic activation is used.

## 2.4.2 Advancements in Convolutional Neural Networks

Since 2012 and more specifically after the spike in research interest triggered by the AlexNet (Krizhevsky et al. 2012) architecture, there have been many advancements in the field of CNNs. This section includes the most characteristic CNN architectures along with their contribution in the field of deep learning.

### 2.4.2.1 The LeNet Architecture

The LeNet (LeCun et al. 1998) architecture, is considered the father of CNN architectures. The LeNet-5 consists of 5 convolution layers, followed by pooling and two hidden fully connected layers. A summary of the architecture is illustrated in **(Figure 2.25)**.

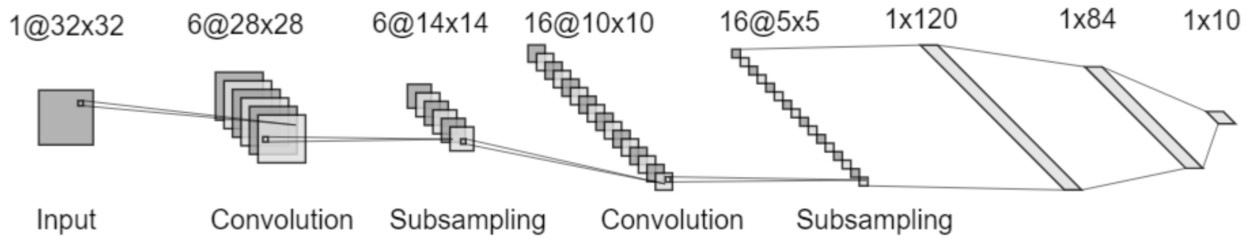


**Figure 2.25** The LeNet-5 architecture**.**

As an input the architecture uses an input layer of 1024 units which are used for raw grayscale image input of size 32×32 pixels in size. The second layer is a convolution layer which extracts 6 different feature maps with relatively large filter size of 28×28 spatial size, followed by a max-pooling layer of 14×14 filters. The second convolution layer extracts 16 feature maps of with filter size of 10×10. A second sub-sampling layer is followed with filter size 5×5 followed by 2 fully connected layer and an output layer of 10 neurons; one for each possible class of the training dataset. The architecture was trained using the MNIST dataset (Deng 2012) and used in banking industry to recognize handwritten bank notes. Compared with modern CNN architectures, such as the AlexNet (Krizhevsky et al. 2012), it is a relatively swallower architecture, using sigmoid activation functions as rectified linear units, appeared almost 15 years layer, yet the performance of the network on the classification task of handwritten digits was relatively high.

AlexNet (Krizhevsky et al. 2012) architecture, presented in Section 2.4.1, was based on LeNet-5 architecture, improved the initial model at many points. The introduction of rectified linear units (ReLU) as activation functions for the neurons of the network along with the introduction of local response normalization layers reduced drastically the overfitting issues of the initial model allowing the entire architecture to go deeper and thus improve the predictive power of the entire model. Parallel GPU training neural network training was also firstly introduced by the AlexNet authors. Furthermore, the experiments included, helped to understand the importance of smaller filter sizes for the feature maps. Finally, in this architecture the introduction of dropout layer in the last fully connected layers to reduce overfitting was adopted.

### 2.4.2.2 The ZFNet Architecture

This ZFNet architecture (Zeiler & Fergus 2014) is a CNN architecture based on the original AlexNet (Krizhevsky et al. 2012) architecture, which keeps the same number of convolution and pooling layers, with small changes on the hyper-parameters of the layers, with the most significant one to expanding the filter size of the middle convolution layers **(Figure 2.26)**. This enabled the network to achieve a top-5 error rate of 14.8% in ILSVRC 2013 challenge, marking it as the winner of the challenge that year. Although the network is relatively similar to AlexNet, it was trained on only 1.3 million images compared to AlexNet which was 15 million. The first change compared to AlexNet was the smaller filter size of the first convolution layer which changed from 11×11 to 7×7 size. This enabled the network to retain more pixel information from the input volume. The authors of that study found that the original large 11×11 filter size, opt-out a lot of the relevant, spatial information found in pixels and was the reason why ZFNet was able to be trained with much smaller dataset. The same principle was followed for the rest of the layers, decreasing the filter size in deeper layers, providing more abstract features to the final layers. Although winning the ILSVRC challenge was a major achievement for the network, the main contribution of (Zeiler & Fergus 2014) was that they used feature map visualization, which gave an understanding of how the convolution layer operates and thus provide means to "debug" existing architectures.

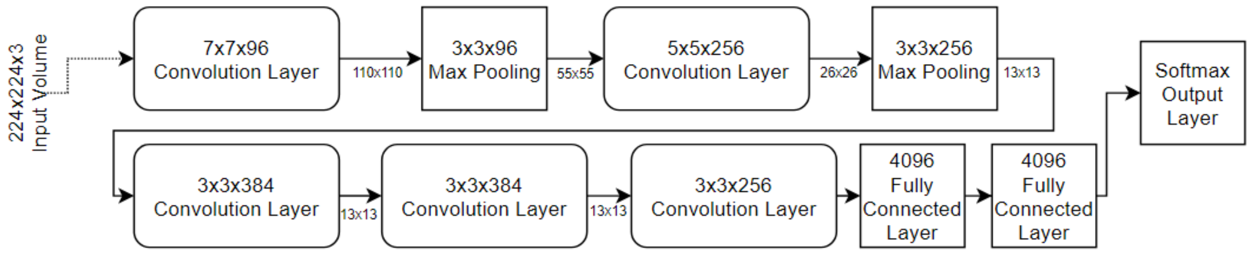


**Figure 2.26** The ZFNet architecture (Zeiler & Fergus 2014).

The ZFNet architecture was used to demontrate that classification performance of CNNs can be attributed mainly to the existence of large datasets, such as ImageNet (Deng et al. 2009), combined with the existence of powerful computational resources, such as GPUs. By that time, there was no clear understanding of how CNNs work. Their contribution to provide a visualization the feature maps, increased the general understanding of the CNNs as it became possible to visually assess the quality of the features learned during training. The feature map visualization was implemented using a methodology called "DeConvNet" which acts as a reverse convolution layer.

DeConvNet works by attaching a deconvolution layer after every convolution layer of a trained network. To examine the features that a feature map has learned in the $n^{th}$ layer, the activations of that map are held while, the rest of feature maps are set to zero. Then the feature map is passed through the deconvolution layer which has the same features as the original CNN. The input vector then passes through a series of up-sampling layers, named unpooling, rectify and filter operations, one for each preceding layer until the input space match the input volume. The result of the operation for the first two layers on ZFNet model are illustrated in **(Figure 2.28)**. More specifically it can be observed that the initial convolution layer learns more specific features about the images, like colors that reassemble close the original input volume, while moving to deeper layers the network learn more abstract features like corners.

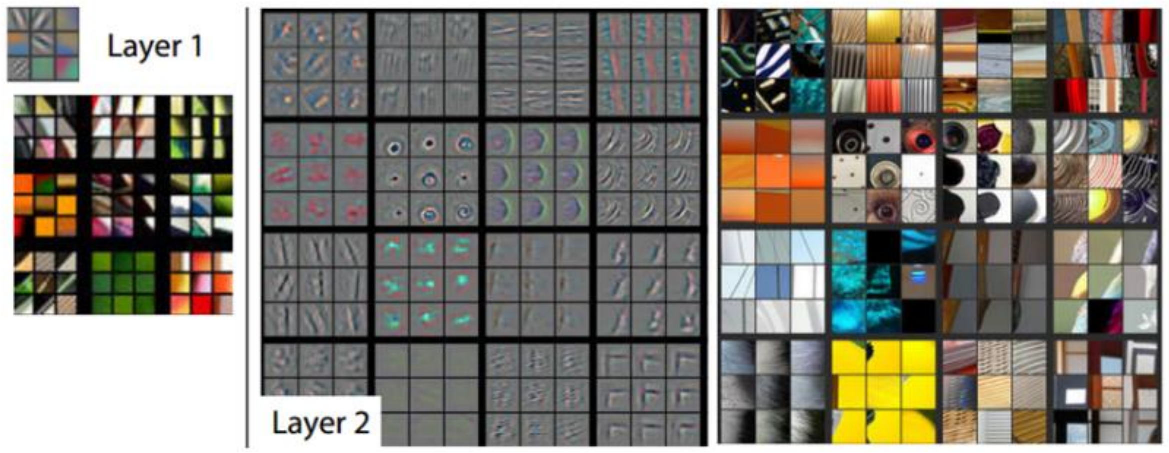
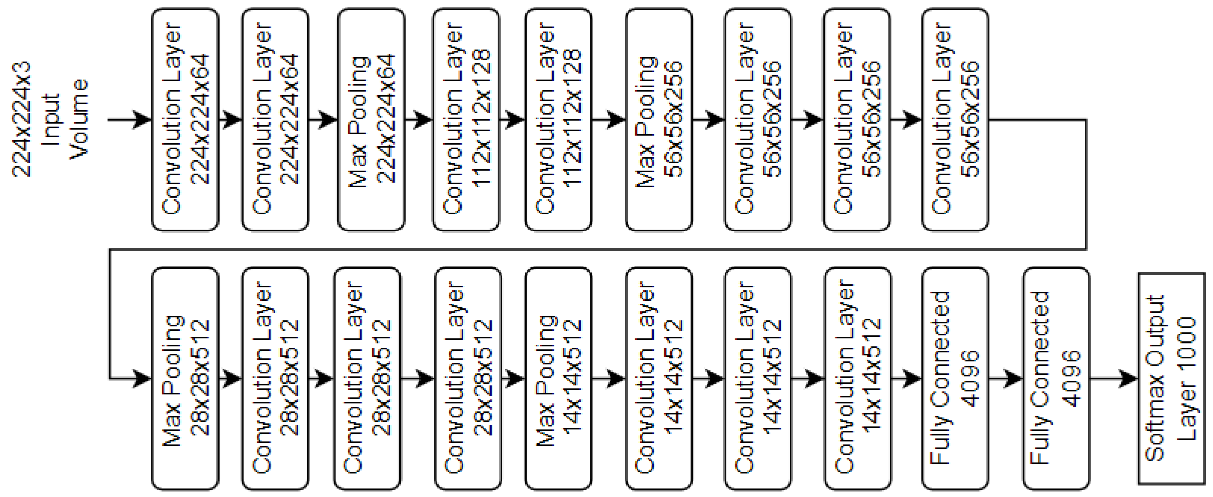


**Figure 2.27** Deconvolution operation of the first and second layer of ZFNet model using as input the images in the right  (Zeiler & Fergus 2014).

### 2.4.2.3 The VGGNet Architecture

A popular architecture, known for its depth is the “VGGNet” architecture (Simonyan & Zisserman 2014) which was proposed in ILSVCR 2014. Aims of the architecture was the exploration of deeper CNN architectures and their behavior upon training. Although the architecture is deeper than, ZFNet and AlexNet, it is relatively simple, as the receptive field of all convolution layers is fixed, with size of 3×3 and stride 1. The idea behind the fixed size filters is that the variable size of AlexNet (11×11, 5×5 and 3×3), can replicated in respect of the receptive field coverage by making use of multiple 3×3 building blocks. Two variants were proposed, one having 16 layers and another having 19 layers **(Figure 2.29)**. In respect to number of free-parameters the VGGNet architecture is relatively large, as it consists of $138 \times 10^6$ free-parameters in its smaller version (16 layers), which make the training process a relatively computational and resource demanding operation, prone to overfitting.



224x224x3 Input Volume
Convolution Layer 224x224x64
Convolution Layer 224x224x64
Max Pooling 224x224x64
Convolution Layer 112x112x128
Convolution Layer 112x112x128
Max Pooling 56x56x256
Convolution Layer 56x56x256
Convolution Layer 56x56x256
Convolution Layer 56x56x256
Max Pooling 28x28x512
Convolution Layer 28x28x512
Convolution Layer 28x28x512
Convolution Layer 28x28x512
Max Pooling 14x14x512
Convolution Layer 14x14x512
Convolution Layer 14x14x512
Convolution Layer 14x14x512
Fully Connected 4096
Fully Connected 4096
Softmax Output Layer 1000

**Figure 2.28** The VGGNet-19 architecture (Simonyan & Zisserman 2014).

| Network Configuration | | | | | |
|---|---|---|---|---|---|
| A | A-LRN | B | C | D | E |
| 11 Trainable Layers | 11 Trainable Layers | 13 Trainable Layers | 16 Trainable Layers | 16 Trainable Layers | 19 Trainable Layers |
| 224×224×3 Input | | | | | |
| Conv: 3×3×64 | Conv: 3×3×64<br>**LRN** | Conv: 3×3×64<br>**Conv: 3×3×64** | Conv: 3×3×64<br>Conv: 3×3×64 | Conv: 3×3×64<br>Conv: 3×3×64 | Conv: 3×3×64<br>Conv: 3×3×64 |
| Max Pooling | | | | | |
| Conv: 3×3×128 | Conv: 3×3×128 | Conv: 3×3×128<br>**Conv: 3×3×128** | Conv: 3×3×128<br>Conv: 3×3×128 | Conv: 3×3×128<br>Conv: 3×3×128 | Conv: 3×3×128<br>Conv: 3×3×128 |
| Max Pooling | | | | | |
| Conv: 3×3×256<br>Conv: 3×3×256 | Conv: 3×3×256<br>Conv: 3×3×256 | Conv: 3×3×256<br>Conv: 3×3×256 | Conv: 3×3×256<br>Conv: 3×3×256<br>**Conv: 1×1×256** | Conv: 3×3×256<br>Conv: 3×3×256<br>**Conv: 3×3×256** | Conv: 3×3×256<br>Conv: 3×3×256<br>Conv: 3×3×256<br>**Conv: 3×3×256** |
| Max Pooling | | | | | |
| Conv: 3×3×512<br>Conv: 3×3×512 | Conv: 3×3×512<br>Conv: 3×3×512 | Conv: 3×3×512<br>Conv: 3×3×512 | Conv: 3×3×512<br>Conv: 3×3×512<br>**Conv: 1×1×512** | Conv: 3×3×512<br>Conv: 3×3×512<br>**Conv: 3×3×512** | Conv: 3×3×512<br>Conv: 3×3×512<br>Conv: 3×3×512<br>**Conv: 3×3×512** |
| Max Pooling | | | | | |
| Conv: 3×3×512<br>Conv: 3×3×512 | Conv: 3×3×512<br>Conv: 3×3×512 | Conv: 3×3×512<br>Conv: 3×3×512 | Conv: 3×3×512<br>Conv: 3×3×512<br>**Conv: 1×1×512** | Conv: 3×3×512<br>Conv: 3×3×512<br>**Conv: 3×3×512** | Conv: 3×3×512<br>Conv: 3×3×512<br>Conv: 3×3×512<br>**Conv: 3×3×512** |
| Max Pooling | | | | | |
| Fully Connected 1×4096 | | | | | |
| Fully Connected 1×4096 | | | | | |
| Fully Connected 1×1000 | | | | | |
| Softmax | | | | | |

**Figure 2.29** Variations of VGGNet architecture from smaller (A) to larger (E) (Simonyan & Zisserman 2014**).**

Due to the large number of free-parameters, (Simonyan & Zisserman 2014) used an incremental approach to train the network, in which training starts with a smaller, swallow network. Every time that the network converges, the training process pauses, more layers are added and training resumes, effectively using the trained layers as initializers for the untrained, newly added layers. The network used for each incremental training stage is illustrated in **(Figure 2.29)**. Although this methodology is effective, it is relatively time consuming, as it requires an entire network to be trained before it can be used as an initializer for a deeper network. The VGGNet-16 variant was trained in ILSVRC-2012-validation dataset achieving a top-5 error of 7.5% and on ILSVRC-2012-test with top-5 error of 7.4%.

### 2.4.2.4 The GoogLeNet and the Inception-Based architectures

GoogLeNet (Szegedy et al. 2015) also known as Inception-v1 was the winner of  ILSVRC-2014 challenge, with top-5 error rate of 6.7%, and introduced a ra dically different CNN architecture design. Compared to the VGGNet architecture that was proposed in the same challenge, GoogLeNet, over 100 layers organized in the so-called "inception" modules **(Figure 2.30)**. The inception module aims to minimize the computational resources required by the process of stacking multiple convolution layers together along with the problem of overfitting due to the increased number of free-parameters.

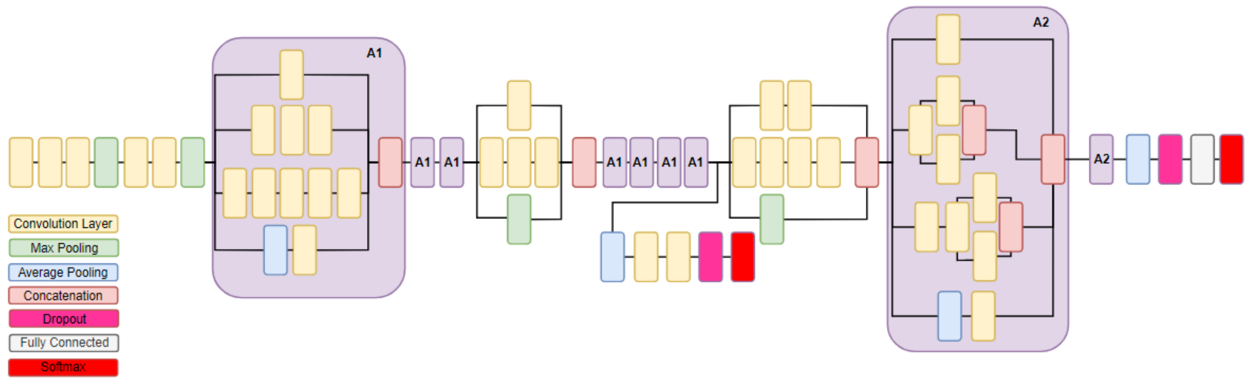


**Figure 2.30** The GoogLeNet architecture (Szegedy et al. 2015)**.**

Compared to conventional CNNs, the inception module, illustrated in **(Figure 2.31)**, process the input volume in parallel. In the architecture, each inception module has the chance to perform both convolution and pooling operations using multiple filter sizes, as opposed of having one operation per layer. The output of all parallel layers is then concatenated creating a feature reach representation of the input volume, which is then passed to the next module of the network. To manage the increased number of free-parameters introduced by the multiple feature map concatenation, inception module uses 1×1 convolutions, also known as pointwise convolutions, which control the number of feature maps.

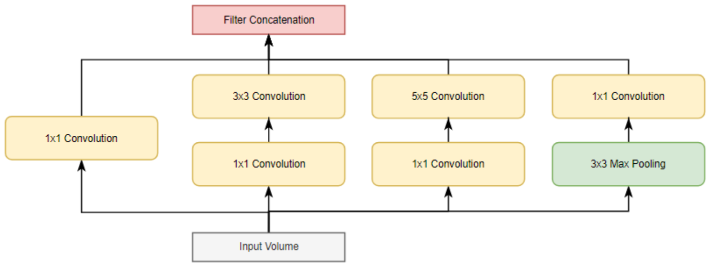


**Figure 2.31** The inception module (Szegedy et al. 2015)**.**

In total the architecture features 9 inception modules, with over 100 layers. Furthermore, the architecture does not use any fully connected layer, following a similar approach with the fully convolutional architecture (Springenberg et al. 2014), which reduced the number of free parameters by 12 times compared to AlexNet. ReLU activations along with max-pooling was employed to deal with the overfitting problem which improve the nonlinearity of the network. By replacing the conventional output fully connected layer average pooling layer, reduced the input volume from 7×7×1024 to 1×1×1024.

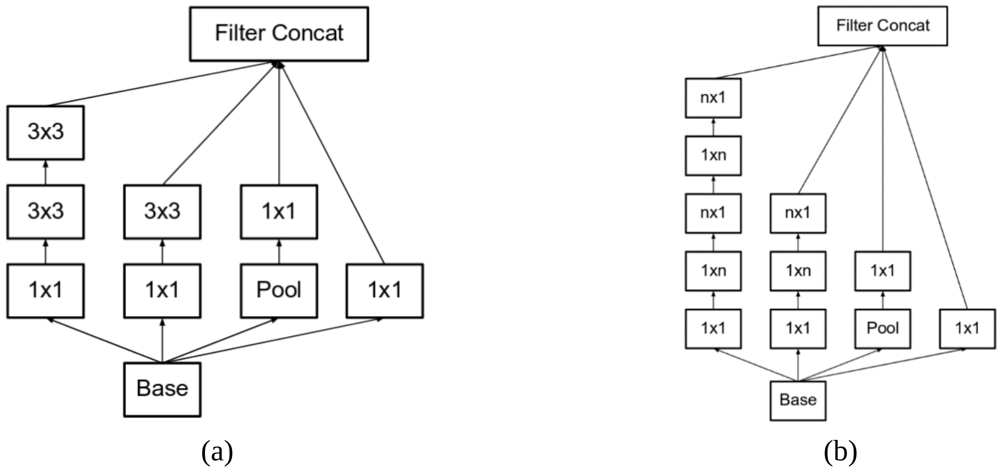


**Figure 2.32** (a) 5×5 convolution expressed as 3, 3×3 convolution layers (b) N×N factorized convolution (Szegedy et al. 2016)**.**

Variations of GoogLeNet architecture include, Inception-v2 and Inception-v3 (Szegedy et al. 2016) and feature increased classification accuracy along with computational performance improvements. Inception-v2 aimed to reduce the representational bottleneck of inception module, which arises from severe dimensionality reduction, introduced typically by the pointwise convolution usage. To increase the computational efficiency of the network, the authors of that study proposed to use factorized convolutions. As an example, using this methodology a 5×5 convolution layer can be expressed in 3, 3×3 convolution layers, which computationally wise, is 2.78 times computationally cheaper. This is illustrated in **(Figure 2.32a)**. This can be extended to any N×N convolution layer which can be expressed as a combination of 1×N and N×1 convolutions **(Figure 2.32b)**. The authors found that this methodology, can be 33% cheaper in the case of 3×3 convolutions.

In an effort to battle the bottleneck problem, the filter banks found in the original Inception module were also expanded, making them wider instead of deeper. The authors found that in Inception-v2 the auxiliary classifiers did not contribute a lot especially in the end of the architecture, mainly due to neuron saturation. For this reason in Inception-v3 variant, they enhanced Inception-v2 with factorized 7×7 convolutions, batch normalization (Ioffe & Szegedy 2015) and trained using label smoothing, which was proposed as a regularization method, and the RMSProp optimizer (Hinton et al. 2012).

### 2.4.2.5 The ResNet architecture

The ResNet architecture (He et al. 2016), was introduced in ILSVRC-2015 challenge winning it with a remarkably low top-5 error rate of 3.6%, considerably lower than the pre-accentors and for the first time, outperforming the human top-5 error rate which is between 5 and 10%. The architecture was relatively deep, yet simple. It introduced the concept of residual blocks, which made use of residual connections, similarly stacked together as in the case of Inception model, forming a deep CNN architecture of 152 layers **(Figure 2.33)**.

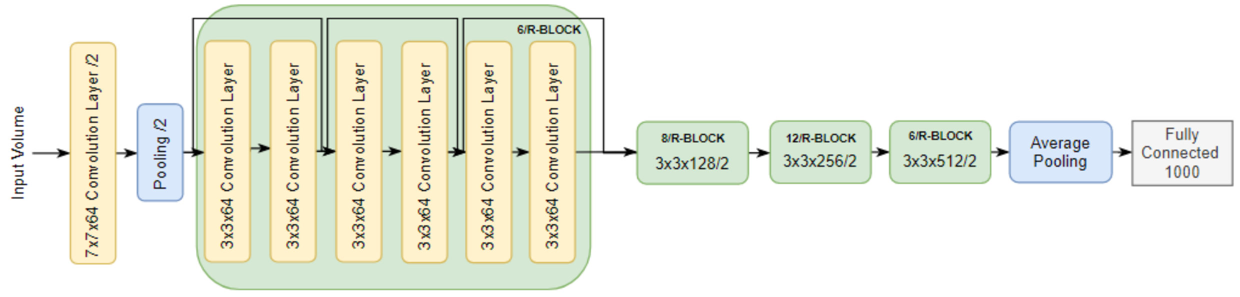


**Figure 2.33** A ResNet architecture variant with 34 layers (He et al. 2016)**.**

A residual block consists of a three-step process, convolution – ReLU – convolution forming a block that implements $\boldsymbol{H}(\boldsymbol{x}) = \boldsymbol{F}(\boldsymbol{x}) + \boldsymbol{x}$. Notice the addition of $\boldsymbol{x}$ in the function. This is the residual connection of the block which adds the input volume back to the output of the residual block. This addition was introduced to battle the problem of vanishing gradient as it enables an easier flow of the gradient in the backpropagation backward pass **(Figure. 2.34)**. Using residual modules, enabled the architecture to become very deep and still trained effectively, without facing extreme overfitting problems. It should be noted that the architecture that won the ILSVCR challenge was trained on 8 high end GPUs for three weeks. Although the residual block is effective in battling the problem of vanishing gradient, it does not prevent overfitting. This was demonstrated by the network authors (He et al. 2016) by experimenting on a 1202 layer ResNet architecture which significantly under-performed when compared to the 152 layer ResNet, which was attributed to the increased number of free-parameters which led to overfitting.

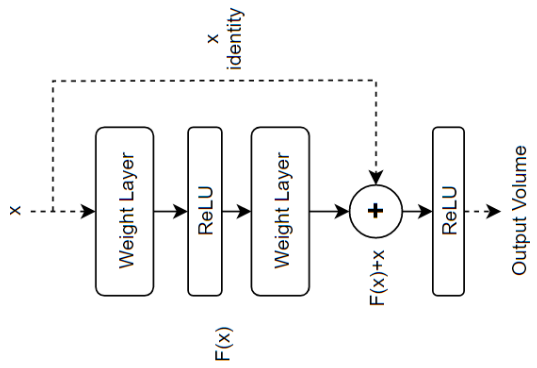
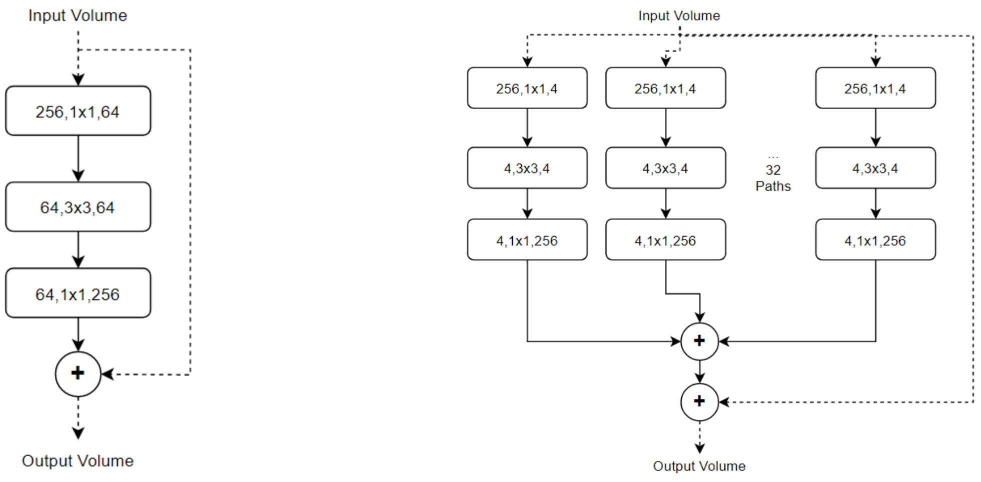


**Figure 2.34** A visual representation of the residual block (He et al. 2016).



(a) (b)

**Figure 2.35** A visual representation of (a) the original residual block of ResNet and (b) a residual module in ResNeXt model with $c = 32$.

ResNet inspired many architectures that followed. More specifically Inception-v4 and Inception-ResNet (Szegedy et al. 2017) variants are improvements of Inception-v2 and Inception-v3 architectures, that aimed to explore the residual connections and formalize the original inception module, achieving higher classification performance with lower computational requirements. The ResNeXt (Xie et al. 2017), which appeared and won the ILSVRC-2017 challenge was also inspired by the original ResNet design. The architecture enhanced the residual module with multiple parallel convolutions, which perform an aggregated transformation of the input volume. The number of parallel blocks ($c$) define the cardinality of the module, which is in effect a new dimension on top of the width and depth of the original ResNet module. An example of such module with $c = 32$ is illustrated in **(Figure 2.35)**.

### 2.4.2.6 The DenseNet architecture

The Densely Connected Convolutional Network (Huang et al. 2017) also known as DenceNet, extends the residual connections in such a way that every layer in the network is connected with every other layer, in a feed-forward fashion, meaning that a DenseNet has $\frac{n(n+1)}{2}$ residual connections. Each layer of the network has the feature maps of all previous layers as input. The advantage of this is that the gradient is not vanished the time it reaches the end of the network.

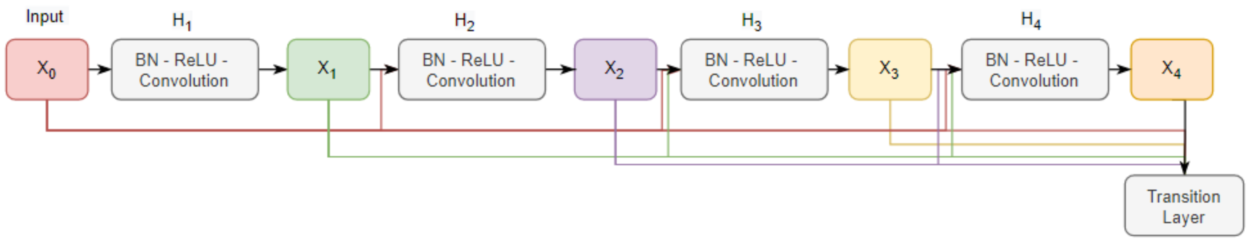


**Figure 2.36** A visual representation of 5-layer dense bock (Huang et al. 2017).

The DenseNet architecture can be composed by four types of dense blocks. The first is the basic DenseNet composition layer, in which each layer is followed by batch normalization, ReLU activations and a 3×3 convolution layer. The second type is called “BottleNeck” or DenseNet-B, in which a pointwise convolution is inserted between a DenseNet composition layer. The third type of layer, named DenseNet Compression (DenseNet-C), focuses on the model compactness as it tries to reduce the feature maps within a transition layer which introduces a compression factor between 0 and 1. When this compression factor is 1 then all feature maps are retained, yet when its smaller it the number of feature maps will change accordingly. When both bottleneck and transition layer are used with a compression factor lower than 1 the model is referred as DenseNet-BC. The last type of DenseNet is composed by multiple dense blocks and transition layers in which the dense blocks are followed by pointwise convolutions and 2×2 average pooling layers. The feature maps in this type of dense block, have the same size and thus the transition layer output can be concatenated forming a feature reach representation of the block. To reduce the computational complexity, fully connected layers are replaced by global average pooling followed by a Softmax classifier.

### 2.4.2.7 Mobile Oriented CNN architectures

Although the advancements in CNN architectures towards increasing their classification performance are plenty, one of the main drawbacks is that as their depth increases, and their computational complexity follows the same trend. This results in CNN architectures, that although they can be very powerful, they are unable to perform inference in a computationally limited environment, such as mobile phones and embedded devices. The main reason behind that is the high number of free-parameters which result in relatively large number of Floating-Point Operations (FLOPs) which can only be executed, in reasonable time, in high-end systems equipped with modern GPUs. To deal with this limitation, a variety of architectures have been proposed that typically require much lower computational resources, usually in expense of classification performance.

#### 2.4.2.7.1 The SqueezeNet architecture

The SqueezeNet (Iandola et al. 2016) architecture, was one of the first mobile-oriented CNN architectures. The idea behind is architecture was to create a neural network composed by efficient building blocks, aiming to reduce the number of free-parameters of the network. Compared to the

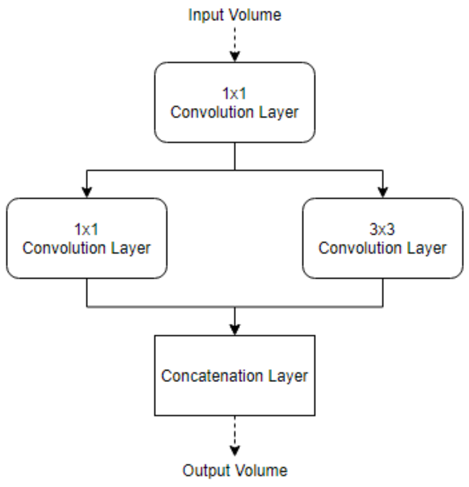


**Figure 2.37** Fire module visual representation.

AlexNet (Krizhevsky et al. 2012) architecture which uses five convolution layers with large filter sizes and fully connected layers, SqueezeNet uses small pointwise convolutions combined with convolution layers having a filter size of 3×3. The building block behind the architecture is known as the “Fire module”. The first layer of the module is a pointwise convolution layer which squeezes the input volume. The output of the layer is then parallelly guided to another pointwise convolution and a convolution layer with filter size 3×3. The output of them is then concatenated. This is illustrated in **(Figure 2.37)**.

The SqueezeNet architecture consists of eight fire modules in succession, where between them max-pooling is used for spatial dimensionality reduction. To minimize the free-parameters the architecture is fully convolutional, since no fully-connected layers are used. Following similar approach to DenseNet type models, global average pooling is employed along with softmax as the output layer of the network. In total the network consists of $1.25\times10^6$ free-parameters, making it 50 times smaller than AlexNet, while managing to maintain the same classification accuracy.

### 2.4.2.7.2 The MobileNet architectures

A family of network architectures that were designed specifically for mobile application usage are known as MobileNets. The first version of the architecture was named MobileNet-v1 (Howard et al. 2017), made use of the concept of depthwise separable convolutions, originally proposed in (Chollet 2017), to replace the computationally expensive convolutions. The architecture is relatively simple, as it consists of, usually 13, modules that depthwise separable convolution, batch normalization and pointwise convolution to fuse the separable convolutions. The module is illustrated in **(Figure 2.38)**. Notice the capped ReLU activations used after the batch normalization. These are normal ReLU activations but with a capped output, which in the case of MobileNets is 6. The ReLU activation can be expressed as $ReLU(x, x_{max}) = \min(\max(x, 0), x_{max})$. The benefits of the capped ReLU version include the increased speed in GPU inference and help the network to learn more sparse features earlier. In total the MobileNet-v1 architecture consists of $4.24\times10^6$ free-parameters which are three times smaller than SqueezeNet, yet, in ImageNet classification, MobileNet-v1 outperforms the later by 20% and 9% in the top-1 and top-5 error score respectively.

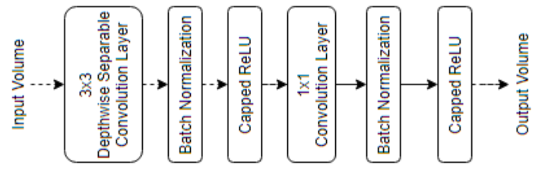


**Figure 2.38** The basic building block of the MobileNet-v1 architecture (Howard et al. 2017).

The architecture includes a hyper-parameter named “depth-multiplier”, also known as “width-multiplier”, which controls the number of feature maps extracted by the convolution layers. This parameter effectively balances the computational complexity versus the classification performance of the network. Depth-multiplier should be selected according to the application needs. Choosing a multiplier smaller than 1, results in a smaller computationally efficient network, with low classification performance.

MobileNet-v2 (Sandler et al. 2018) is an updated version of the original MobileNet-v1 architecture, which increases the classification performance while reducing the computational complexity. It has ~1 million lower number of free-parameters. The architecture uses depth-wise separable convolutions, yet arranges them with inverted residual connections **(Figure 2.40)**. The idea behind this module is to expand, reduce and then re-expand the input volume, which is the opposite of SqueezeNet approach.  Same as the first version, the network uses the depth-multiplier hyper-parameter to control the balance of computation efficiency and classification performance.

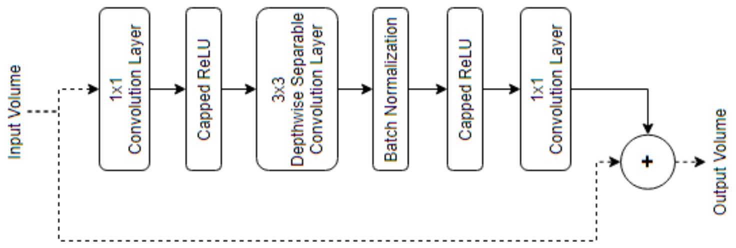


**Figure 2.39** The basic building block of the MobileNet-v2 architecture (Sandler et al. 2018).

MobileNet-v3 (Howard et al. 2019) is the third revision of the MobileNet architecture. Instead of revising the original architecture, the authors, used MnasNet-A1 (Tan et al. 2019) mobile CNN architecture and adjusted using NetAdapt (T.-J. Yang et al. 2018) algorithm. NetAdapt is an algorithm that by automatically simplifying a pre-trained model, reaches to a given latency, which is translated to computational efficiency, while keeping the classification performance at a certain threshold. The MobileNet-v3 architecture was also adjusted manually, by adjusting the computationally expensive layers of the network, use "swish" activations instead of capped ReLU and introduced squeeze-and-excitation modules.

The authors identified that the regular 3×3 convolution layers with 32 filters used in the previous version of the network, although have low number of free-parameters, are computationally expensive, mainly due to the spatial size of the input feature maps. Experiments showed, that using 16 filters were sufficient to reduce the complexity without damaging the classification performance. The capped ReLU activations were replaced with hard-swish activation functions, also known as h-swish, $h_{swish(x,x_{max})} = x * \frac{ReLU\left(x+\frac{x_{max}}{2},x_{max}\right)}{x_{max}}$. This activation allows for a small negative slop to be present, compared to conventional ReLU activations, which was found to be beneficial to the overall performance, with more profound effect when used in deeper layers of the network. To reduce the computational complexity, h-swish was only used in the deeper layers. A visual illustration of the main building block of the architecture is illustrated in **(Figure 2.40)**.

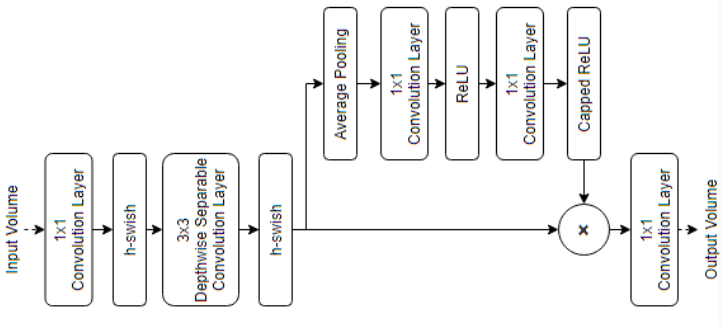


**Figure 2.40** The basic building block of the MobileNet-v3 architecture (Howard et al. 2019).

The final optimization of the architecture involves the last layers of the model, in which, the previous architectures used a pointwise convolution that expand the feature maps just after the global average pooling layer. This was done so that the classification layer will have enough features for reasoning. Although that worked well, it introduced a bloated layer that was computationally expensive compared to the rest of the network. For this reason, in MobileNet-v3 architecture this layer was placed before the global average pooling layer, so that it works with the much smaller feature maps of size 1×1 instead of 7×7. This change enabled the authors to remove the previous bottleneck layer, without classification performance loss. The authors released several variations of the architecture, depending on the classification and computational performance requirements, with the smallest requiring $2.9\times10^6$ number of free parameters and the largest $5.4\times10^6$. Similar to the previous versions of the network, the smaller variations sacrifice the classification performance in favor of computational efficiency.

### 2.4.2.7.3 The BlazeFace architecture

The BlazeFace (Bazarevsky et al. 2019) architecture, is a mobile oriented CNN architecture that aims to perform accurate face detection in mobile devices with sub-millisecond inference times. Although the architecture was designed to be used as a feature extractor for object detection task and not a general-purpose CNN, it is worth mentioning as it consists of only $0.1\times10^6$ free-parameters. The architecture is inspired by the MobileNet-v1 architecture, extending it with residual connections. The main building block of BlazeFace is illustrated in **(Figure 2.41)**. The most profound changes to the MobileNet-v1 architecture was the increase of the filter size in the first depthwise separable convolution layer, to 5×5, similarly to MnasNet-A1 (Tan et al. 2019). Instead of using the MobileNet-v2 residual connections, which are between the bottleneck layers, BlazeNet added them on the depthwise separable convolution layers, in which, to match the output volume feature maps, channel padding is used after max-pooling. This effectively adds empty feature maps to the input volume. Finally, the activations used were simply ReLU instead of capped ReLU. In total the architecture consists of 16 modules, some of which include stride of 2 in the depthwise separable convolution. When stride is used the residual connection spatial dimensionality reduction is performed using max-pooling, while in the rest of the cases only channel padding is used.

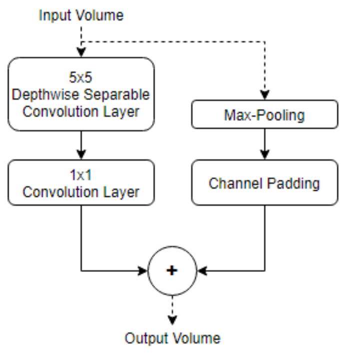


**Figure 2.41** The basic building block of the BlazeFace architecture (Bazarevsky et al. 2019). The max-pooling layer is used only when the depthwise separable convolution layer includes a stride of 2.

### 2.4.2.7.4 The SqueezeNext architecture

SqueezeNext (Gholami et al. 2018) is an architectural improvement of the original SqueezeNet model. The basic building block of the architecture consists of two bottleneck layers in series, in which the first reduces the feature maps in half and the second by four. The original convolution layer with 3×3 filter size that follows the bottleneck layers is replaced with two equivalent factorized convolution layers, 3×1 and 1×3 respectively, which increase the depth of the model. The final layer is an expansion pointwise convolution layer which increases the feature maps to be equal with the ones found in the input volume. To this the input volume of the block is added using

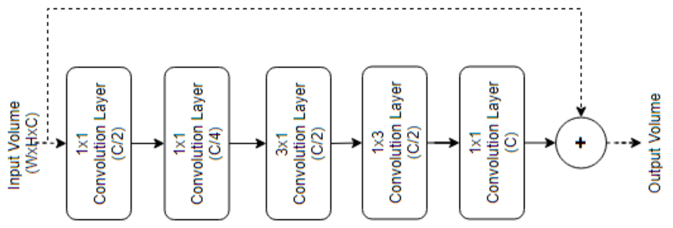


**Figure 2.42** The basic building block of the SqueezeNext architecture (Gholami et al. 2018).

a residual connection. This basic building block of the architecture is illustrated in **(Figure 2.42)**. Similarly to MobileNets, the SqueezeNext architecture includes a hyper-parameter that can control the size of the model. The authors examined four variants of the model, with the first been the lightest, requiring $0.7 \times 10^6$ free-parameters and achieving 59.05% and 82.60% top-1 and top-5 accuracy in ImageNet classification task.

### 2.4.2.7.5 The ShuffleNets architectures

The ShuffleNet (Zhang et al. 2018) architecture is one of the first CNN architectures specifically designed to be computationally efficient. The idea behind the model was that many modern architectures use pointwise convolutions which are relatively expensive in terms of FLOPs. To deal with this issue, grouped convolution were used instead, which although they easier to compute they introduce side effects that harm the classification performance of the network. In an effort to mitigate these effects, channel shuffling was introduced, from which the name of the network derives. The principle behind the grouped convolutions is to divide the input volume feature maps into groups on which the convolution operation is performed. Having, for example, two groups, only half parameters are required as each convolution filter only works on half the input volume. It is important to state here that the free-parameters between the groups are not shared, so each group needs its own set of parameters.

The grouped convolution is similar to depthwise convolution and in fact, the later can be implemented using a generalized version of the grouped convolution. The main difference between the two is that for depthwise convolution, each feature map forms its own group and thus it has one output feature map. Using the generalized form of grouped convolutions, the number of output feature maps of each group do not have to equal the number of input feature maps in the group.

The main drawback of the grouped convolution approach is that the output derives from a fraction of the original input. As a solution to this problem, the feature maps can be shuffled after the group convolution, which in effect rearranges the output feature maps along the depth dimensions. This is illustrated in **(Figure 2.43)**.

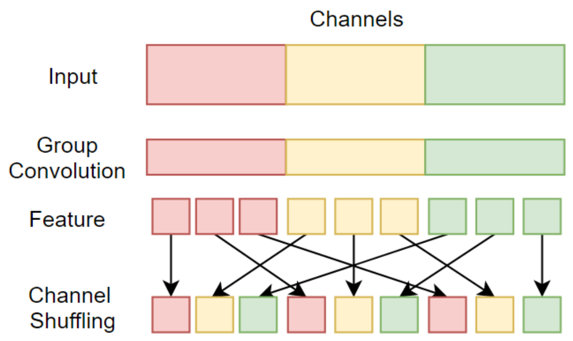


**Figure 2.43** Channel shuffling visualization after grouped convolution.

The basic building block of the ShuffleNet architecture is illustrated in **(Figure 2.45)**. In total the network consists of 8 building blocks. An interesting side effect of grouped convolutions is that the more groups the architectures consist of, the smaller the computational footprint is. Examining the block, the first layer is a bottleneck layer performed using a grouped convolution with filter size of 1×1 and reduces the number of feature maps to four. This is followed by channel shuffling which is then followed by a depthwise separable convolution of filter size 3×3, which is followed by batch normalization yet without an activation function. The final layer consists of a grouped convolution without channel shuffling in which a residual connection is added using the standard algebraic sum operation. To reduce the spatial size, some blocks use stride of 2 in the last layer, where in that case, the input feature maps are concatenated to the output feature maps, matching their dimensions using average pooling with pooling size of 3×3. The final ShuffleNet architecture scores 68% and 86.4% top-1 and top-5 score in ImageNet classification challenge having $2.4\times10^6$ free-parameters.

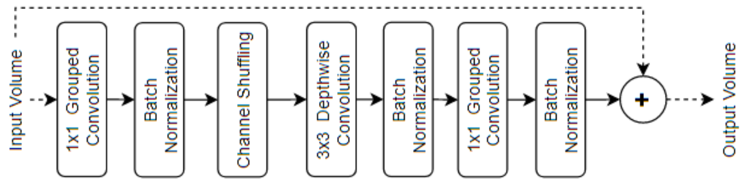


**Figure 2.44** The basic building block of the ShuffleNet (Zhang et al. 2018) architecture.

The ShuffleNet-v2 architecture (Ma et al. 2018) is a revised version of the original ShuffleNet design. The authors primarily focused on measuring the computational performance overhead of each layer on specific, mobile hardware. The conclusions of that study was to opt-out the use of grouped convolutions, as although they require fewer free-parameters, to increase their classification performance, more feature maps were required. The result of this was increased Input and Output (I/O) on RAM which in return results into slower network performance. The revised building block of ShuffleNet-v2 is illustrated in **(Figure 2.45)**. It can be noted that the original grouped convolutions were replaced by a channel split operation in which half of the channels are sent as is through a residual connection to the feature map concatenation layer and the rest follow the bottleneck block. The bottleneck layer is now performed using regular pointwise convolution instead of grouped which is then followed by batch normalization and 3×3 depthwise separable convolution. Another change is that the algebraic sum of the residual connection has been replaced with a concatenation layer. The reasoning behind this change is that the elementwise operations, are relatively expensive compared to concatenation. The last layer of the block is a channel shuffling layer, similar to the original ShuffleNet architecture. The architecture uses a scale factor as a hyper-parameter to control the number of feature maps of the model, which in return controls the classification performance versus the computational complexity of the network. Using a scale factor of 1, the architecture performs similarly to the original ShuffleNet, having 69.4% and 88.9% top-1 and top-5 score respectively in ImageNet classification challenge using $2.3\times10^6$ free parameters. The largest ShuffleNet-v2 variation consists of $6.7\times10^6$ free parameters and scores 77.1% and 93.3% top-1 and top-5 score, respectively. Although the network has higher classification performance with lower number of free parameters when compared to MobileNets, the model is not optimized for GPU usage and as a result, MobileNets are more widely adopted.

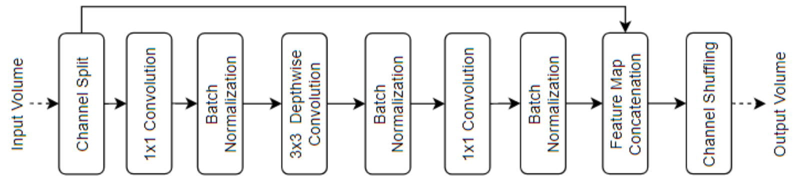


**Figure 2.45** The basic building block of the ShuffleNet-v2 (Ma et al. 2018) architecture.

Similar approach with the ShuffleNet-v1 architecture was followed by CondenceNet-v1 (Huang et al. 2018) and the newer CondenceNet-v2 architectures (Yang et al. 2021), which used the DenceNet architecture as a starting point and included grouped convolutions with channel shuffling layers along with an alternation to the residual connection where feature map concatenation was used instead of algebraic sum. The lightweight version of CondenceNet-v1 architecture, which consists of 8 building blocks and can be compared with ShuffleNet-v1, scores 71% and 90% top-1 and top-5 accuracy in ImageNet classification challenge and requires $2.9\times10^6$ free parameters.

### 2.4.2.7.6 Other Computationally Efficient CNN Architectures

Nowadays the need for computationally efficient CNN architectures is becoming more apparent, especially by the increased use of ML in mobile devices. In the previous sections a variety of mobile oriented architecture was presented, covering the most notable architectures and their contributions. There is a plethora of other architectures that aim towards CNNs with lower computational footprint. Such architectures include, the ESPNet-v1 (Mehta et al. 2018) and a lighter version of it, ESPNet-v2 (Mehta et al. 2019), which focused on decomposing the standard convolution layer into pointwise convolutions. These are followed by spatial pyramids of dilated convolutions. The lighter version of ESPNet-v2 consists of $3.49\times10^6$ free parameters and achieves 72.1% and 90.4% on top-1 and top-5 accuracy respectively in ImageNet classification challenge. The DiCENet (Mehta et al. 2020) architecture in an effort to reduce the computational complexity replaces the regular convolutions with dimension-wise convolution and fusion. The light version of the network consists of only $1.81\times10^6$ free parameters scoring 66.5% and 86.6% top-1 and top-5 accuracy, yet heavier version has $3.98\times10^6$ free parameters and has similar classification accuracy with ESPNet architecture. FBNet (Wu et al. 2019) and ChamNet (Dai et al. 2019) architectures, similarly to (Tan et al. 2019) focused on improving the NAS algorithm, yet in the context of not having to train and evaluate each potential architecture separately. The lighter versions of the proposed networks, named FBNet-A and ChamNet-C have $4.3\times10^6$ and $3.4\times10^6$ free parameters, respectively. GhostNet (Han et al. 2020) which focused on reduction of number of feature maps and lower elementwise operations. Furthermore, the authors focused on the creation of extra feature maps using linear, cheap operations and more specifically depthwise convolutions. The lightest version of GhostNet scores 66.2% and 86.6% top-1 and top-5 accuracy in ImageNet classification, and consists of $2.6 \times 10^6$ free parameters. The MixNet (Tan & Q. V. Le 2019) architecture introduced mixed convolutions which use multiple filter sizes in a single convolution layer. The idea behind it is that the filter size matters, when it comes to classification performance, so instead of deciding a single size, the authors used filter sizes of 3×3, 5×5 and 7×7. The light version of the MixNet architecture consists of $4.1 \times 10^6$ free parameters and 75.8% and 92.8% top-1 and top-5 accuracy in ImageNet classification. EfficientNet (Tan & Q. Le 2019) is a widely used mobile oriented architecture that includes three different hyper-parameters that control the computational efficiency and classification performance of the architecture. This enables the network to be used in a variety of different applications, including conventional GPU enabled deployments and mobile or embedded usages. The first hyper-parameter controls the width of the network, and it controls the number of filters used per convolutional layer. The second controls the depth of the network by controlling the number of convolution layers used in its blocks. The last parameter controls the input resolution, which conventionally is 224×224 but can be scaled up to increase the accuracy. The light version of EfficientNet features $4.7 \times 10^6$ free parameters and scores 74.8% and 92.2% top-1 and top-5 accuracy in ImageNet classification task, while one of the largest versions, requires $13 \times 10^6$ free parameters and scores 81.5% and 95.7% top-1 and top-5 accuracy in the same task. LB-FCN *light* (Diamantis et al. 2019) architecture, which will be discussed in the next sections use similar filter sizes with the MixNet (Tan & Q. V. Le 2019) architecture, yet operates in principle of multi-scale feature extraction using parallel convolutions to increase the classification performance and maintain low number of free parameters. The light version of the architecture consists of only $0.3 \times 10^6$ free-parameters making it one among the smallest in the mobile-oriented CNN architecture list. Concluding, there is a large variety of computationally efficient CNN architectures that have been proposed in literature, deciding the “best” is a question of finding the one that performs better in based on the available hardware and operating system.

### 2.4.3 Generative Adversarial Networks

The performance of discriminative models such as CNNs were revolutionary in the field of CV. Deep generative models were less impactful since they require to estimate complex probability distributions of high dimensionality, which in the past was often considered impossible. GANs, introduced by (Goodfellow et al. 2014), overcome this issue by combining two deep models, from which one is acting as a generative model (Generator) and another acting as a discriminative model (Discriminator). While training, the two networks act as opponents in game were the generator tries to fool the discriminator by producing real-like fake samples. The introduction of GANs was relatively successful in the context of data generation and with the most profound effect in the field of CV, mainly because of their remarkable performance in generating real-like images. Their applications include, image generation (Goodfellow et al. 2014; Diamantis et al. 2019), image-to-image translation such as style transfer (Karras et al. 2019) and super resolution (Ledig et al. 2017), image generation from text (Reed et al. 2016) and even music generation (Gui et al. 2020).

GANs belong to a broad range of algorithms called generative models. The basics of a generative model is to map a simple probability distribution, such as a uniform distribution, to a more complex one. The complex probability distribution is named model distribution. In effect a generative model tries to map a probability distribution to a high-dimensional space distribution. To understand that, let us consider the process of generating images that resemble cats using random pixels. In that brute-force approach, not all images would resemble cats. In fact, some pixels can be considered to have lower probability than others when it comes to their contribution in the cat generation. The result of this process is that the cat image follows a high-dimensional probability distribution over the input space. Therefore, the problem of generating a new image of a cat is equivalent to the problem of generating a new vector that follows the "cat probability distribution" over the N dimensional vector space. The problem can be considered as generating a random variable with respect to a specific true dimensional distribution. The true data distribution is relatively hard or impossible to be defined. For this reason, an empirical approach is employed which approximates the true data distribution through training using samples from the same distribution that we try to approximate. There are many generative models that aim to approximate true data distributions. They can be classified into two main categories: explicit and implicit. Explicit generative models have access to the model likelihood function and are typically trained by maximizing the likelihood. They are commonly used in probabilistic modeling as training procedure optimizes a well-defined quantity and the likelihood can be used for model comparison and selection. Such models include Restricted Boltzmann Machines (RBMs) (Hinton 2012), Deep Boltzmann Machines (DBMs) (Salakhutdinov & Hinton 2009) and Variational Autoencoders (VAEs) (Kingma & Welling 2019). GANs (Goodfellow et al. 2014) belong in the implicit model classification and provide a sampling mechanism for generating data, but do not require to explicitly define a likelihood function, which in many cases, such as photo-realistic image synthesis is relatively hard to find. This section focuses on the advancements of GANs, mainly due to their profound contribution in the field of CV and the broad range of applications in which they can be used.

The GAN framework was originally proposed in the work of (Goodfellow et al. 2014). That work defined a process of training simultaneously two opponent networks; the generator $G$ and the discriminator $D$. Both networks can be expressed as differentiable functions, so that the generator receives an input from a simple probability distribution $z$, such as random noise, and outputs a sample $x$, $G(z;\theta_G)$, and the discriminator receives the output of the generator $D(x;\theta_D)$ and predicts if the input came from the probability distribution of the generator (fake) or from the true data distribution (real). The generator and discriminator, with $\theta_G$ and $\theta_D$ representing the free parameters of the networks respectively, are usually deep neural networks, such as CNNs, which are able to cope with the complex nature of the true probability distributions. The GAN framework is illustrated in **(Figure 2.46)**.

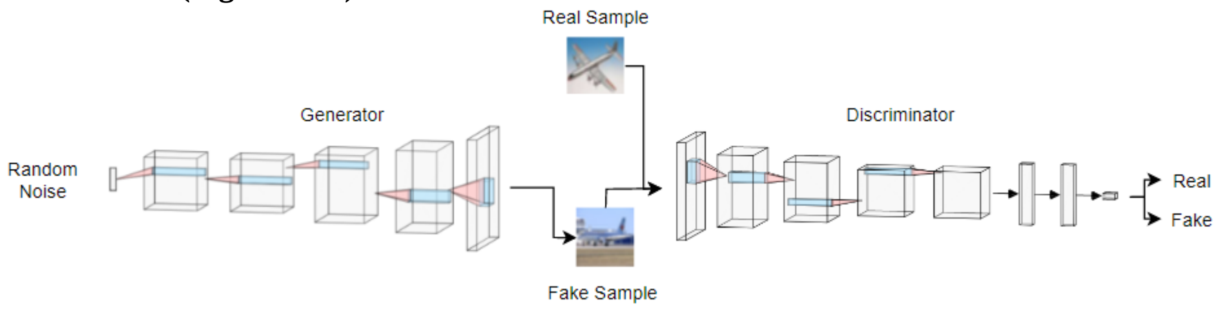


**Figure 2.46** A visual illustration of the GAN framework (Goodfellow et al. 2014).

The GAN framework (Goodfellow et al. 2014) was introduced with two loss functions; the first is known as the minimax GAN loss and the second is known as non-saturating GAN loss. In both cases the discriminator loss function is the same as it seeks to minimize the probability assigned to real and fake samples. Mathematically the discriminator loss aims to maximize the average of the log probability for real samples and the average log of the inverted probabilities of fake samples Eq. (2.75). As this is a maximization problem, if Eq. (2.75) is applied directly, it would require stochastic gradient ascent weight change instead of the conventional stochastic gradient descent.

$$C = log(D(x)) + log\left(1 - D(G(z))\right) \tag{2.75}$$

For this reason, Eq. (2.75) is usually implemented as a binary classification problem with class labels 0 and 1 for fake and real samples, respectively. Based on that, the model fitting process aims to minimize the average binary cross entropy, which is also known as log loss:

$$C = y \cdot -log(y') + log(1 - y) \cdot -\log(1 - y') \tag{2.76}$$

where $y$ corresponds to the expected output and $y'$ the prediction.

The minimax GAN loss refers to the simultaneous minimax optimization of the generator and the discriminator models. Minimax is an optimization strategy in a two-player turn-based game for minimizing the loss for the worst case of the player. In the case of GANs the discriminator and the generator are the players of the game in which upon each turn their weights of their model are getting updated. The min and max from the minimax loss refer to the minimization and the maximization of the generator and discriminator loss, respectively. This is expressed as the maximization of Eq. (2.75) and minimization of (2.77):

$$C_G = \log\Big(1 - D\big(G(z)\big)\Big) \tag{2.77}$$

Although the use of minimax methodology served in the mathematical analysis of the GAN framework, in practice this loss does not provide sufficient gradient for the generator to learn well. The saturating gradient becomes apparent especially in the early stages of learning, as the discriminator can easily identify the forged generated samples of the untrained generator.

To overcome the gradient saturation of minimax, the non-saturating GAN loss was proposed. The generator loss is modified so that it maximizes the log discriminator probabilities for the generated samples instead of minimizing the inverted probabilities:

$$C_G = \log\Big(D\big(G(z)\big)\Big) \tag{2.78}$$

This also changes the framing of the problem. In fact, using the minimax loss the generator tries to minimize the probability of the samples predicted as fake, while in the non-saturating loss, the generator tries to maximize the probabilities of samples predicted as real. (Goodfellow et al. 2014) found that this loss prevents gradient saturation and in general promotes stable learning.  Similarly, to the discriminator the implementation of the non-saturating loss is done by expressing the problem as binary classification. This is done by flipping the labels of real and fake samples and minimizing the cross-entropy.

Many loss functions have been proposed over the years (Pan et al. 2020). Two widely used losses are the Least squares GAN loss (Mao et al. 2017) and the Wasserstein GAN loss (Arjovsky et al. 2017). Least squares GAN loss was proposed aiming to solve the limitations of using binary cross-entropy loss when generated images. The authors observed that when binary cross-entropy is used, the generated images are relatively different that the real, which can lead to small or vanishing gradients, resulting from little to no weight updates for the model. The least square GAN loss aims for the discriminator to minimize the sum squared difference between the predicted and the real values (2.86) and for the generator to minimize the sum squared difference between the predicted and expected values as though the predicted images were real Eq. (2.80). The main benefit of this loss is that it penalizes more the large errors, which results in larger corrections and thus, battles the problem of vanishing gradient.

$$C_D = (D(x) - 1)^2 + D\big(G(z)\big)^2 \tag{2.79}$$

$$C_G = \Big(D\big(G(z)\big) - 1\Big)^2 \tag{2.80}$$

The Wasserstein GAN loss (Arjovsky et al. 2017), was motivated by the fact that conventional GANs aim to minimize the distance between the predicted and actual probability distributions of real and the generated samples. The authors proposed an alternative methodology in which the the problem instead of following the conventional Kullback-Leibler divergence (KLD) modeling, follows the Earth-Mover's distance (EMD), which is also called Wasserstein-1 distance. EMD calculates the distance between two probabilities in terms of the cost of turning one distribution into another. Wasserstein GAN loss changes the notion of discriminator as a critic, that updates more frequently than the generator. The critic receives as an input a sample, which instead of predicting a probability of been real or fake, predicts a real value which can be considered a score. The score is calculated so that the distance between real and generated samples are maximally separated. This is achieved by calculating the average predicted score across the real and generated samples and multiplying the average score by 1 and -1, respectively. The main benefit of Wasserstein GAN loss is that it produces strong gradient in almost every case, which enables the continuation of training. Another benefit of this loss is that it directly correlates with the generator generating power as the lower the loss, the better the generated samples. The result of this is that the methodology, gives the loss of the GAN convergence properties as it explicitly seeks for the minimization of the generator loss.

Training a GAN is a complicated task as it involves the training of two different networks. In general, training the discriminator involves the optimization of two loss functions: the generator loss and the discriminator loss. The first is ignored upon the discrimination phase while the second is used when training the generator.

During the discriminator training, the network classifies real samples coming from the training dataset and fake samples coming from the generator. The discriminator loss penalizes the discriminator for errors involving sample misclassifications of been real while is fake and visa-versa. The error backpropagation is typically used to update the weights of the discriminator. In the generator training phase, the network is trained through the discriminator network. To perform, input is provided through the generator, whose output is connected directly to the discriminator network and using the output of the discriminator the generator loss, penalizes the generator for producing samples that are classified correctly by the discriminator. The error backpropagation passes the error backward, through the entire network stack, including the discriminator, yet on the update phase, only the weights of the generator are updated.

In the training process of a GAN, the generator and discriminator training are performed in two alternating steps. The first involves the training of the discriminator, for one or more epochs and the second trains the generator for one or more epochs depending on the GAN architecture. On each turn of this process, the weights of the network that waits its turn, do not update, which is also called weight freezing. This give a “fair chance” of the network that is being trained to recognize the “flows” of the “opponent” network. Accessing the convergence of a GAN is a relatively hard. This is because as the generator training progress, the discriminator becomes more and more inaccurate on trying to access the validity of the input data. A perfectly trained generator should degrade the discriminator accuracy to 50% which would be equal of flipping a coin to guess if a sample is real or fake. The disadvantage of this is that as the training progress, the discriminator feedback is becoming less meaningful, which if when continues at some point the discriminator starts providing random feedback. This random feedback can lead to generator performance degradation which is also known as “generator collapse”.

A common problem other than the vanishing gradient, in which GANs may run into while training is called “mode collapse”. This problem arises when a generator, instead of producing a variety of samples, it learns to generate a single sample, that “tricks” the discriminator. The result of this is that if the generator starts to produce the same sample repeatedly, that can lead the discriminator training to optimize for this edge case.  If the discriminator gets stuck into a local minimum and it does not find the best strategy to reject the generated sample, it is easy for the generator in the next training iteration to find another output that tricks the discriminator. By repeating this process several times, the generator is overoptimized for the discriminator of that training iteration. To mitigate this problem, Wasserstein GAN loss (Arjovsky et al. 2017) can be used, which enables the discriminator to train optimally without the problem of vanishing gradient. Unrolled GANs (Metz et al. 2016) are also commonly used because of their generator loss function as it incorporates the possible outputs of future discriminators, $k$ steps in the future, along with the current discriminator output, leading to a generator that does not overfit to a single discriminator. Another common problem in GAN training, is that it is relatively common to simple not converge. This can be due to the limited amount of training samples or poorly constructed training strategy, such as not allowing the discriminator to train on enough iterations. Regularization techniques are commonly employed in an effort to mitigate this problem. Such methods include to introduction of noise in the input of the discriminator (Arjovsky & Bottou 2017), or penalizing the weights (Roth et al. 2017) of the  discriminator.

## 2.4.4 Advancements in Generative Adversarial Networks

Many variations of the original GAN framework (Goodfellow et al. 2014),  have been proposed over the years (Gonog & Zhou 2019). The variations can be classified into two different taxonomies: loss variations and architectural variations. In the first taxonomy, the main focus of the change is on the loss function of the generator and/or the discriminator while on the second the focus is shifted on the neural network(s) used for the generator and the discriminator. A summary of these architectures along with their taxonomy (Wang et al. 2019) is included in **Table 2.2**.

**Table 2.2** Summary of recent GAN models.

| Taxonomy | Year | Name | Summary |
|---|---|---|---|
| Architecture – Latent space | 2014 | Conditional GAN (CGAN) (Mirza & Osindero 2014) | Info of labels into the discrimination and the generator |
| Architecture – Latent space | 2016 | Bidirectional GAN (BiGAN) (Donahue et al. 2016) | Learning inverse mapping using an encoder |
| Architecture – Latent space | 2016 | SGAN (Odena 2016) | Multi-headed discriminator |
| Architecture – Latent space | 2016 | InfoGAN (Chen et al. 2016) | Label classification |
| Architecture – Latent space | 2017 | Auxiliary Classifier GAN (AC-GAN) (Odena et al. 2017) | Auxiliary classifier |
| Architecture – Application specific | 2016 | SRGAN (Ledig et al. 2017) | Image super-resolution |
| Architecture – Application specific | 2017 | CycleGAN (Zhu et al. 2017) | Image style-transfer |
| Architecture – Application specific | 2017 | DiscoGAN (Kim et al. 2017) | Image style-transfer |
| Architecture – Application specific | 2017 | DualGAN (Yi et al. 2017) | Image style-transfer |
| Architecture – Application specific | 2017 | Pix2Pix (Isola et al. 2017) | Image style-transfer |
| Architecture – Application specific | 2017 | Face Completion GAN (Li et al. 2017) | Face completion |
| Architecture – Application specific | 2018 | AlphaGAN (Lutz et al. 2018) | Image matting |
| Architecture – Application specific | 2018 | Moco-GAN (Tulyakov et al. 2018) | Video generation |
| Architecture – Application specific | 2019 | DVD-GAN (Clark et al. 2019) | Video generation |
| Architecture – Application specific | 2019 | SinGAN (Shaham et al. 2019) | Manipulations of image when trained on single image |
| Architecture – Application specific | 2019 | StyleGAN (Karras et al. 2019) | Face generation |
| Architecture – Application specific | 2020 | StyleGAN2 (Karras et al. 2020) | Face generation |
| Architecture – Network change | 2015 | LAPGAN (Denton et al. 2015) | Laplacian pyramid coding |
| Architecture – Network change | 2016 | DCGAN (Radford et al. 2015) | Generator with transposed convolutions |
| Architecture – Network change | 2017 | BEGAN (Berthelot et al. 2017) | Discriminator using an autoencoder |

| Taxonomy | Year | Name | Summary |
|---|---|---|---|
| Architecture – Network change | 2017 | Progressive GAN (PROGAN) (Karras et al. 2017) | Progressive training |
| Architecture – Network change | 2018 | Self-attention GAN (SAGAN) (H. Zhang et al. 2019) | Usage of a self-attention module |
| Architecture – Network change | 2018 | Non-stationary texture synthesis (NSTS-GAN) (Zhou et al. 2018) | Texture Synthesis from image patches |
| Architecture – Network change | 2019 | AutoGAN (Gong et al. 2019) | Automatic searching for multi-level architectures |
| Architecture – Network change | 2019 | BigGAN (Brock et al. 2018) | Self-attention module with deeper networks and larger batch size |
| Architecture – Network change | 2020 | Your Local GAN (YLG) (Daras et al. 2020) | Local sparse attention layer |
| Loss function – Type | 2014 | Fully Connected GAN (FCGAN) (Goodfellow et al. 2014) | Jensen–Shannon divergence |
| Loss function – Type | 2016 | Least Square GAN (LS-GAN) (Mao et al. 2017) | Pearson divergence |
| Loss function – Type | 2016 | f-GAN (Nowozin et al. 2016) | f-divergence |
| Loss function – Type | 2016 | Unrolled GAN (UGAN) (Metz et al. 2016) | Gradient loss with second order derivatives |
| Loss function – Type | 2017 | LS-GAN (Mao et al. 2017) | Designated margin difference between real and generated samples |
| Loss function – Type | 2017 | Geometric GAN (Lim & Ye 2017) | Hinge loss |
| Loss function – Type | 2017 | WGAN (Arjovsky et al. 2017) | Wasserstein distance |
| Loss function – Type | 2018 | Relativistic GAN (RGAN) (Jolicoeur-Martineau 2018) | Integral probability metric |
| Loss function – Type | 2019 | Sphere GAN (Park & Kwon 2019) | Riemannian manifolds |
| Loss function – Regularization | 2016 | Mode Regularized (MRGAN) (Che et al. 2016) | Missing mode penalization |

| Taxonomy | Year | Name | Summary |
|---|---|---|---|
| Loss function – Regularization | 2017 | WGAN-GP (Gulrajani et al. 2017) | WGAN based with gradient penalty |
| Loss function – Regularization | 2017 | WGAN-LP (Petzka et al. 2017) | WGAN based with Lipschitz penalty |
| Loss function – Regularization | 2018 | Spectral normalization GAN (SN-GAN) (Miyato et al. 2018) | Spectral Normalization |
| Loss function – Regularization | 2019 | Self-supervised GAN (SS-GAN) (T. Chen et al. 2019) | Self-supervision to avoid discrimination forgetting previous generations |

A detailed review of all the methodologies included in **Table 2.2**, is out of the scope of this thesis, yet a selection of those is worth examining due to their contribution in the field of GANs.

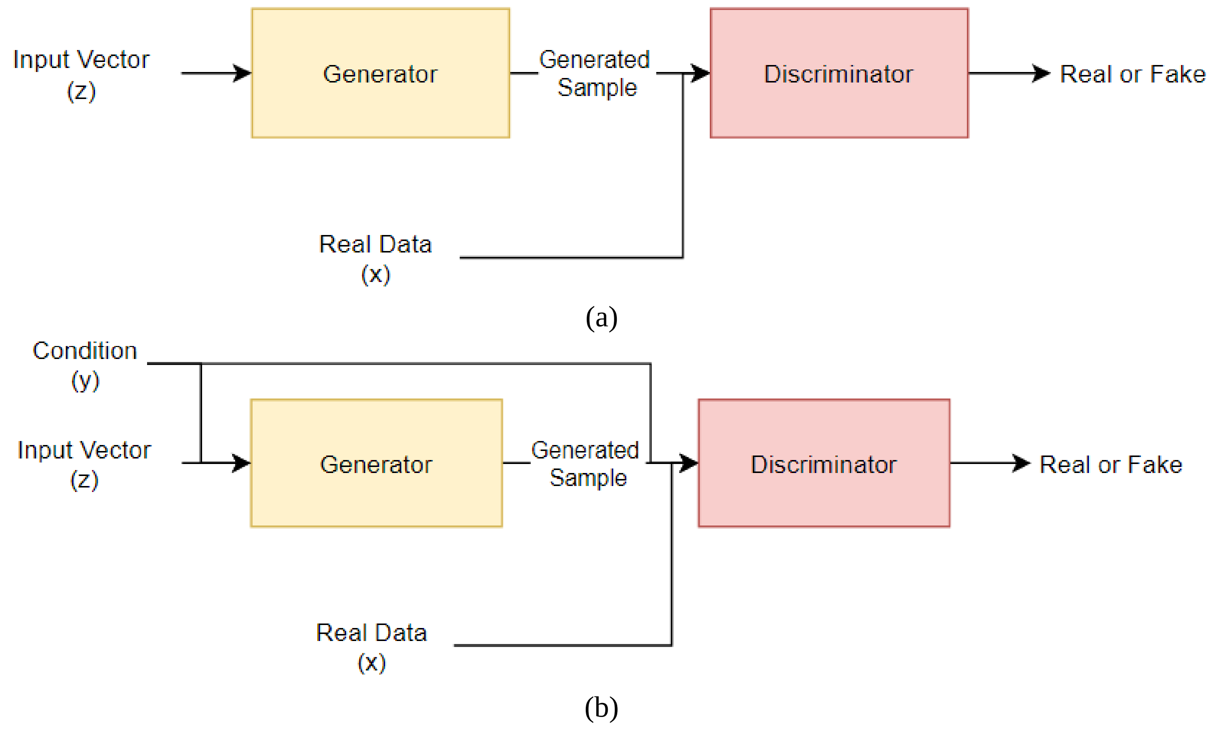


**Figure 2.47** A visual comparison between (a) the original GAN architecture (Goodfellow et al. 2014) and (b) the CGAN architecture (Mirza & Osindero 2014).

### 2.4.4.1 The Conditional GAN

The conventional GAN framework (Goodfellow et al. 2014), was originally designed to take as an input a random noise vector a produce an image. Although this is effective, the random noise input allows no control over the type of the generated image. Conditional GANs (CGANs) (Mirza & Osindero 2014) aim to improve that by incorporating class label information in the GAN model.

The introduction of additional information as an input not only enables targeted output, it also improves the GAN performance, as it promotes more stable training and quicker convergence. CGAN authors validated conditional GANs on the MNIST (Deng 2012) dataset, from which the class labels were used as additional information when training the generator and the discriminator of the model. An illustration of the CGAN architecture is illustrated in **(Figure 2.47b)**.

### 2.4.4.2 The InfoGAN

InfoGAN (Chen et al. 2016) is an extension of the CGAN architecture (Mirza & Osindero 2014) and aims to incorporate interpretable representations in an unsupervised manner by maximizing the mutual information between the conditional variables, *i.e.* class labels, and the generated data. To achieve that, InfoGAN, incorporates a classifier $(Q)$ which predicts the conditional variable $(y)$ given by the generator $G(z|y)$. This closely resembles an autoencoder which aims to encode the $G(z|y)$ by minimizing the cross entropy between the actual conditional variable $y$ and the predicted variable $y'$. Another change compared to CGAN is that the discriminator of InfoGAN does not take as an input the conditional variable and instead operates the same as the original discriminator of (Goodfellow et al. 2014).

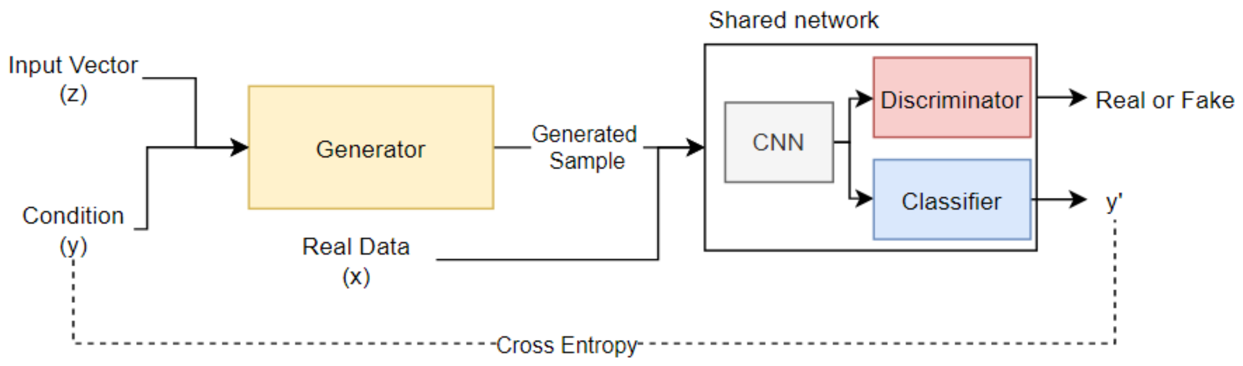


**Figure 2.48** An illustration of InfoGAN architecture (Chen et al. 2016). The shared network block illustrates a single network with two output layers; one for discrimination and one for the classification.

To reduce the computational resources required by the introduction of the classifier, both the discriminator and the classifier use the same network with different two output layers. The first output layer is considered the discriminator and predicts if a sample is real or fake and the second the classifier for the conditional variable. The architecture is illustrated in **(Figure 2.48)**. A relatively similar architecture is AC-GAN (Odena et al. 2017), which share the same principles with InfoGAN, yet the discriminator and the classifier are combined using an auxiliary classifier. It should be noted that while in CGAN and InfoGAN case, the conditional variable can be from different domains, the AC-GAN architecture limits the variable to be a class label from the same domain.

### 2.4.4.3 The Deep Convolutional GAN

Deep Convolutional GAN (Radford et al. 2015), also known as DCGAN, is the first GAN architecture that incorporates deconvolutional layers in the generator architecture. The use of deconvolutional layers in CNNs was firstly introduced in ZFNet (Zeiler & Fergus 2014) for feature visualization. DCGAN makes use of the spatial up-sampling ability of the deconvolution layer, to generate images with similar statistics with the real image, but in higher resolution. Compared to the original GAN architecture DCGAN introduced changes that benefit the high-resolution generation and stabilize training. More specifically, pooling layers were replaced by convolution layers with a stride of 2 for the discriminator and fractional-strides for the generator. Batch normalization was used in both generator and discriminator. For the generator ReLU activations were used across all layers of the network with the exception of the output layer in which tanh was used. To prevent the discriminator to saturate, LeakyReLU activations were used across the discriminator layers. LeakyReLU is a variation of the conventional ReLU activation which allows for a relatively small negative number to "leak" from the activation.

### 2.4.4.4 Image to Image Translation GANs

The object in an image-to-image translation problem is to learn the mapping between an output and an input image using a training set of aligned image pairs. An example is illustrated visually in the **(Figure 2.49)**. A variety of GAN models have been proposed for this process, with the first

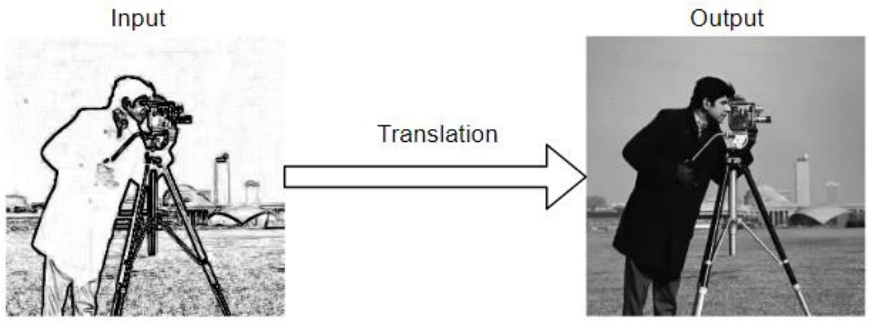
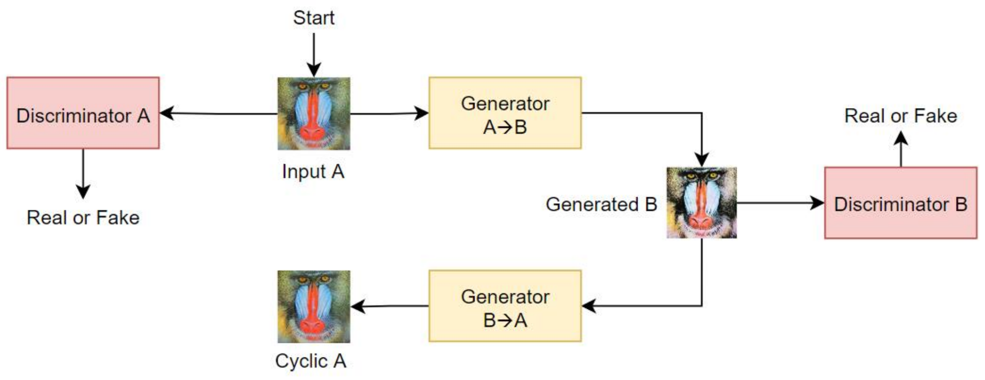


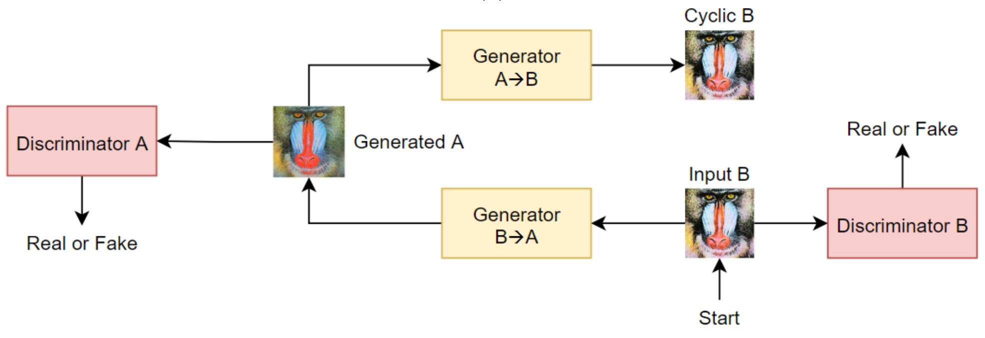
**Figure 2.49** An example of image-to-image translation in which a translation model is used to map/translate edges to a real image.

been the Pix2Pix (Isola et al. 2017) architecture which re-purposed the CGAN (Mirza & Osindero 2014) architecture to translate images when paired input and output samples are available. Such samples include image edges used as an input and output the actual image. As an example, Pix2Pix used canny filters to create the training image pairs.

While this approach works relatively well, Pix2Pix was not able to translate images from unmapped pairs. Aiming towards this direction, CycleGAN (Zhu et al. 2017) was introduced which uses a cycle consistency loss to enforce the mapping between the two domains. In effect this is achieved by transforming the source distribution to target and then back to the source distribution leading to get samples from the source distribution **(Figure 2.50)**. As an input, CycleGAN takes an image from a domain $Domain_A$ which is then given to a generator $G_{A\rightarrow B}$. The role of $G_{A\rightarrow B}$ is to translate the image from the $Domain_A$ to the target $Domain_B$. The generated image is then given to a second generator $G_{B\rightarrow A}$ which converts it back to the original $Domain_A$. The output of this process is an image that is close to the original input space, which in return creates a meaningful mapping between the two pairs. Two inputs are used in both discriminators, one been the original image and the other the generated sample. The goal of the discriminator is to distinguish them, so that the discriminator defies the adversary and reject the generated samples. The result of this, while training, is that the generator tries to trick the discriminator by creating samples as close to the original image $Domain_B$.



(a)



(b)

**Figure 2.50** A visual illustration of CycleGAN (Zhu et al. 2017) image translation framework.

DiscoGAN (Kim et al. 2017), is a relatively similar architecture with CycleGAN, as they both aim to learn two transformation functions, from a $Domain_A \rightarrow Domain_B$ and $Domain_B \rightarrow Domain_A$. Furthermore, both follow the same principle of transforming an image from one domain to another and then back to the original domain the output should match the original image. The primary difference between the two models is that in the case of DiscoGAN **(Figure 2.51)**, two reconstruction losses are used, one for each domain, while CycleGAN uses a single cycle-consistency loss.

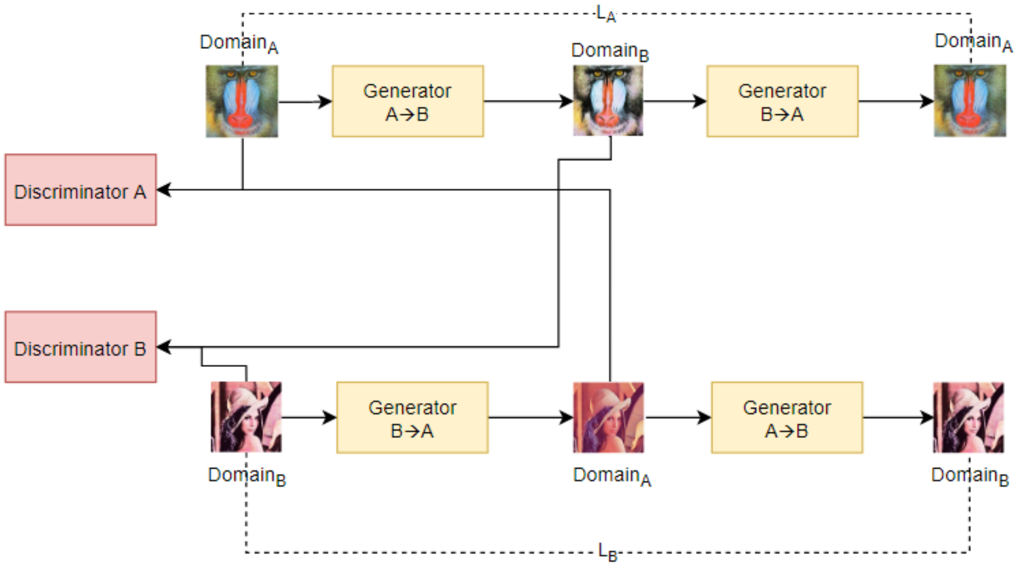


**Figure 2.51** A visual illustration of DiscoGAN (Kim et al. 2017) image translation framework.

# CHAPTER 3

# FUZZY POOLING

Convolutional neural networks are artificial learning systems typically based on two operations: convolution, which implements feature extraction through filtering, and pooling, which implements dimensionality reduction. The impact of pooling in the classification performance of the CNNs has been highlighted in several previous works, and a variety of alternative pooling operators have been proposed. However, only a few of them tackle with the uncertainty that is naturally propagated from the input layer to the feature maps of the hidden layers through convolutions. In chapter a novel pooling operation based on (type-1) fuzzy sets is presented that aims to cope with the local imprecision of the feature maps. An investigation of its performance in the context of image classification is also included. Fuzzy pooling is performed by fuzzification, aggregation and defuzzification of feature map neighborhoods. It is used for the construction of a fuzzy pooling layer that can be applied as a drop-in replacement of the current, crisp, pooling layers of CNN architectures. Several experiments using publicly available datasets show that the proposed approach can enhance the classification performance of a CNN. A comparative evaluation shows that it outperforms state-of-the-art pooling approaches.

## 3.1 Introduction

CNNs (Krizhevsky et al. 2012; LeCun et al. 1989) have revolutionized CV and image analysis. At the core of every CNN architecture there is a special type of neural layer called convolutional layer. This bioinspired layer has a neuron arrangement that mimics the connections of the visual cortex. The number of connections of each neuron in a convolutional layer is called receptive field. This is a key element of the layer as it determines the size of the filter applied throughout the input volume of the layer. The weights between the same type of filters in a convolutional layer are shared, forming a feature map. A single convolutional layer can produce multiple feature maps. One or more convolutional layers are usually connected to each other through a pooling layer for spatial dimensionality reduction (Simonyan & Zisserman 2014).

The progress in CNN-based ML research is rapidly evolving (Jiao & Zhao 2019). Advances include novel architectures (Diamantis et al. 2019), methodologies for training (Ioffe & Szegedy 2015), activation functions (He et al. 2015a; Maas et al. 2013), and convolutional layer optimizations (Chollet 2017), whereas reported classification performance enhancements are usually associated with increase in the complexity of the networks (Szegedy et al. 2017; He et al. 2016). The role of pooling in classification performance has been highlighted in previous studies (Malinowski & Fritz 2013; Zeiler & Fergus 2013; Yu et al. 2014; Graham 2014; Yildirim & Baloglu 2019) however, only a few approaches have been reported tackling the problem of uncertainty (Sharma et al. 2019).

A pooling layer, typically, performs a down-sampling operation to reduce the spatial size of an input volume using a sliding window of $k \times k$ features per feature map, with a stride $\sigma$. Pooling is performed on each window, reducing the size of the respective patches from $k \times k$, to a single feature. This results in a reduction of the number of free-parameters of the CNN, and thus in a reduction of the overall network complexity. Contemporary CNN architectures use the maximum (max-pooling) or the average pooling operations (Boureau et al. 2010), mainly due to their implementation simplicity. In the case of max-pooling, the majority of features are discarded, favoring only the highest neuron responses, whereas in the case of average pooling the features are mixed. As a result, the information represented by the features of the respective feature maps is distorted and possible uncertainties, *e.g.*, due to input noise, are propagated to subsequent layers and dispersed throughout the network.

In this chapter we propose a novel *fuzzy pooling* operation used for the construction of a fuzzy CNN pooling layer, tackling with uncertainties in the feature values. This is achieved by transforming the crisp input volume space into a fuzzy feature space, generated by the memberships of the original feature values, to fuzzy sets, facilitating linguistic representation of different value intervals. Fuzzy pooling is implemented by fuzzy aggregation and defuzzification of the fuzzified input features. This is performed aiming to a better preservation of the information of the original feature maps.

## 3.2 Related work

Fuzzy set theory has been proved effective in modeling uncertainty in the context of robust image processing and analysis applications. Such uncertainties may originate from various sources, including greylevel ambiguity, vagueness of image features, noise introduced by the image sensor (Chacón M 2006). Relevant recent applications include image segmentation based on multiple-kernel fuzzy *c*-means clustering (Chen et al. 2011), and segmentation by thresholding, based on type-2 fuzzy sets (Yüksel & Borlu 2009). In the context of pedestrian segmentation in infrared images, symmetry information based fuzzy clustering has been exploited. In (Lee et al. 2005) a genetic-based fuzzy image filter was proposed to remove additive identical independent distribution impulse noise from highly corrupted images with very promising results. In (Kumar et al. 2015) an alternative to conventional histogram-based image descriptors is presented, where fuzzy membership functions are used. In the same study a novel methodology is presented, named “gamma mixture fuzzy model”, which enables the detection of geometrically consistent correspondence between two images.

In the field of ML, and more specifically of ANNs, fuzzy set theory has been employed to model data uncertainties. In (Juang & Ku 2005) a recurrent fuzzy network was presented. The network was capable of performing temporal sequence recognition and it was applied in the challenging problem of gesture recognition. In the subject of dynamic-system modeling,  a locally recurrent fuzzy neural network with support vector regression was proposed (Juang & Hsieh 2010). More specifically a five-layered network was considered in which the recurrent capabilities come from locally feeding the activations of each fuzzy rule back to itself. Recently in (Kebria et al. 2019) an adaptive interval type-2 fuzzy neural network control scheme was proposed in the context of teleoperation systems with time-varying delays and uncertainties.

Pooling alternatives for CNNs include trainable pooling approaches such as (Malinowski & Fritz 2013), which jointly optimizes both the classifier and the pooling regions, instead of relying on fixed, spatial pooling regions. A CNN-specific stochastic pooling operation was proposed in (Zeiler & Fergus 2013), aiming to be used as an effective regularization method in deep networks in combination with other regularization methods such as data augmentation and dropout (Srivastava et al. 2014) layers. The method relies on a non-deterministic approach of randomly picking an activation from the pooling region according to a multinomial distribution, which is given by the activities in the pooling region. Similarly another pooling operation named mixed pooling was proposed in (Yu et al. 2014) which also aims to be used as a regularization method in CNN training. The approach randomly selects between average and max pooling operations in a non-deterministic way, following similar principles applied in typical dropout layer. Another promising stochastic pooling operation, named Fractional Max-Pooling, was proposed in (Graham 2014), as a variation of max-pooling in which the pooling regions can output more than one values at a time. A recent pooling algorithm, called RegP (Yildirim & Baloglu 2019), follows a different, deterministic approach. The algorithm analyzes each activation value in the pooling region by examining the values of the surrounding activations and computes a score that represents the number of same or similar values around them. The activation with the maximum similarity value is selected as the output. In ambiguous cases, were multiple activations have the same score, the average value is selected as the output of the pooling region.

Only a few works have considered fuzzy set theory with respect to pooling. Recently in (Sharma et al. 2019) a type-2 fuzzy based pooling was proposed, as a solution to the value selection uncertainty that is present in conventional pooling operations, such as max and average pooling. The methodology reduces the spatial input size in two steps. Initially the dominant values of the pooling window are identified using type-2 fuzzy logic. Spatial size reduction is performed using type-1 fuzzy logic with weighted average of the dominant values found in the first step. To identify the importance of each value, the values are compared to a threshold computed using the average of the minimum and maximum values of Gaussian membership functions applied on the input space. The algorithm requires a minimum set of important values to be present in order to apply the fuzzy pooling operation; if this criterion is not met, the algorithm falls back to the conventional crisp pooling. In that sense, that approach can be considered as hybrid. The authors evaluated their type-2 fuzzy based pooling on the standard LeNet (LeCun et al. 1989) CNN architecture, using two standard datasets showing promising results compared to max and average pooling.

In this chapter we propose an effective fuzzy pooling methodology that can be used as a drop-in replacement of the pooling layer used in any of the current CNN architectures. The proposed methodology considers the CNN feature maps as locally imprecise, due to the uncertainty propagated from the previous layers, and it uses fuzzy sets as a means to model this imprecision and implement pooling through fuzzy aggregation and defuzzification. The importance of each value in a pooling window is characterized by a score obtained from its membership value in type-1 fuzzy sets which are determined by the activation function used by the previous layer. Comparing to (Sharma et al. 2019), no value-specific thresholds are used, which enables the proposed fuzzy pooling methodology to be applied uniformly on all the pooling areas of the input space; thus not leaving any uncertainties to be propagated to the following layers of the network.

## 3.3 Type-1 Fuzzy Pooling

Fuzzy pooling is defined in the context of a CNN architecture. It constitutes the basis of a novel pooling layer for uncertainty-aware dimensionality reduction. Considering that the pixel values of the input images of a CNN are prone to uncertainty (*e.g.*, noise, color, and geometrical ambiguity), and that the information is forwardly propagated from the input to the subsequent hidden layers, the uncertainty, which is part of this information, is also propagated through the different network layers; thus, affecting the values of their feature maps. Convolution is a local operation; therefore, the uncertainty is expected to be also local in the output space of a convolutional layer.

Given a feature map extracted from a convolutional layer, the uncertainty in its values can be modeled by fuzzy sets:

$$\tilde{y}_v = \{\langle x, \mu_v(x)\rangle \mid x \in E\}, v = 1, \dots, V \tag{3.1}$$

representing overlapping value intervals that can be linguistically expressed, *e.g.*, as small, medium and large values. The universe *E* is selected upon the output value ranges representing overlapping value intervals that can be linguistically expressed, *e.g.*, as small, medium and large values. The universe *E* is selected upon the output value ranges of the neural activation functions of the convolutional layer. To illustrate this, the Rectified Linear Unit (ReLU) (Nair & Hinton 2010) activation function is considered as a representative example. ReLU is defined as:

$$ReLU(x) = \max(0, x) \tag{3.2}$$

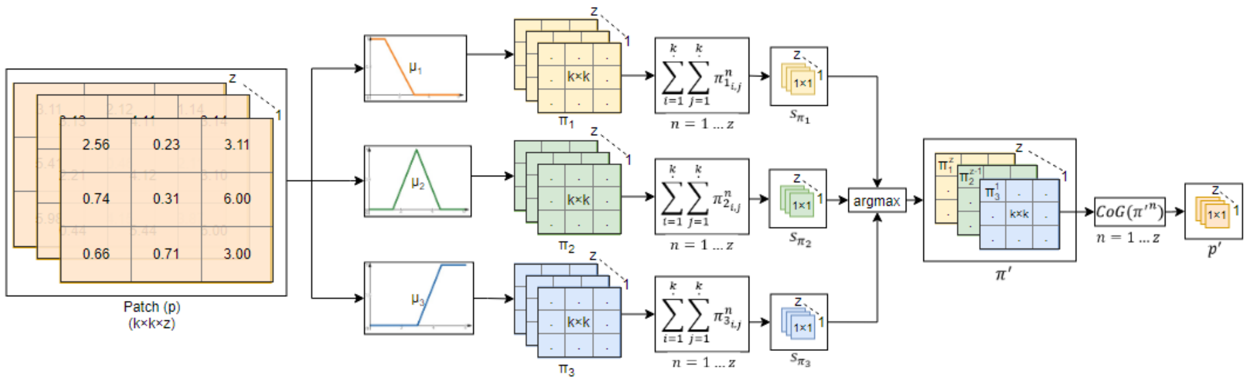


**Figure 3.1** Schematic representation of the proposed fuzzy pooling operation applied on a single volume patch extracted from a set of $z$ feature maps.

Where $x \in \mathbb{R}$, and it is usually preferred to conventional sigmoid activations, because it is computationally simpler, and it reduces the possibility of vanishing gradients from which, deep neural networks suffer (Hochreiter 1998). Also, empirical studies have shown that a network with ReLU activation functions tend to converge faster than sigmoid (Krizhevsky et al. 2012). Recent studies (Howard et al. 2017) suggest that ReLU is capped by a maximum value $r_{max}$:

$$ReLU(x, r_{max}) = \min(\max(x, 0), r_{max}) \tag{3.3}$$

where typically $r_{max} = 6$, as it has been shown that it helps the network learn sparse features earlier. Therefore, in this case $E = [0, r_{max}]$.

Let $\beta$ stand for a volume $w \times h \times z$, representing a set of $z$ feature maps $\beta^n$ with a size of $w \times h$, *i.e.*, $\beta = \{\beta^n \mid n = 1,2, \dots, z\}$. The first step of the proposed methodology is sampling the input volume with a pooling window of size $k \times k$. Commonly used values for these hyperparameters include $k = 3$ and $\sigma = 2$, which result in a reduction of the width and the height of the input volume in half. With this process a set of volume patches is obtained from the input volume $\beta$ with stride $\sigma$. Each volume patch, consists of spatial patches $p^n$ extracted from feature maps $\beta^n$, *i.e.*, $p = \{p^n \mid n = 1,2, \dots, z\}$. The number of patches $c$ that can be extracted from an input volume $\beta$ can be calculated by:

$$c = \frac{(w - k + 2t_w)(h - k + 2t_h)}{2\sigma + 2} \tag{3.4}$$

where $t_w = \frac{(\sigma-1)(w-\sigma+k)}{2}$ and $t_h = \frac{(\sigma-1)(h-\sigma+k)}{2}$ is the zero-padding used in the patch extraction process on the width and height axis of the input volume $x$ respectively.

Let $p_{i,j}^{n}$ stand for an element of a volume patch $p$ at depth $n$ and position $i, j$ where $i = 1, \dots, k$, $j = 1, \dots, k$ and $n = 1, \dots, z$. Without loss of generality, let us consider a set of three fuzzy sets defined by (1) for *V*=3. These $\tilde{y}_1, \tilde{y}_2$ and $\tilde{y}_3$ fuzzy sets with membership functions $\mu_1$, $\mu_2$ and $\mu_3$, are used to represent small, medium and large values of $p_{i,j}^{n}$, respectively. The membership functions of these sets are used for the fuzzification of the patches. For example, in the case of triangular membership functions $\mu_i, i = 1{,}2{,}3$, using (1), this can be expressed as follows:

$$\mu_1\left(p_{i,j}^{n}\right) = \begin{cases} 0 & p_{i,j}^{n} > d \\ \dfrac{d - p_{i,j}^{n}}{d - c} & c \le p_{i,j}^{n} \le d \\ 1 & p_{i,j}^{n} < c \end{cases} \tag{3.5}$$

where $d = \frac{r_{max}}{2}$ and $c = \frac{d}{3}$,

$$\mu_2\left(p_{i,j}^{n}\right) = \begin{cases} 0 & p_{i,j}^{n} \le a \\ \dfrac{p_{i,j}^{n} - a}{m - a} & a \le p_{i,j}^{n} \le m \\ \dfrac{b - p_{i,j}^{n}}{b - m} & m < p_{i,j}^{n} < b \\ 0 & p_{i,j}^{n} \ge b \end{cases} \tag{3.6}$$

where $a = \frac{r_{max}}{4}$ , $m = \frac{r_{max}}{2}$ and $b = m + a$,

$$\mu_3\left(p_{i,j}^{n}\right) = \begin{cases} 0 & p_{i,j}^{n} < r \\ \dfrac{p_{i,j}^{n} - r}{q - r} & r \le p_{i,j}^{n} \le q \\ 1 & p_{i,j}^{n} > q \end{cases} \tag{3.7}$$

where $r = \frac{r_{max}}{2}$ and $q = r + \frac{r_{max}}{4}$.

For each patch $p^n, n = 1, \dots, z$, a fuzzy patch $\pi_v^n$ is defined as

$$\pi_v^n = \mu_v(p^n) = \begin{pmatrix} \mu_v\left(p_{1,1}^{n}\right) & \dots & \mu_v\left(p_{1,k}^{n}\right) \\ \dots & \ddots & \dots \\ \mu_v\left(p_{1,k}^{n}\right) & \dots & \mu_v\left(p_{k,k}^{n}\right) \end{pmatrix} \tag{3.8}$$

Pooling begins with the spatial aggregation of the values of the fuzzy patch, using the fuzzy algebraic sum operator ($\dot{\Sigma}$) (Zimmermann 2011), as follows:

$$s_{\pi_v}^n = \sum_{i=1}^{k}\sum_{j=1}^{k} \pi_{v_{i,j}}^n, \qquad n = 1, \dots, z. \tag{3.9}$$

This operator was selected as a standard $s$-norm considering all the neighboring values of the fuzzy patch. It has a relatively low computational complexity, and it can be easily vectorized to be efficiently performed on a GPU. The value of each $s_{\pi_v}^n$ is considered as a score quantifying the overall membership of $p^n$ to $\tilde{y}_v$. Based on these scores, for each volume patch $p$ a new fuzzy volume patch $\pi'$ is created by selecting the spatial fuzzy patches $\pi_v^n, v = 1, \dots, V$ that have the largest scores $s_{\pi_v}^n$, *i.e.*,

$$\pi' = \left\{\pi'^n_v = \pi_v^n \mid\ v = \text{argmax}\left(s_{\pi_v}^n\right), n = 1,2, \dots, z\right\} \tag{3.10}$$

This way patches of higher certainty are selected. The dimensionality of each patch is then reduced by defuzzification using the Center of Gravity (CoG):

$$p'^n = \frac{\sum_{i=1}^{k}\sum_{j=1}^{k}\left(\pi'^n_{i,j} \cdot\ p_{i,j}^n\right)}{\sum_{i=1}^{k}\sum_{j=1}^{k}\pi'^n_{i,j}}, n = 1 \dots\ z \tag{3.11}$$

where $p' = \left\{p'^n \mid\ n = 1,2, \dots, z\right\}$.

**Algorithm 3.1** Proposed Fuzzy Pooling.

```
Algorithm
Input:   Input Volume $\beta$
1:       $P$ = Extract patches from $\beta$
2:       foreach ($p$ in patches $P$) do
3:           for ($v = 1 \dots V$) do
4:               for ($n = 1 \dots z$) do
5:                   Calculate $\pi_v^n$ using Eq. (3.8)
6:               end for
7:           end for
8:           for ($v = 1 \dots V$) do
9:               for ($n = 1 \dots z$) do
10:                  Calculate the scores $s_{\pi_v}^n$ using Eq. (3.9)
11:              end for
12:          end for
13:          Calculate $\pi'$ using Eq. (3.10)
14:          Calculate $p'$ using Eq. (3.11)
15:      end for
Output:  return $p'$
```

## 3.4 Evaluation Methodology

To evaluate the performance of the proposed methodology we conducted experiments on widely used, publicly available datasets. The experimental investigation is divided in two parts. The first part compares the proposed, over current pooling approaches with respect to classification performance. The second part performs a qualitative assessment of the proposed fuzzy pooling, aiming to investigate why it favors the classification performance.

### 3.4.1 Classification Results

Classification performance was assessed using MNIST (Deng 2012), Fashion-MNIST (Xiao et al. 2017) and CIFAR-10 (Krizhevsky et al. 2009) datasets. MNIST dataset **(Figure 3.2)** consists of 70.000 grayscale 28×28 pixels in size images of handwritten digits split into two subsets from which 60.000 are used for training and 10.000 for testing. Fashion-MNIST **(Figure 3.3)** includes images having the same size with the original MNIST dataset, but instead of classes of handwritten digits it includes classes of clothes. This renders the classification problem more challenging, especially on swallower networks, such as the one used in this chapter. CIFAR-10 dataset **(Figure 3.4)** can be considered as the equivalent of MNIST dataset in natural images. It consists of 60.000 natural RGB images of 32×32 pixels in size from 10 different classes from which, 50.000 are used for training and 10.000 for testing. The dataset contains 6.000 images per class. We have selected these datasets as they are relatively simpler compared to other, larger datasets such as ImageNet (Deng et al. 2009), which would require complicated CNN architectures to be used in order to yield any meaningful results.

In an effort to minimize the performance bias introduced by the high number of hyper-parameters required by deep CNN architectures, such as EfficientNet (Tan & Q. Le 2019), ResNet (He et al. 2016), VGGNet (Simonyan & Zisserman 2014), we choose to evaluate the classification performance of the proposed pooling methodology using the LeNet (LeCun et al. 1989) baseline CNN architecture **(Figure 3.5)**. Although the classification performance of such a baseline architecture is significantly lower compared to state-of-the-art architectures, it offers a relatively low number of hyper and free-parameters (weights) which highlights the performance impact of pooling.

In the convolutional layer, the capped ReLU activation (3) is used. The proposed pooling layer performs fuzzy pooling using the fuzzy membership functions defined in (5-7), with $r_{max} = 6$, as suggested in (Howard et al. 2017). Thus, the parameters of the membership functions are $d = 3$, $c = 1$, $a = 1.5$, $m = 3$, $r = 3$, and $q = 4.5$.

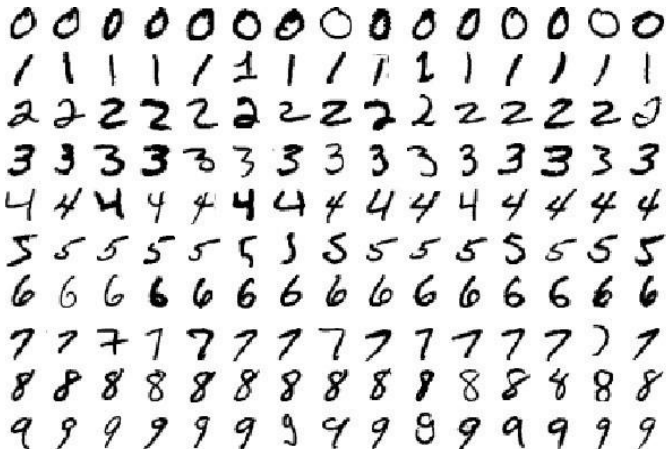


**Figure 3.2** Sample images of the 10 classes from MNIST (Deng 2012) dataset

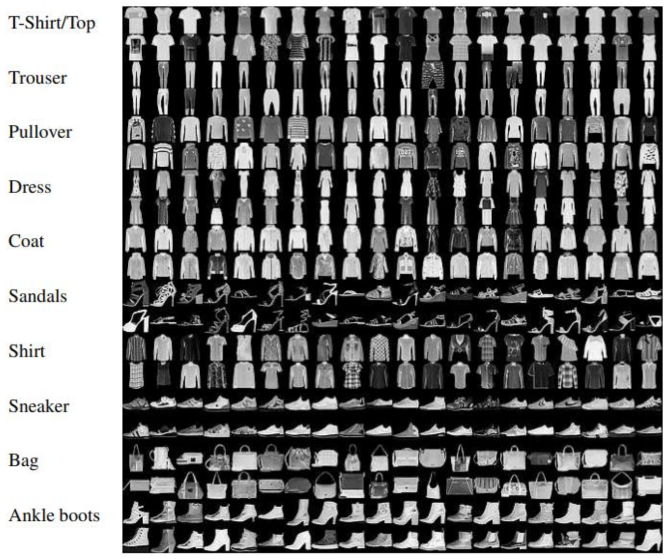


**Figure 3.3** Sample images from the 10 classes of Fashion-MNIST (Xiao et al. 2017) dataset.

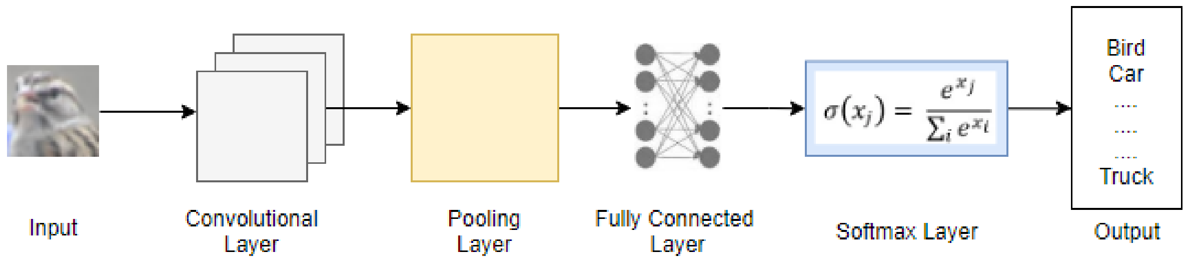


**Figure 3.4** Standard CNN LeNet (LeCun et al. 1989) architecture

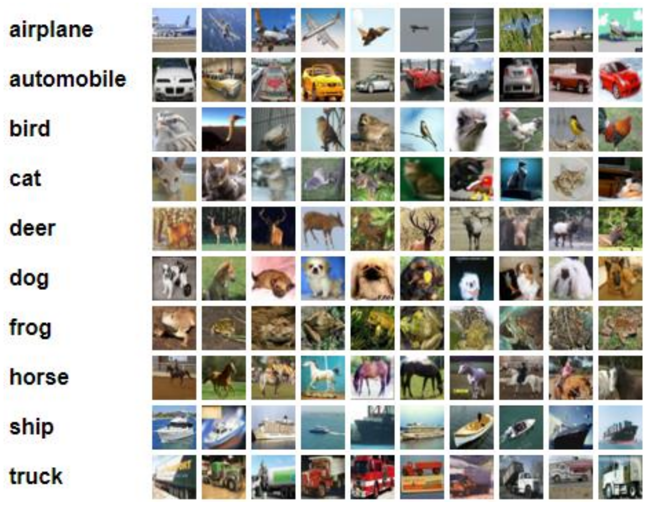


**Figure 3.5** Sample images from the 10 classes of CIFAR-10 (Krizhevsky et al. 2009) dataset.

For the baseline architecture training we used the Stochastic Gradient Descent (SGD) (Kiefer et al. 1952) with a batch-size of 32 images. We did not perform any type of data-preprocessing or data-augmentation, in an effort to keep the experiments focused solely on the impact of the selection of the pooling layer on the overall classification performance of the network. All experiments were conducted using the same software and hardware equipment. More specifically, the proposed pooling was implemented using the Tensorflow (Abadi et al. 2016) framework in Python, which is a popular framework for deep learning applications, enabling training and inference of the model to be conducted on Graphical Processing Units (GPUs). All the experiments were conducted using the training and testing subsets provided by the datasets, which are class-balanced. For this reason, to assess the classification performance of the proposed pooling, classification accuracy was used as a sufficient measure. Comparative evaluations were conducted using the same, baseline architecture described above, switching only the pooling layer of the baseline architecture. The methods considered for comparison include the max-pooling, the average pooling, the state-of-the-art RegP (Yildirim & Baloglu 2019) and the type-2 fuzzy pooling (Sharma et al. 2019). The results obtained per dataset are presented in **Tables 3.1, 3.2** and **3.3**, respectively.

It can be noticed that the proposed methodology outperforms the existing state-of-the-art and conventional pooling approaches. This can be attributed to the value selection approach that it follows, which is based entirely on fuzzy logic. The results show that the classification performance improvement is independent from the dataset used. On the contrary, the type-2 fuzzy pooling approach (Sharma et al. 2019), does not perform well on the more complex CIFAR-10 dataset.

**Table 3.1** Comparative Accuracy Results of the Type-1 Fuzzy Pooling Methodology on MINST Dataset (Deng 2012)

| Methodology | Classification Accuracy |
|---|---:|
| Max Pooling | 88.48% |
| Average Pooling | 94.06% |
| RegP (Yildirim & Baloglu 2019) | 95.46% |
| Type-2 Fuzzy Pooling (Sharma et al. 2019) | 94.40% |
| Fuzzy Pooling (Diamantis & Iakovidis 2020) | **98.56%** |

**Table 3.2** Comparative Accuracy Results of the Type-1 Fuzzy Pooling Methodology on CIFAR-10 Dataset (Krizhevsky et al. 2009)

| Methodology | Classification Accuracy |
|---|---:|
| Max Pooling | 70.73% |
| Average Pooling | 74.83% |
| RegP (Yildirim & Baloglu 2019) | 75.44% |
| Type-2 Fuzzy Pooling (Sharma et al. 2019) | 27.92% |
| Fuzzy Pooling (Diamantis & Iakovidis 2020) | **78.35%** |

**Table 3.3** Comparative Accuracy Results of the Type-1 Fuzzy Pooling Methodology on Fashion-MNIST Dataset (Xiao et al. 2017)

| Methodology | Classification Accuracy |
|---|---:|
| Max Pooling | 84.28% |
| Average Pooling | 85.90% |
| RegP (Yildirim & Baloglu 2019) | 86.41% |
| Type-2 Fuzzy Pooling (Sharma et al. 2019) | N/A |
| Fuzzy Pooling (Diamantis & Iakovidis 2020) | **88.57%** |

### 3.4.2 Qualitative Assessment

As noted in the previous subsection, the performance advantage of the proposed pooling approach relies on the feature selection strategy it applies. However, to obtain a deeper understanding of how it affects the feature maps of the CNN, a qualitative assessment of its effects has been performed. Considering that the feature maps are 2D image representations, to obtain visually meaningful results we performed comparisons on a collection of standard images obtained from the USC Image Database – Miscellaneous dataset (Anon n.d.). The dataset consists of 44 images from which 16 are RGB images and 28 grayscale. The aspect ratio of the images is 1:1 while the spatial size is 256×256, 512×512 or 1024×1024 pixels. To conduct the experiments uniformly across the images, as a pre-processing step, the images were downscaled to 256×256 pixels in size and converted to grayscale. To assess the effect of pooling on these images, different quality metrics were estimated after the pooling operations. These include the Root Mean Square (RMS) contrast (Peli 1990), Peak Signal-to-Noise Ratio (PSNR) (Instruments 2013) and the Structural Similarity Index (SSIM) (Wang et al. 2004) between the original image and the resulting image after pooling using max, average and the proposed fuzzy pooling. RMS contrast is defined as the standard deviation of the pixel intensities; therefore, a larger value of RMS contrast indicates a better contrast. PSNR measures image quality with respect to distortions, in decibels (dB); with the higher quality images to have higher PSNR values. SSIM is an index that considers image degradation as perceived change in structural information; with the SSIM for non-degraded images to be equal to 1. The average results are summarized in **Table 3.3**, with a standard deviation of approximately 2.3% with respect to the estimated values. The results indicate that the output images obtained with the proposed approach have a higher contrast and lower noise levels

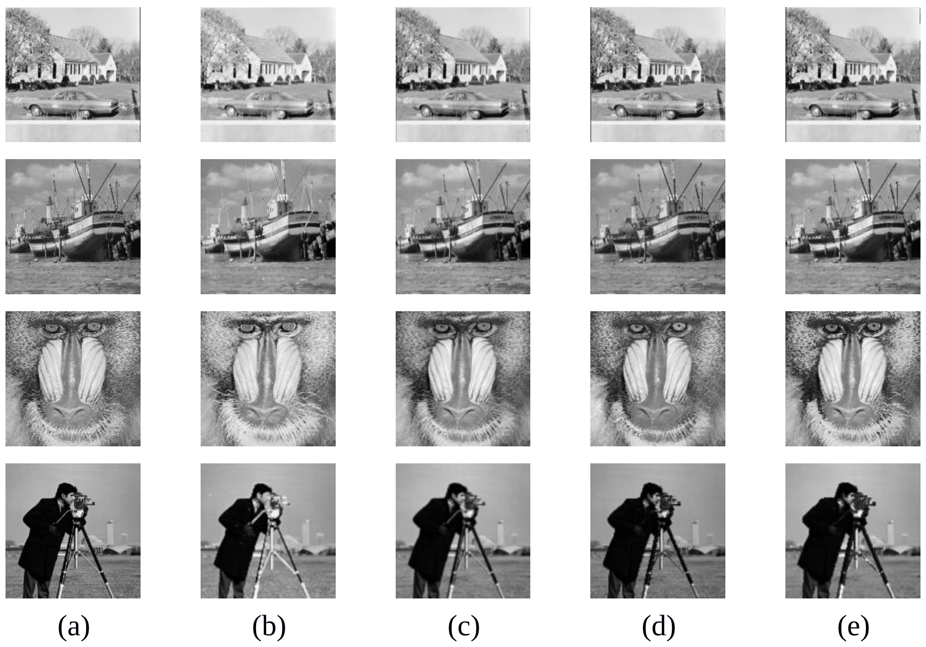


**Figure 3.6** Visual comparison of pooling results on standard images found in (Anon n.d.) and (Gonzalez & Woods 2018) datasets. The images of "House", "Fishing Boat", "Baboon" and "Cameraman" are presented on rows 1 to 4 respectively. (a) Original images, (b) Max-pooling, (c) Average pooling, (d) RegP pooling (e) Proposed type-1 fuzzy pooling.

compared to the other pooling approaches. The SSIM is a measure of the perceptual similarity of the images before and after processing. Therefore, in these terms the results indicate that the proposed approach has an obvious advantage over max-pooling, whereas it provides a visually compatible result with average pooling. However, at this point it should be noted that pooling is performed on feature maps and not directly on images, which are assessed by artificial neuron arrangements and not humans. Considering the classification results presented in the previous subsection, it can be derived that the perceptual similarity is insufficient to justify the observed performance advantage (**Table 3.1, 3.2 and 3.3**).

Figure 5 illustrates the results of the different pooling operations tested on representative images from the USC dataset. It can be noticed that the output of the average pooling operation looks smoother and therefore, more satisfactory for the human observer, which justifies the results in terms of SSIM. In most cases the max-pooling operator cause human-perceivable distortions, *e.g.*, it destroys the face of the cameraman, and it inverts the eyes of the baboon. On the contrary, the output of the proposed fuzzy pooling is both perceptually compatible and it better preserves the information of the original images, while enhancing their contrast. Also, it is worth noting that the boundaries of some objects, *e.g.*, the tripod of the camera and the region over the wheel of the car in the "house" image, look more 'digital', as compared with the original image. This effect is due to the minimization or absence of greylevel diffusion on the object boundaries in the output images. Such a diffusion observed in the original and the outputs of the compared pooling approaches can be considered as an indication of greylevel uncertainty on the object boundaries, which may be positive for human perception; however, it can limit the spatial discrimination of the features in a feature map.

**Table 3.4** Comparative Results of the Proposed Type-1 Fuzzy Pooling Methodology on USC (USC, 2018) Dataset

| Metric | Max Pooling | Average Pooling | RegP Pooling | Fuzzy Pooling |
|---|---|---|---|---|
| RMS Contrast (Peli 1990) | 44.28 | 47.98 | 46.88 | **48.19** |
| PSNR (Instruments 2013) (dB) | 5.29 | 5.29 | 5.28 | **5.42** |
| SSIM (Wang et al. 2004) | 0.72 | **0.78** | 0.73 | 0.77 |

To make these observations clearer to the reader, we have included magnifications of representative samples from the images of **(Figure 3.6)**. These samples are illustrated in **(Figure 3.7)** showing that that the proposed methodology preserves the important details of the image better than max and average pooling. It is important to note that in the second row of **(Figure 3.7)** the "mast" from the original fishing boat image, has disappeared in the case of the widely used max pooling.

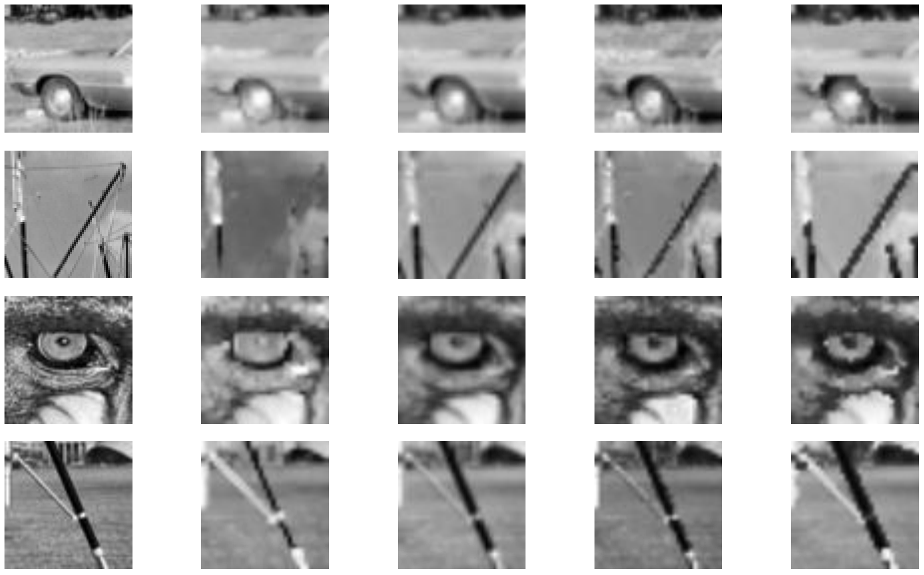
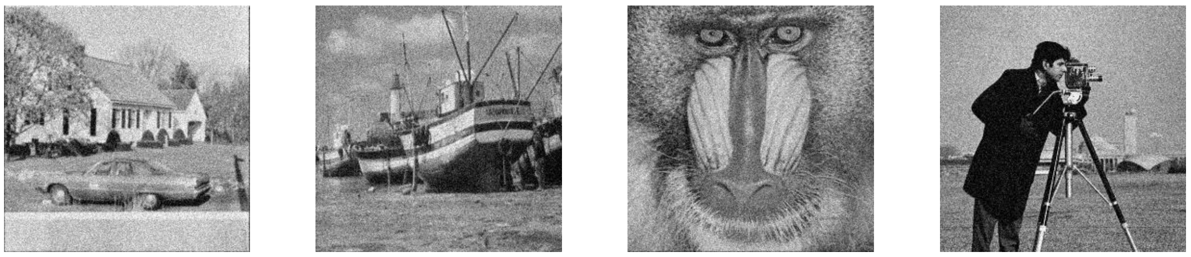


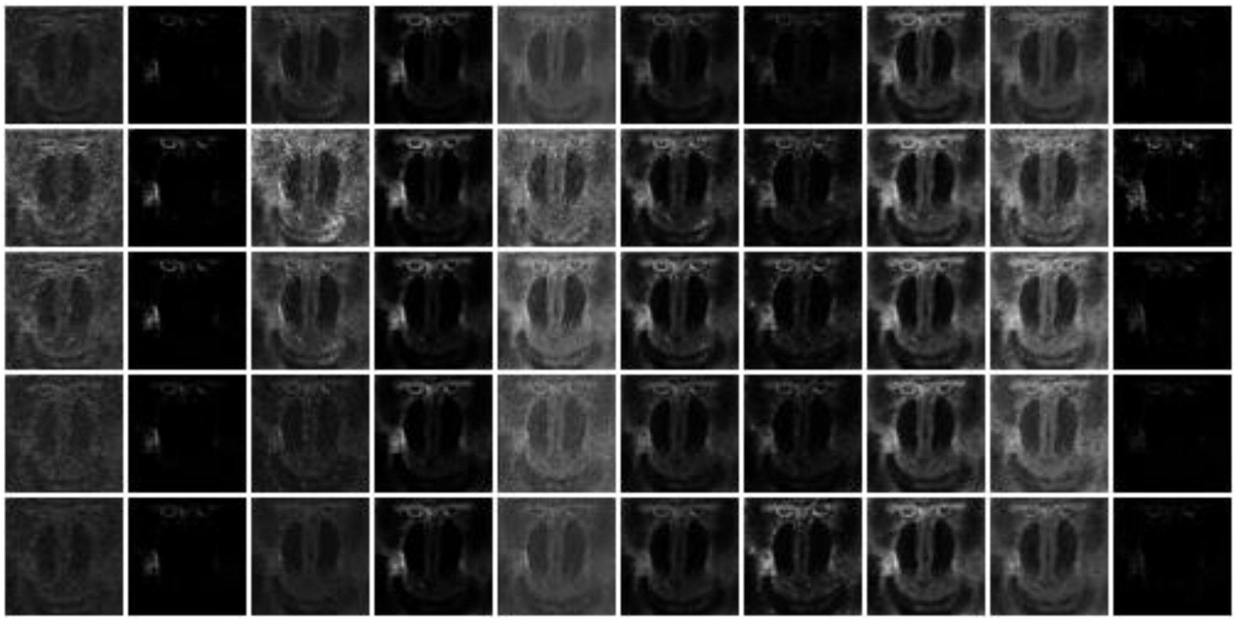
**Figure 3.7** Visual comparison of details on the images illustrated in **(Figure 3.6)**. (a) Original images, (b) Max-pooling, (c) Average pooling, (d) RegP pooling (e) Fuzzy pooling.

To provide further insights on the way the proposed fuzzy pooling copes with uncertainty propagation to the feature maps of a CNN, indicative feature map visualizations are provided in **(Figure 3.8)**. These feature maps were extracted from the network described in Section IV.A **(Figure 3.5)**, using the same input images, as the ones used to produce the results of **(Figure 3.6)**. More specifically, we selected the network that was pre-trained on CIFAR-10 (Krizhevsky et al. 2009) dataset, as this includes more general classes of objects, resembling those illustrated in the input images. To increase the uncertainty levels of the input images, Gaussian noise was added with variance 0.01, resulting in images with a PSNR of 20 dB. The various pooling methods compared, were applied on the feature maps resulting from the convolutional layer. The pooling results in **(Figure 3.8)** show that the visualizations of the different methods have significant differences with respect to their capability to preserve as many as possible from the details of the convolutional layer. It can be noticed that the proposed fuzzy pooling approach looks more similar with the output of the convolutional layer. As this may not be obvious to all readers, it can also be noticed by the average PSNR estimated per pooling methodology on these images, as a representative metric. In the case of the proposed fuzzy pooling this is 23.38 dB, whereas the respective values for max, average and RegP pooling it is 19.25 dB, 21.64 dB, and 21.51 dB, with a standard deviation of ± 0.3 dB in all cases.

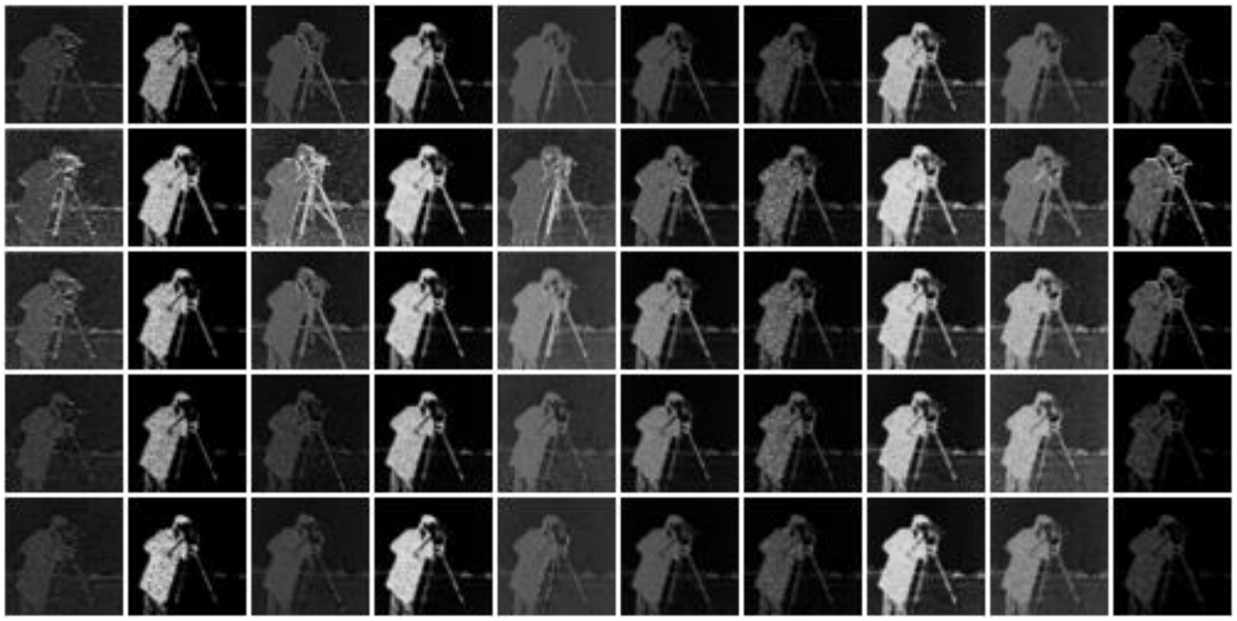

(a)

(b)

(c)



(d)



(e)

**Figure 3.8** Visual comparison of pooling results on a subset of 10 feature maps obtained by the convolutional layer of the CNN **(Figure 3.5)**. In each figure, the first row contains the original feature maps, and the rest of them the results of max-pooling, average pooling, RegP, and Fuzzy Pooling, respectively. (a) Images with Gaussian noise (b) “House”, (c) “Fishing Boat”, (d) “Baboon” and (e) “Cameraman”.

## 3.5 Conclusions

In this chapter we presented a novel *fuzzy pooling* operation for CNN architectures, coping with the uncertainty of feature values. Experiments performed on publicly available datasets, show that the proposed methodology significantly increases the classification performance of CNNs, as compared to other state-of-the-art pooling approaches. We show that fuzzy pooling can be used as a drop-in replacement of existing pooling layers, in CNN architectures, increasing the generalization performance. Furthermore, experiments conducted on standard image datasets (Anon n.d.)(Gonzalez & Woods 2018), show that the proposed methodology is able to preserve better the important features of the pooling areas. This was validated both visually and statistically by the higher classification performance obtained using the fuzzy pooling approach.

Future work includes optimization of the current implementation of fuzzy pooling to fully exploit GPU-level parallelism. This will enable us to perform larger-scale experimentation with very large datasets, such as ImageNet (Deng et al. 2009), using deeper CNN architectures, such as (Simonyan & Zisserman 2014). Other interesting research perspectives include the extension of the learnable set of network parameters to include the parameters for the fuzzy rules, and the extension of the proposed approach using generalized fuzzy sets, such as intuitionistic fuzzy sets.

# CHAPTER 4

# MACHINE LEARNING FOR COMPUTER-AIDED ENDOSCOPY

Gastrointestinal (GI) diseases are becoming more and more common (Foundation 2017). Each year in the United States 130,000 patients are diagnosed with colon cancer, making it the second most common form of cancer in the country. Recent studies show that the modern way of life, especially in the developed countries, has increased the number of cases with GI lesions. In this chapter we aim to provide a computer-aided approach for abnormality detection addressing variety of diseases, such as polyps, vascular bleeding, and inflammatory conditions.

The GI tract can be broken down into four sections, namely the esophagus, the stomach, the small intestine, and the colon. A typical examination method of the GI tract is Flexible Endoscopy (FE) (Muller & Sonnenberg 1995) and its variations (Yao 2013; Kiesslich et al. 2005; Gono et al. 2004). Wireless Capsule Endoscopy (WCE) (Swain 2008) is becoming increasingly popular as a method of capturing images from the entire GI tract due to its non-invasiveness. This method uses a swallowable camera to capture low-resolution images throughout the entire GI tract which are afterwards examined by a clinician. A lot of manual human effort is required, which is typically interpreted into 45-90 minutes work, demanding undisrupted concentration. Thus, the review of an entire WCE video is prone to human errors since the video reviewers can become tired over the time. This raises the need for a computer-aided diagnosis methodology that could increase the overall diagnostic accuracy, and reduce the required examination time.

Computer-aided abnormality detection in endoscopic images of the GI tract has been an active research subject over the last 18 years (Iakovidis & Koulaouzidis 2015; Liedlgruber & Uhl 2011; Karkanis et al. 2003). Abnormality detection refers to the ability of discriminating abnormal tissues from normal image contents. Normal image contents include non-pathologic tissues and intestinal content, such as debris and bubbles. First approaches were aiming to the detection of abnormalities in FE (Liedlgruber & Uhl 2011; Karkanis et al. 2003). In that context, abnormality detection systems contribute to the early detection of life-threatening conditions such as cancer. Their use can contribute in speeding up the FE procedures, which are generally uncomfortable for the patients. An added benefit is that cost reduction can be achieved by the use of such systems, as they could enable less experienced personnel to perform the examination.

The abnormality detection methodologies that have been proposed in the context of GI FE (Vemuri 2019; Liedlgruber & Uhl 2011) and WCE (Dray et al. 2021; Iakovidis & Koulaouzidis 2015) can be grouped into two main categories, according to the type of features used to describe the images. The methodologies of the first category are based on hand-crafted features for the representation of image properties, including color, texture and shape (Iakovidis & Koulaouzidis 2015;

Koulaouzidis et al. 2017; Vasilakakis et al. 2016; Sommen et al. 2016; Mamonov et al. 2014; D.-Y. Liu et al. 2016; Liedlgruber & Uhl 2011; Iakovidis et al. 2015; Hegenbart et al. 2013; Häfner et al. 2015; Cong et al. 2015; Cong et al. 2016). However, such features are usually selected based on considerations about the modality used to acquire the images or about the abnormalities to be detected. In the second category, which comprises more recent methodologies, the feature extraction process is automatic. This is usually implemented through adaptation on the annotated dataset used for training of the overall system. State-of-the-art approaches of this kind are based on CNNs (Tajbakhsh et al. 2015; Sekuboyina et al. 2017; Ribeiro et al. 2016; He et al. 2018; Wimmer et al. 2016).

Recently, supervised methodologies based on weakly annotated images have shown promising results for the classification of endoscopy images. The so-called weak labels are essentially keywords, only semantically describing image content. Therefore, weak labeling constitutes a time-efficient approach to obtain image annotations from the experts (Wang et al. 2015; Wang et al. 2016). In this context we proposed a MIL-based approach following the Bag of visual Words (BoW) model, for classification of GI endoscopy images (Vasilakakis et al. 2016). More recently, we proposed a methodology for weakly supervised detection and localization of abnormalities in GI endoscopy images (Iakovidis et al. 2018). This includes Weakly supervised CNN-based (WCNN) classification of the endoscopic images, followed by the detection of salient points, which were subsequently filtered by a clustering process to enable within-frame localization of the abnormalities. Specifically for bleeding detection and segmentation, a two stage approach has been proposed by (Jia & Meng 2017). Initially the images obtained from WCE, are classified as active or in-active subgroups based on handcrafted statistically derived color probability features. Then the segmentation is done using a deep FCN architecture (Springenberg et al., 2014), *i.e.*, an architecture composed of only convolutional layers.

Other CNN architectures for classification of weakly labeled images, proposed in the context of GI endoscopy, include a CNN that receives RGB images along with their Hessian and Laplacian transformations as input (Seguι et al. 2016); a cascaded CNN architecture for the recognition of the different organs of the GI tract and normal intestinal content (Chen et al. 2017); and, a CNN architecture for blood detection, using an SVM instead of the fully-connected layer of the conventional CNNs (Jia & Meng 2016). A recent, generic CNN-based approach to abnormality detection in GI endoscopy has been proposed in (Zhang et al. 2016) . It utilizes a pre-trained CNN architecture, and more specifically the CaffeNet (Jia et al. 2014), as a feature extractor. The features are extracted from the intermediate layers of the network. The extracted feature-maps are then used to train an SVM classifier. A remarkable aspect of that approach is that it was trained solely on ImageNet (Russakovsky et al. 2015), which is a large dataset of natural images that does not include any endoscopic or other relevant images.

While the usage of CNNs, including FCNs, has provided superior results compared to other conventional approaches, they generally require large training datasets. A drawback of current CNN approaches is that the use of smaller training datasets limits their generalization capacity. This derives from the fact that, as the number of the free parameters of the network increases, the need for more training examples also increases, in order to avoid overfitting (Du & Swamy 2006). However, the availability of such large training datasets in the medical domain is usually limited; thus, CNN training can become challenging. The challenge is to develop an architecture that generalizes well, even with smaller datasets. Furthermore, the increase of free parameters, increases the needs for computational resources, with a consequent deterioration of the time-performance of the system for both training and testing (Krizhevsky et al. 2012; Simonyan & Zisserman 2014). Considering that the access to high-end Graphical Processing Units (GPUs) can become costly, the development of a less resource-demanding architecture is a challenge that needs to be addressed.

To address these challenges we have proposed the Look-Behind Fully Convolutional Neural Network (LB-FCN) (Diamantis et al. 2019), which is a CNN architecture that focuses on minimizing the number of free parameters along, which enables it to generalize well even when the number of training data are limited. LB-FCN architecture is a result of two novel studies; In the first, a cross-dataset experimental study (Diamantis et al. 2018) that investigates the generalization performance of an earlier version of architecture on various publicly available datasets. The second study (Vasilakakis et al. 2018) investigated the generalization performance of a similar, multi-scale feature extraction enabled architecture, named MM-CNN, in the context of weakly-supervised multi-label endoscopy video frames classification.

In some cases, obtaining even small number of images for training is relatively hard, mainly due to the annotations that are required to be added to the images by a skilled physician. Aiming to address this problem, we recently proposed a novel GAN methodology in which GI tract images are generated automatically (Diamantis et al. 2019). The normal (healthy) and abnormal (unhealthy) generated images were used as training dataset to train the LB-FCN architecture with promising results when tested on real images.

## 4.1 The Look-Behind Fully Convolutional Neural Network Architecture

The design of the proposed architecture follows the FCN approach (Springenberg et al. 2014), where the fully-connected layers, typically used in the conventional CNN architectures (Krizhevsky et al. 2012; Simonyan & Zisserman 2014), are replaced by fully-convolutional layers. It is based on two fundamental elements, which differentiate it from other FCN architectures. These are: a) the Multi-scale Convolutional Block (MCB), which enables multi-scale feature extraction from its input, and b) the LB connection, which aims to preserve the input volume, along

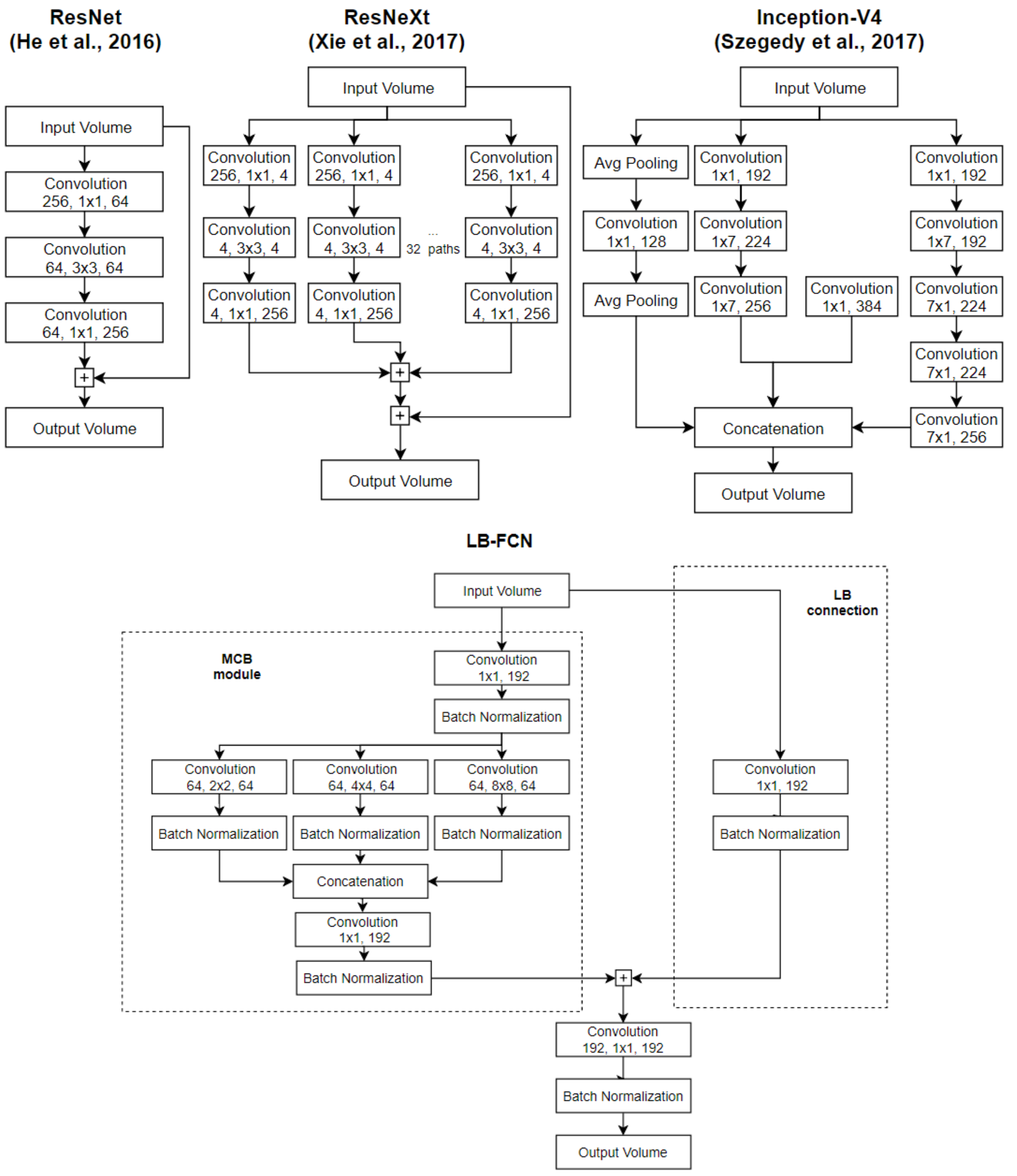


**Figure 4.1** A comparison between the core components of ResNet (He et al. 2016), ResNeXt (Xie et al. 2017), Inception-v4 (Szegedy et al. 2017) and the proposed LB-FCN (Diamantis et al. 2019)architectures. The term “volume” represents either a set of feature maps (in case of hidden network components) or a single image (in the case of network’s input).

with the extracted features per MCB. Both the MCB and the LB connection form the basic, complete structural component (module) of the LB-FCN, illustrated in **(Figure 4.1)**. The input volume, in the case of the first LB-MCB module of the LB-FCN, is an RGB endoscopic image, and in the case of subsequent layers it is the output of the previous LB-MCB modules. Overall, LB connections contribute in enhanced classification performance; however, in some cases where the performance is not significantly affected, they can be pruned for reduced computational complexity.

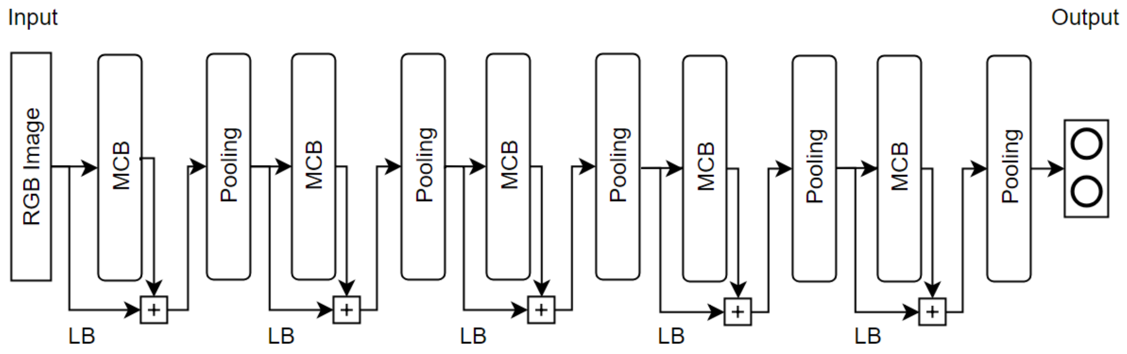


**Figure 4.2** Schematic representation of the LB-FCN (Diamantis et al. 2019) architecture proposed in this study.

The MCB is a small CNN composed of five convolutional layers **(Figure 4.1)**. The first convolutional layer performs a convolution operation with a filter size 1×1 on the input volume. The output of this layer is used as input to a parallel arrangement of three convolutional layers, with the same number of filters, performing 8×8, 4×4 and 2×2 convolution operations, respectively. This way, larger, medium, and smaller features of the input space can be captured. The choice of these filters has been driven by preliminary experimentation using various numbers of filters (1 to 5) with different sizes (from 2×2 to 12×12) on the available datasets. We observed that by using less than 3 filters the classification performance was deteriorating, whereas by using more than 3 filters the classification performance was not improving. The output feature maps of these parallel layers are concatenated and subsequently entered to the fifth convolutional layer of the MCB, with 1×1 filter size. Each MCB has a respective LB connection in parallel, forward passing its input through a 1×1 convolutional layer.

An addition operator is used to aggregate the outputs of the MCB and the LB connection. The resulting feature maps pass through a convolutional layer with 1×1 filter size. Multiple MCBs along with or without LB connections can be sequentially arranged and connected to each other.

After the aggregation of the output of the MCB with the output volume of the LB connection, a pooling operation is performed. Pooling is performed by a convolution layer of filter size 2×2 and stride 2. The usage of a convolution layer instead of a conventional max-pooling layer, was employed to introduce another level of non-linearity to the network architecture, and to unify and logically simplify the overall network architecture. The max pooling layer can be replaced by a convolution layer of appropriate size and stride without affecting the overall classification performance of the model (Springenberg et al. 2014). The 1×1 convolution operations that are used across the MCB serve two purposes. Firstly, they are helping to keep the overall number of free parameters of the network manageable and secondly, they allow the addition operation of the LB connection to the output of MCB to be valid.

An illustration of the LB-FCN architecture is included in **(Figure 4.2)**. It is composed of five modules, four of which are complete, including both MCB and LB connections as in **(Figure 4.1)**. One of them is incomplete, in the sense that it includes an MCB module without an LB connection (which was pruned as it did not contribute to an increase in the classification performance). Each MCB receives 192 feature maps as input, while for each large, medium, and small convolutional blocks, 64 feature maps are extracted. Experimentally, we have observed that by using less structural components the classification performance was deteriorating, whereas the use of more than five modules did not result in any classification performance increase either. The last layer of the network is composed of two neurons with Softmax activations, which are used as the output of the model.

All the convolutional layers have PReLU activations followed by batch normalization. Normalization is performed so that the values are centered on a zero mean with a unit standard deviation. It was empirically confirmed that this choice can contribute in a faster convergence by using higher learning rates, and also in limiting overfitting phenomena without using a dropout layer (Srivastava et al. 2014). The PReLU was chosen over the conventional non-parametric ReLU activation function, because its use has proven to be beneficial in overcoming saturation problems that have been observed with the latter during training (He et al. 2016).

## 4.1.2 Experiments and Evaluation of LB-FCN architecture

### 4.1.2.1 The Datasets

Extensive experimentation was performed to investigate the classification performance of LB-FCN on two representative datasets of different GI endoscopy modalities that include a variety of abnormalities. The first dataset (D1) was made publicly available from Endovis challenge, held in MICCAI 2015 (Navab et al. 2015). We used the data from the sub-challenge referring to the detection of abnormalities in gastroscopic images (Abnormal 2015) . The selection of this dataset over others in that challenge was driven by the diversity of the abnormalities that it included, and the fact that it also included normal images. The gastroscopy challenge dataset was derived from a total of 10,000 images obtained from 544 healthy volunteers and 519 volunteers having various abnormalities, such as cancer, bleeding, and gastritis.

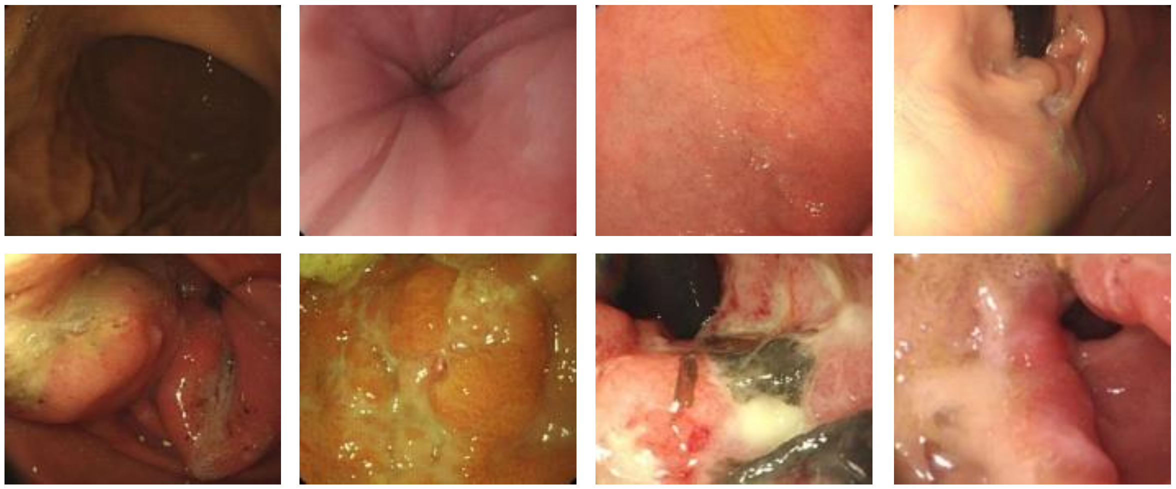


**Figure 4.3** Schematic representation of the LB-FCN (Diamantis et al. 2019) architecture proposed in this study.

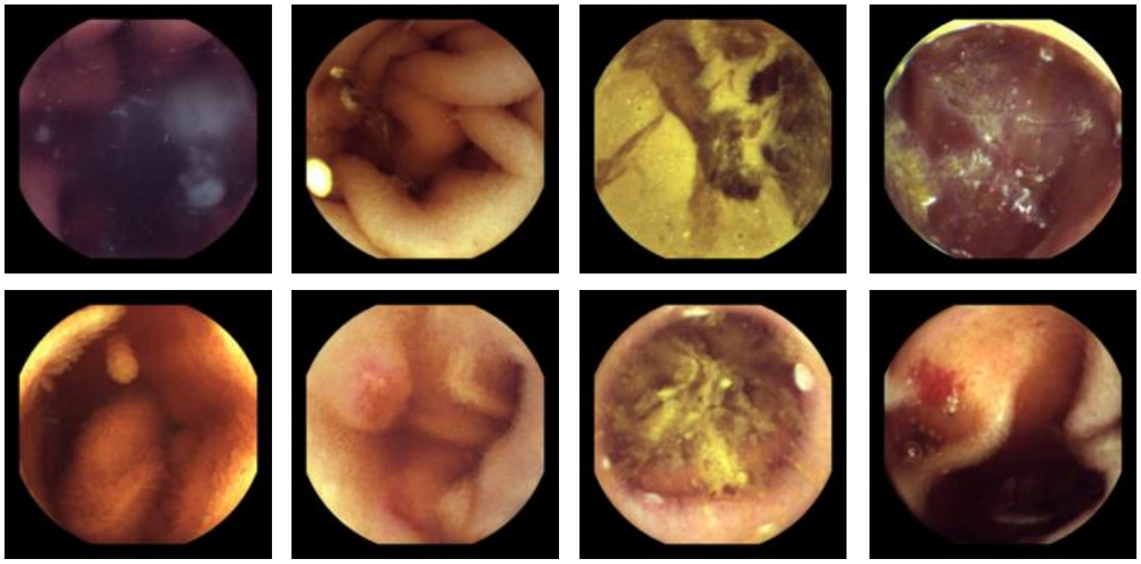


**Figure 4.4** Sample images from KID Dataset. The first row contains normal images, whereas the second row contains images with abnormalities.

The images had originally a resolution of 768×576 pixels and they were cropped by the data providers down to 489×409 pixels in order to be anonymized (Cong et al. 2015). For the purposes of the challenge a subset of 698 images from 137 volunteers was released **(Figure 4.3)**. The dataset then was split into two approximately balanced subsets; one for training with 465 images (205 normal and 260 abnormal) and one for test containing 233 images (104 normal and 129 abnormal) (Abnormal 2015)("EndoVisSub - Abnormal," 2015). This dataset will be referred to as D1B.

The second dataset (D2), originates from our database called KID (Koulaouzidis et al. 2017). This is a public, open access database of both semantically and graphically annotated WCE images and videos **(Figure 4.4)**. Dataset 2 consists of a total of 2,352 images with a resolution of 360×360 pixels. It contains 1,778 normal images from the whole GI tract, including 282 esophagus images, 599 stomach images, 728 small bowel images, and 169 images from the colon. It also contains a total of 574 images of various abnormalities found along the entire gastrointestinal tract, including vascular (303 images), polypoid (44 images) and inflammatory (227 images) conditions. It should be noted that in order to keep the dataset as realistic as possible, images with artifacts naturally occurring during a WCE procedure were not excluded. These include blurry frames, bubbles, intestinal juices, stool, and other debris.

#### 4.1.2.2 Evaluation Methodology

To evaluate the classification performance of LB-FCN architecture a comparison to state-of-the-art abnormality detection systems for GI endoscopy, a 10-fold cross-validation (CV) procedure was followed to limit the bias, *i.e.*, the dataset was randomly partitioned into 10 equally sized disjoint subsets, a single subset was retained as the validation data for testing the model, and the remaining 9 subsets was used as training data. This was repeated until all subsets are used for testing. Therefore, per CV fold, in the case of D1 a total of 628 images were used for training and 70 images were used for testing, and in the case of D2 a total of 2117 images were used for training and 235 images were used for testing. The distribution of the normal and abnormal images, as well as the distribution of the abnormal frame categories, were held approximately constant in the training and testing sets per fold, *i.e.*, 76% normal to 24% abnormal, out of which 53% were vascular, 8% were polypoid and 40% inflammatory conditions.

The metrics used to assess the classification performance include accuracy (ACC), specificity (SPC) and sensitivity (TPR), as estimated from Eq. (4.1,4.2,4.3).

$$ACC = \frac{TP + TN}{TP + TN + FP + FN} \tag{4.1}$$

$$SPC = \frac{TN}{TP + FP} \tag{4.2}$$

$$TPR = \frac{TP}{TP + FN} \tag{4.3}$$

$$FPR = 1 - SPC \tag{4.4}$$

where the number of true negatives are denoted as *TN*, true positives as *TP*, false positives *FP* and false negatives *FN*.

The Receiver Operating Characteristic (ROC) curves were considered to visualize the tradeoff between TPR and FPR at different decision thresholds. The Area Under ROC (AUC) is used as a more reliable and intuitive classification performance measure, that is insensitive to class imbalance (Fawcett 2006), which characterizes most medical datasets, as the ones used in this study.

### 4.1.2.3 Results

The proposed LB-FCN architecture was evaluated using all images of datasets D1 and D2. Since the proposed architecture uses weakly labeled images, the available graphic annotations were not used. Only semantic annotations of the images, indicating whether they contain an abnormality or not, were used as ground truth. The semantics "abnormal" and "normal" are represented as vectors (1,0) and (0,1), respectively, at the output of the network.

The network was trained using Root Mean Square Propagation (RMSProp) (Hinton et al. 2012) optimizer with an initial learning rate $n$=0.01 and fuzz factor $\varepsilon = 1e-8$. The network was implemented utilizing the Python Keras (Gulli & Pal 2017) library on top of the TensorFlow (Abadi et al. 2016) graph framework trained for 2000 epochs with mini-batch of size 32 samples on an NVIDIA GTX-960 GPU, with 1024 CUDA (Nickolls et al. 2008) cores, 2GB of RAM and clock speed of 1127MHz.

In the following, LB-FCN is compared with state-of-the-art architectures, both in terms of effectiveness and efficiency.

#### 4.1.2.3.1 Effectiveness Assessment

The evaluation of the network using CV yielded a mean Area Under Curve (AUC) of 99.72% **(Figure 5.5)**. The entire training process of each fold took 2 hours as the network had only 9 million parameters to be trained (while conventional CNN architectures used in this context, such as CaffeNet (Zhang et al. 2016) and VGG-16 (Simonyan & Zisserman 2014), have between 60-130 million parameters).

For completeness and further promote reproducibility of the results, we evaluated the classification performance of the proposed architecture on dataset D1B. This resulted in an AUC of 99.82%, which is comparable to that obtained with CV **(Figure 4.6)**.

For generality, the experiments on dataset D2 were performed using the same LB-FCN configuration with the experiments on datasets D1 and D1B. The average Receiver Operation Characteristic (ROC) curve obtained by the evaluation of the model using CV had an AUC of 93.5% **(Figure 4.7)**.

**Table 4.1** Comparative abnormality detection results using 10-fold cross-validation on datasets D1, D1b And D2, as they were obtained for an optimum selection of the meta-parameters of the compared nets and methods.

| | *LB-FCN* (Diamantis et al. 2019) | | | *BoW* (Vasilakakis et al. 2016) | | | *Transfer Learning* (Zhang et al. 2016) | | | *WCNN* (Iakovidis et al. 2018) | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | D1 | D2 | D1B | D1 | D2 | D1B | D1 | D2 | D1B | D1 | D2 | D1B |
| Accuracy (%) | **97.84** | **88.29** | **97.42** | 89.20 | 76.80 | 90.56 | 89.90 | 80.01 | 90.98 | 89.90 | 77.50 | 90.90 |
| Sensitivity (%) | **98.05** | **92.11** | **97.11** | 91.10 | 45.40 | 90.70 | 90.70 | 86.22 | 91.40 | 90.70 | 36.20 | 93.00 |
| Specificity (%) | **97.67** | 76.49 | 97.67 | 87.20 | 88.60 | 90.38 | 88.20 | 60.78 | 90.38 | 88.20 | **91.30** | 88.50 |
| AUC (%) | **99.72** | **93.50** | **99.82** | 94.60 | 80.20 | 95.44 | 96.30 | 81.62 | 96.34 | 96.30 | 81.40 | 96.84 |
| | *FCN* (Springenberg et al. 2014) | | | *ResNet* (He et al. 2016) | | | *ResNeXt* (Xie et al. 2017) | | | *Inception-v4* (Szegedy et al. 2017) | | |
| | D1 | D2 | D1B | D1 | D2 | D1B | D1 | D2 | D1B | D1 | D2 | D1B |
| Accuracy (%) | 97.13 | 87.60 | 96.13 | 96.27 | 87.22 | 95.99 | 95.98 | 86.93 | 96.56 | 95.98 | 87.92 | 96.99 |
| Sensitivity (%) | 96.66 | 77.87 | 95.34 | 97.94 | 83.26 | 95.12 | 95.63 | 80.13 | 96.89 | 94.60 | 82.17 | 96.89 |
| Specificity (%) | 97.53 | 85.97 | 97.11 | 96.04 | 90.42 | **98.07** | 96.44 | 89.13 | 96.15 | 97.53 | 90.71 | 97.11 |
| AUC (%) | 97.19 | 81.92 | 96.23 | 97.19 | 86.84 | 97.10 | 96.03 | 84.63 | 96.52 | 96.16 | 86.44 | 97.03 |

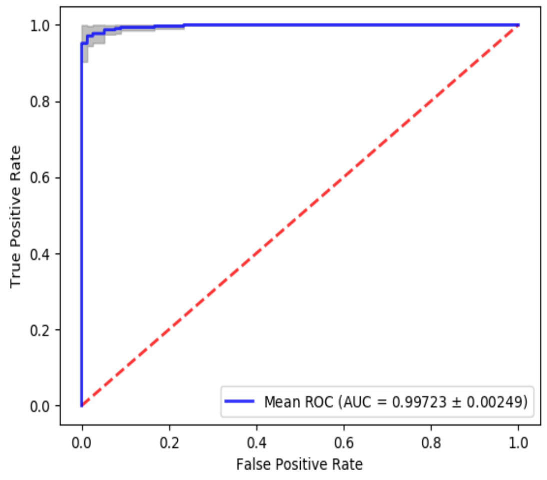


**Figure 4.5** Mean ROC obtained by 10-fold CV on dataset D1, using LB-FCN. The grey area around the curve the represents the respective confidence band.

To compare the performance of LB-FCN for abnormality detection in endoscopic images we implemented the most relevant state-of-the-art CNN architectures, including Transfer Learning (Zhang et al., 2017), FCN (Springenberg et al. 2014), ResNet (He et al. 2016), (Xie et al. 2017) ResNeXt and Inception-v4 (Szegedy et al. 2017). The results obtained on the same datasets, using 10-fold cross validation, are summarized in **Table 4.1**. In the same table provide the results of

BoW (Vasilakakis et al. 2016) and WCNN (Iakovidis et al. 2018) as they were recently reported on the same datasets using the same evaluation procedure.

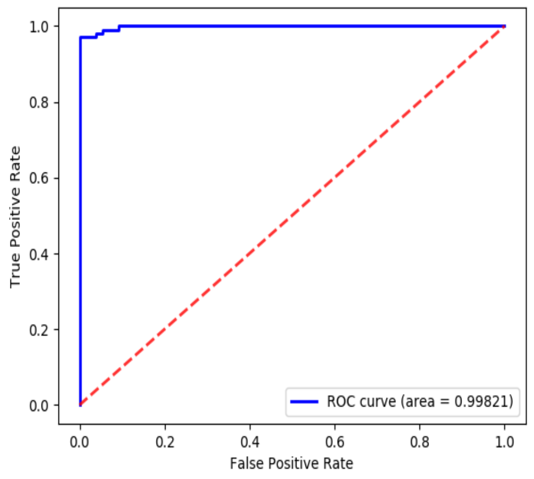


**Figure 4.6** ROC obtained on dataset D1B, using LB-FCN.

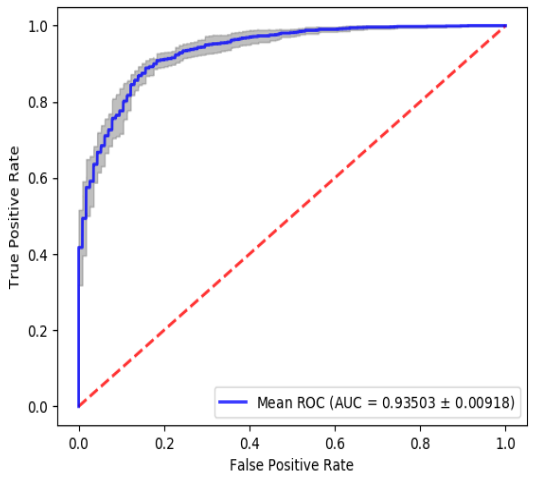


**Figure 4.7** Mean ROC obtained by 10-fold CV on dataset D1, using LB-FCN. The grey area around the curve the represents the respective confidence band.

To validate the significance of the results obtained by the proposed architecture in comparison to the results obtained with the state-of-the-art architectures and methods we performed two statistical significance tests; a non-parametric Friedman test and a two-sided Wilcoxon rank sum test (Wilcoxon 1947), In both tests the null hypothesis (*i.e.*, that the samples are derived by identical continuous distribution with equal means and are independent) was rejected (*p*-value < 0.05), which indicates that there are differences between the examined methods at a 5% significance level.

**Table 4.2** Comparison of the computational complexity of the top-ranked state-of-the-art architectures of **Table 4.1**.

| Architecture | FLOPs | Convolution Layers | Trainable Free Parameters |
|---|---|---|---|
| LB-FCN (Diamantis et al. 2019) | **$1.36 \times 10^7$** | **32** | $8.26 \times 10^6$ |
| ResNet (He et al. 2016) | $4.74 \times 10^7$ | 53 | $2.35 \times 10^7$ |
| ResNeXt (Xie et al. 2017) | $2.83 \times 10^7$ | 94 | **$5.63 \times 10^6$** |
| Inception-v4 (Szegedy et al. 2017) | $2.05 \times 10^8$ | 149 | $4.11 \times 10^7$ |

### 4.1.2.3.2 Efficiency Assessment

The total time-complexity of all convolution layers in a CNN is given by (He & Sun 2015):

$$O\left(\sum_{l=1}^{d} n_{l-1} \cdot s_l^2 \cdot n_l \cdot m_l^2\right) \tag{4.5}$$

where $l$ is the index of convolution layer, $d$ is the number of all convolution layers; $n_l$ is the width of the *l-th* layer, $n_{l-1}$is the input channels of the *l-th* layer; $s_l$ and $m_l$are the spatial size of the filter and the output feature map, respectively. Therefore, for a given dataset with a specific size, the number of computations are proportional to the sum of products described in (5). The same time-complexity applies to both training and testing time, although on different scale; as training time involves one forward and two backward passes because of the error back propagation training algorithm. As a result, the training time of an image is roughly three times the testing time of an image.

The top-ranked architectures of **Table 4.1** are compared in terms of computational complexity in **Table 4.2**. It includes the number of Floating-Point Operations (FLOPs), the number of convolution layers and the number of trainable free parameters. It can be noticed that LB-FCN has a comparable number of trainable parameters (and consequently similar memory requirements) with the rest of the architectures. However, the number of FLOPs of LB-FCN is smaller; more specifically, it is approximately 3.5 times smaller than ResNet, 2 times smaller than ResNeXt, and an order of magnitude smaller than Inception-v4.

It is worth noting that the LB-FCN architecture tends to converge faster. For example, the average number of epochs for the convergence of the LB-FCN architecture on dataset D2 was approximately 2000 as compared to 2400, 2900, 3100 epochs for ResNeXt, ResNet and Inception-v4 respectively. Also, for the same dataset, the average training time of the proposed architecture, is significantly smaller, compared to the rest of tested networks, averaging to 2 hours. ResNeXt required on average 3.4 hours, ResNet 4.2 hours, and Inveption-v4 6 hours.

#### 4.1.2.3.1 Findings

The results obtained from the application of LB-FCN architecture on datasets D1, D1B and D2, show that it outperforms the state-of-the-art architectures and methods in terms of AUC. The difference in AUC of LB-FCN in D1 and D1B datasets from the ResNet, which is the second-best performing methodology, are 2.53% and 2.72% respectively. On the larger and even more diverse dataset D2, the classification performance of LB-FCN is significantly higher, reaching a difference in AUC of 6.66% from the same methodology. We believe that this is due to the multiscale feature extraction capabilities of the architecture. It is important to note here that the methodologies that follow handcrafted feature classification technique (Vasilakakis et al. 2016), are significantly lower than the CNN based approaches.

The multi-scale design of LB-FCN is inspired by GoogLeNet (Szegedy et al. 2015) and its variation Inception-v4 architecture (Szegedy et al. 2017), were features of different abstraction levels are combined to produce a richer representation of the input volume. The LB connections are inspired by the work of Gers and Schmidhuber (Gers and Schmidhuber, 2000) where, the so-called peephole connections, were introduced in Long-Short-Term Memory (LSTM) networks to provide their gates with the ability to maintain information from previous states of the network.

The multiscale feature extraction approach used in MCB is similar but not the same with that of the Inception module of GoogLeNet architecture. While both modules aim to the same goal (multiscale feature extraction), MCB does not use pooling to perform downsampling of the input volume. This enables the LB connection to be aggregated as is, to the output volume of the MCB. Furthermore, the MCB extracts the same number of feature maps, across the entire network, on three different scales (convolution with 2×2, 4×4 and 8×8 filters and stride 1) instead of two (convolution with 3×3 and 5×5 filters and stride 2), extracted by the Inception module. This aims to provide more diverse feature representation of the input volume. In the case of GoogLeNet the number of feature maps extracted by each inception module is doubled, whenever the input volume size is downscaled by two. In the case of LB-FCN architecture, doubling the number of feature maps of the LB-MCB module after every convolutional pooling, did not yield any better results, yet it increased the number of the overall free parameters of the network. Also, it is important to note that unlike the proposed LB-FCN architecture, GoogLeNet neither has any peephole-like connections nor any addition operator **(Figure 5.1)**. Also, unlike GoogLeNet, LB-FCN does not utilize any fully-connected and dropout layers. Other differences of LB-FCN with GoogLeNet include the use of PReLU activations and batch normalization.

Peephole-like connections, known as “identity shortcut connections”, have been adopted in the Residual Blocks (RBs) of  ResNet (He et al. 2016). The RBs do not have MCB-like, parallel layers; instead, it is composed of sequentially arranged convolutional layers. In each RB the shortcut connection transfers its input volume unaltered for addition with the output of its last convolutional layer. The role of the shortcut connection is similar to that of LB. A variant of ResNet architecture, called ResNeXt (Xie et al. 2017) utilizes parallel but not multiscale convolutional layers, as the LB-FCN does. In the case of ResNeXt the outputs of the parallel layers are first summed up together and the resulting feature maps are subsequently aggregated with the output of the shortcut connection using an addition operator. In LB-FCN the outputs of the parallel layers of the MCB are not summed up, instead they are concatenated to form a richer output volume. Similarly to ResNeXt, the output of the MCB is aggregated by addition with the output of the LB connection. This allows the next MCB module to have an aggregated view of the initial input volume with the MCB output volume, which helps the overall classification performance of the network. A notable difference between ResNext and LB-FCN is not only that the MCB extracts feature maps of different filter sizes but that after each convolution operation the output is passed to a 1×1 convolution operation with batch output normalization. We found that the addition of this step increased the classification performance of the network approximately by 1.3% and reduced the training time by 5.6%.

Similarly to our recent WCNN architecture (Iakovidis et al. 2018), and unlike most relevant abnormality detection methodologies (Iakovidis & Koulaouzidis 2015), LB-FCN aims to the detection (not the identification) of various types of abnormalities, and not just a single pathologic condition. From the results we observe a significant advantage of the LB-FCN over WCNN. The differences are 12.1%, 3.42% and 2.98% on D2, D1 and D1B datasets, respectively). We believe that this advantage is due to the introduction of multiscale feature extraction combined with look-behind connections and the overall deeper architecture. The results of ResNet, ResNeXt and Inception-v4 architectures, confirm that the depth of the network plays an essential role on the overall classification performance. This can be observed on the more complex and diverse datasets such as D2, on which all three architectures outperform the classification performance of the shallower WCNN architecture.

The comparative study shows that the hand-crafted feature extraction approach used in (Vasilakakis et al. 2016) results in a significantly lower classification performance compared to LB-FCN architecture. The AUC differences observed are 13.3% and 5.12% on D2 and D1 datasets respectively. It also shows that the LB-FCN outperforms WCNN (Zhang et al. 2016); consequently it outperforms other state-of-the-art CNN-based methodologies, such as (Sekuboyina et al. 2017). It is also worth noting that D1B dataset was introduced in the work of (Cong et al. 2015), in which DSSVM approach was presented achieving an AUC of 89.83%. It is worth mentioning that the AUC difference between DSSVM and LB-FCN on D1B dataset is 9.99%.

## 4.2 Cross-Dataset Abnormality Detection

The multi-scale feature extraction CNN (MFCNN) (Diamantis et al. 2018)  architecture on which LB-FCN architecture is based on, was inspired by our work on the problem of cross-dataset abnormality detection in GI tract images. MFCNN. The study aims to investigate the generalization performance of architecture, compared with conventional, hand-crafted feature techniques on various publicly available GI tract image datasets. The results validate that the MFCNN architecture outperforms state-of-the-art approaches, with results reaching up to 90.66% in terms of the area under the receiver operating characteristic.

### 4.2.1 Evaluation Methodology

In order to evaluate the classification performance of the MFCNN architecture we performed two different sets of experiments. The first set assessed its performance on a single dataset, as a reference, whereas the second set, assessed its ability to generalize using various different datasets for training and testing. The Area under the Receiver Operating Characteristic (AUC) was used as a reliable classification measure, that is not affected by the fact that the class distributions of the datasets used were generally imbalanced (Fawcett 2006). The overall implementation was based on the Keras (Gulli & Pal 2017) library backed by TensorFlow (Abadi et al. 2016) graph framework.  For the training we used RMSProp (Hinton et al. 2012) optimizer with learning rate n=0.001, a fuzz factor ε=1e-8, and a batch size of 32 images.

The single-dataset evaluation was based on the largest and most diverse publicly available dataset of gastrointestinal images, which is “Dataset 2” of KID database (Koulaouzidis et al. 2017). The evaluation was performed by 10-fold cross validation obtaining an average AUC of 93.5%. For comparison purposes we implemented and tested the transfer-learning approach proposed by (Zhang et al. 2016), which resulted in an AUC of 81.62%.

#### 4.2.1.1 The Datasets

Cross-dataset evaluation was based on four datasets: KID (Koulaouzidis et al. 2017), Gastroscopy (GASTRO) (Cong et al. 2015) , CVC-ClinicDB (CVC) (Bernal et al. 2015), ETIS-Larib Polyp DB (ETIS) (Silva et al. 2014) dataset. The last three datasets were made publicly available as part of the Endovis Grand Challenge which was held in MICCAI 2015 conference (Navab et al. 2015). The Gastroscopy dataset contains a total of 698 gastroscopic images of size 489×409 pixels from which 389 contain various pathologic conditions. The rest are normal images. CVC-ClinicDB and ETIS-Larib Polyp DB contain only abnormal images with polyps that were obtained from colonoscopy videos. The former contains 612 images with a size of 384×288 pixels, and the latter contains 196 high resolution images with a size of 1225×966 pixels. The lack of normal images in these datasets makes them unsuitable for weakly-supervised training of the proposed architecture, because both normal and abnormal images are required for this purpose.

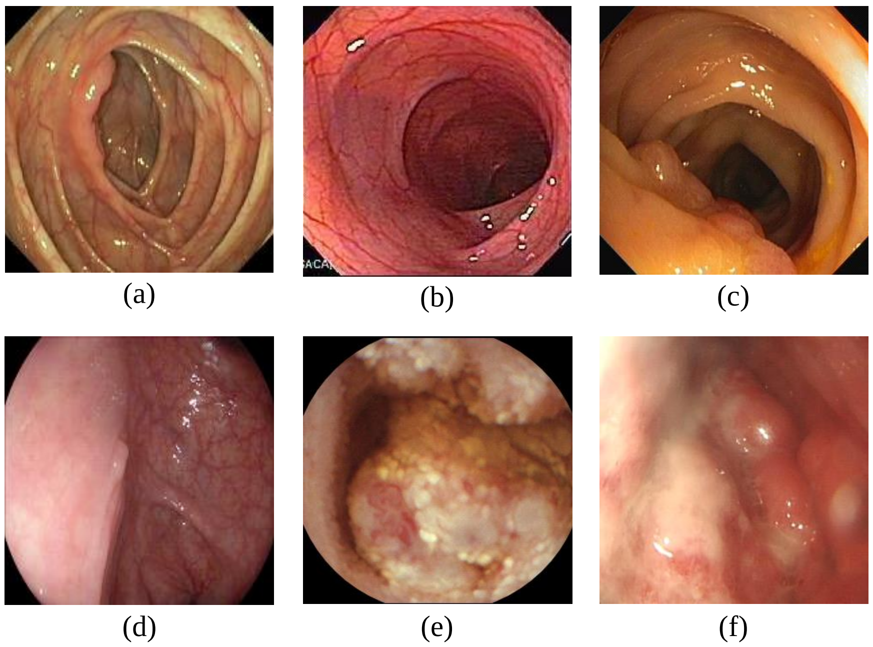


**Figure 4.8** Sample images from (a) GLAB, (b) GATLAS, (c) CVC, (d) ETIS, (e) KID and (f) GASTRO datasets.

In order to be able to use the CVC-ClinicDB and ETIS-Larib Polyp DB for training, we enriched them with 302 images from 10 normal colonoscopic videos from the publicly available GASTROLAB (GLAB) (GASTROLAB 2018) and El Salvador Atlas of Gastrointestinal Endoscopy (GATLAS) (Atlas 2018) video databases. GLAB dataset consists of images from the videos with filenames: vid132, vid146, vid148, vid149, vid176 and vid177, and GATLAS consists of images from videos: colonoscopy, colonoscopy2, videocolonoscopy2, and videocolonoscopy3. The images were sampled with a period of 1 sec. GLAB dataset was combined with CVC dataset (CVC+GLAB) and GATLAS was combined with ETIS dataset (ETIS+GATLAS). All images were linearly downscaled to 64×64 pixels before entering the CNN. A sample image from each dataset that was used in this study is presented in **(Figure 4.8)**.

#### 4.2.1.2 Results

The results obtained from the evaluation of the proposed architecture in comparison to the results obtained by the architecture used in (Zhang et al. 2016) trained and tested on the respective datasets are presented in **Table 4.3**. Training with ETIS+GATLAS dataset and testing on CVC+GLAB yield the best results (90.66%), which we believe is due to the similarity of the two datasets, as they are both colonoscopic datasets with polyp lesions. The lowest score (57.83%) was obtained upon training the architecture with ETIS+GATLAS dataset and testing it on KID dataset. This can be explained by the fact that KID is a very diverse dataset, containing images from the entire GI tract, not only from the colon. The fact that ETIS is smaller than CVC dataset can explain the lower performance of the network trained with ETIS+GATLAS for the classification of KID images, as compared with CVC+GLAB. Comparing the results of the MFCNN architecture with the ones obtained using (Zhang et al. 2016) approach we notice the former yields significantly higher classification performance. We believe that this is due to the training image relevancy, as (Zhang et al. 2016) features are obtained from a pre-trained network which was trained on non-medical images. It is also important to note that (Zhang et al. 2016) approach utilizes a network of approximately 60 million parameters while the proposed architecture has only 9 million. Furthermore, the image classification performed with the MFCNN methodology is handled by the network itself, whereas (Zhang et al. 2016) employs an SVM binary classifier for the classification of the extracted features.

**Table 4.3** Comparative abnormality detection results on cross-dataset evaluation.

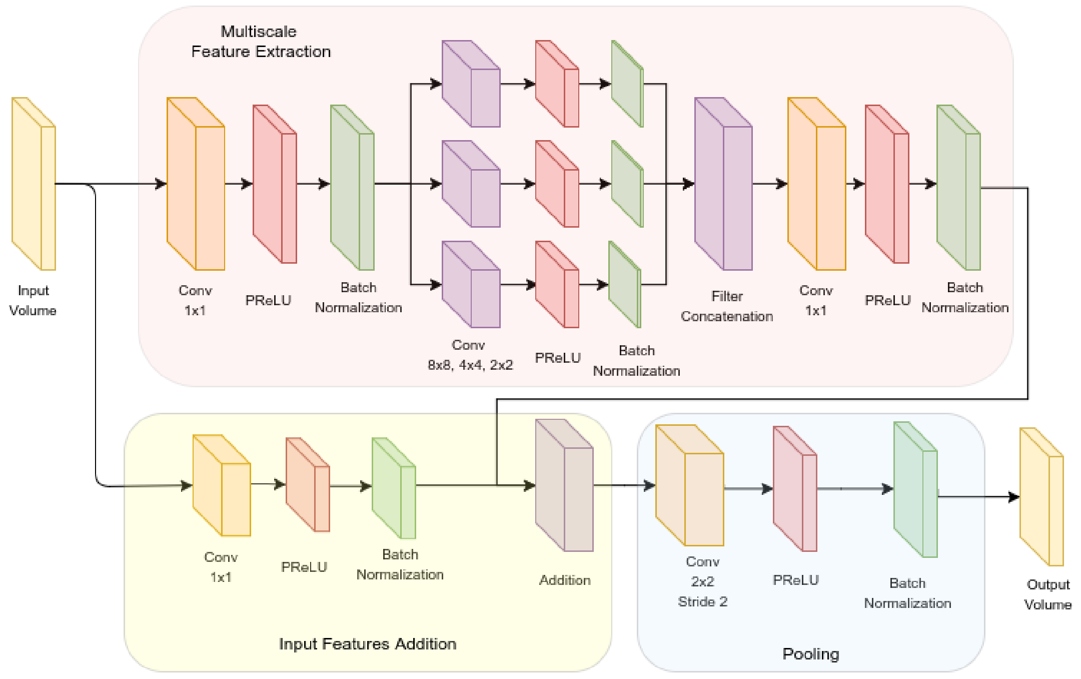
| Train | Test | MFCNN | (Zhang et al. 2016) |
|---|---|---:|---:|
| KID | GASTRO | **82.85%** | 58.41% |
| KID | CVC+GLAB | **61.40%** | 50.32% |
| KID | ETIS+GATLAS | **62.99%** | 50.93% |
| GASTRO | KID | **62.37%** | 53.84% |
| GASTRO | CVC+GLAB | **71.32%** | 50.40% |
| GASTRO | ETIS+GATLAS | **80.17%** | 50.18% |
| CVC+GLAB | KID | **64.12%** | 50.39% |
| CVC+GLAB | GASTRO | **67.87%** | 50.28% |
| CVC+GLAB | ETIS+GATLAS | **76.28%** | 65.80% |
| ETIS+GATLAS | KID | **57.83%** | 50.19% |
| ETIS+GATLAS | GASTRO | 57.91% | **62.68%** |
| ETIS+GATLAS | CVC+GLAB | **90.66%** | 50.30% |

## 4.3 Weakly Supervised Multilabel classification for Semantic Interpretation of Endoscopy Video Frames

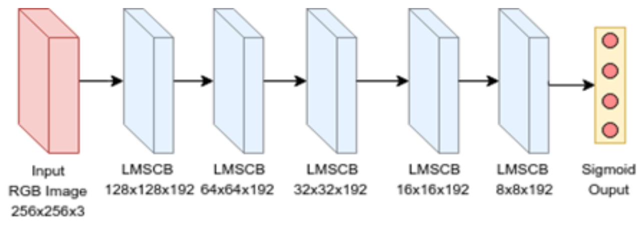
A variety of studies address the problem of abnormality detection in medical images using computer-based systems. Most of these systems are based on binary classification algorithms that rely on fully annotated data in order to operate. In image classification problems such annotations require pixel-level selection within the image that indicate the location of these abnormalities.

Such annotations are relatively hard, and time consuming to obtain. Furthermore, they do not take into consideration the diversity of the image content, which may include a variety of structures and artifacts. In the context of GI video-endoscopy, the semantics of the normal contents of the endoscopic images include mucosal tissues, bubbles, debris, and the hole of the lumen. The abnormal frames might include additional semantics corresponding to lesions or blood. Based on this observation, in (Vasilakakis et al. 2018) we investigated various multi-label classification methods, aiming to a richer semantic interpretation of the endoscopic images. Two novel methodologies were presented, which include an image-saliency enabled bag-of-words approach and a multilabel CNN architecture enabling multi-scale feature extraction (MM-CNN). The weakly-supervised learning is achieved by using only semantically labeled data, *i.e.*, meaningful keywords describing the image, which greatly reduce the time spent on the demanding pixelwise annotation of the training images. The experiments conducted on the publicly available KID (Koulaouzidis et al. 2017) WCE image dataset, show that the weakly-supervised multi-label classification can provide enhanced discrimination of the GI tract abnormalities, with MM-CNN method to provide the best performance.

The MM-CNN architecture is considered a pre-accessor of LB-FCN (Diamantis et al. 2019) architecture, as it employees multi-scale feature extraction in order to obtain a feature rich representation of the weakly annotated data. The network architecture is illustrated in **(Figure 4.9)**. Instead of the convention softmax output layer, MM-CNN uses sigmoid neurons which enables multi-label image classification.



(a)



(b)

**Figure 4.9** The proposed MM-CNN architecture, composed of LMSCBs. (a) The architecture of an LMSCB. The input volume is forwarded to the multi-scale feature extraction component and then to the addition operator. The final feature maps are then forwarded to the pooling component which results in a 50% dimensionality reduction. (b) The overall MM-CNN architecture composed of 5 LMSCB modules and 4 sigmoid output neurons, which are used for the multi-label classification.

## 4.3.1 Experiments and Results

Experiments were conducted to evaluate the effectivity of multi-label classification in the context of semantic interpretation of endoscopy video frames. The performance of both presented methodologies, MM-CNN and saliency-enabled BoW, were compared with state-of-the-art approaches.

In the case of the saliency-enabled BoW methodology, for each video frame, features have been extracted using the proposed Difference of Maxima (DoM) salient point detection method and the "naive" approach of dense feature extraction. For the proposed DoM salient point detection method we used image samples of 24×24 pixels. The BoW model was constructed with a visual vocabulary with sizes in the range from 500 to 2000 words using the k-means clustering algorithm (Drake & Hamerly 2012). The classification of the feature vectors obtained using the BoW method, was implemented by an SVM classifier. We have tested linear, polynomial and Radial Basis Function (RBF) kernels, and followed the grid search approach (Chang 2011) to determine its optimal parameters. The RBF kernel provided the best results, for a minimum cost parameter c=10 and γ=2−8.

In the case of the MM-CNN the training of the network was performed using the back-propagation algorithm with a batch size of 32 images and the root mean square propagation (RMSProp) (Hinton et al. 2012) optimizer with learning rate l = 0.0001 and fuzz factor **ε** = 1e − 8. Furthermore, video frames from the KID dataset 2 have been cropped to 320 × 320 pixels by removing the excess surrounding black border. Then, they were downsized to a resolution of 256 × 256 pixels. The network has been implemented the Keras (Gulli & Pal 2017) Python library backed by TensorFlow (Abadi et al. 2016) graph framework. It was trained using an NVIDIA GTX-960 enabled graphical processing unit (GPU) with 1024 CUDA (Nickolls et al. 2008) cores having 2 GB of RAM and clock frequency of 1127 MHz. It is worth mentioning that the entire training of the network for each fold took approximately 8 h. The early stopping technique was adopted to optimize the network's generalization performance, using 15% of the data as validation subset. The number of training epochs required per fold was approximately 2000. This could be considered as being relatively low when compared to other networks, *e.g.*, the one of (Simonyan & Zisserman 2014). Yet, it happens due to the low number of free-parameters of the overall architecture **(Figure 4.9)**.

To compare the classification performance of MM-CNN with the transfer learning approach in multi-label classification of WCE gastrointestinal tract images, we implemented the methodology followed by (Zhang et al. 2016). More specifically, for the feature extraction we followed the same procedure as presented by the authors, while for the classification of the extracted features, we followed multilabel "one-vs-all" SVM with c = 2−9 and polynomial kernel. The parameters of the SVM were selected after a series of experiments in order to determine the optimal values for the domain.

The classification performance was thoroughly investigated using receiver operating characteristic (ROC) analysis (Fawcett 2006). Experiments were performed using the 10-fold cross validation evaluation scheme, using SVMs as a binary classifier. Multi-label classification was implemented using a derivative of WEKA library (Garner & others 1995) called MEKA (Read et al. 2016).

Initially, we examined the case of binary classification of the video frames into normal and abnormal classes. We investigate the performance of BoW method using the proposed DoM for salient point detection in comparison with the state-of-the-art methodologies, (Yuan et al. 2015) in which SIFT algorithm (Lowe 2004) was used for the detection of interest points and a concatenated feature vector of SIFT and LBP, or SIFT and CLBP (Yuan et al. 2015), for the description of video frame regions. The comparison also includes the method proposed in (Vasilakakis et al. 2016), which use the SURF (Bay et al. 2006) algorithm for salient point detection in the a-channel (SURF(a)) of CIE-Lab and the dense BoW approach and the CNN (Zhang et al. 2016). The results with regards to lesion detection are presented in **Table 4.3**. It can be noticed that the use of the proposed DoM algorithm increases the binary classification performance to an AUC of 0.81%. All methods provide a low sensitivity. This indicates the difficulty of the lesion detection problem. The BoW method using DoM provided significantly higher specificity (less false positives) than all other methods. The higher sensitivity was obtained by CNN, at the cost of a higher false positive rate.

**Table 4.4** Comparative binary classification results, using various weakly supervised BoW methods with SVM classifier and CNN method. The sensitivity, specificity, AUC and the confusion matrix (True Positives – TP, False Negatives – FN, False Positives – FP, and True Negatives – TN) of each method is included.

| Methods | TP | FN | FP | TN | Sensitivity | Specificity | AUC |
|---|---|---|---|---|---|---|---|
| BoW+SURF(a) | 23 | 34 | 22 | 156 | 0.40 | 0.87 | 0.78 |
| BoW+SIFT+LBP | 17 | 40 | 21 | 157 | 0.30 | 0.88 | 0.72 |
| BoW+SIFT+CLBP | 21 | 36 | 20 | 158 | 0.36 | 0.88 | 0.78 |
| BoW+Dense | 24 | 33 | 18 | 160 | 0.42 | 0.89 | 0.8 |
| CNN | 30 | 27 | 27 | 151 | 0.52 | 0.85 | 0.78 |
| **BoW+DoM** | **25** | **32** | **17** | **161** | **0.44** | **0.91** | **0.81** |

Multi-label classification was performed using the following labels: abnormal, debris, bubbles, and lumen hole. Indicative images from the KID dataset for each label are included in **(Figure 4.10)**. The use of DoM for multi-label classification, results in an even higher classification performance than the conventional binary classification scheme. Best results were obtained using the multi-layer perceptron (MLP) multi-label classification method with 100 hidden layer neurons, a learning rate of 0.1, trained with the features extracted from BoW model. The obtained AUC reached up to 0.83% using a vocabulary of 800 visual words. The results for all weakly methods using BoW features are presented in **(Figure 4.11)**. The basic methods for multi-label classification, which used, were binary relevance (BR), label combination (LC), ranking and thresholding (RT) and pairwise classification (PC). For all multi label methods we used the same SVM with radial basis function kernel (RBF) with c=10. As in the binary classification experiments, these parameters were determined using the afore-mentioned kernels and grid-search approach. Also, **(Figure 4.11)** includes the results of CNN (Zhang et al. 2016) for multi-label classification in order to compare our proposed MM-CNN. It can be noticed that MM-CNN provided the highest performance compared to all the other approaches and achieved an AUC equal to 0.90%.

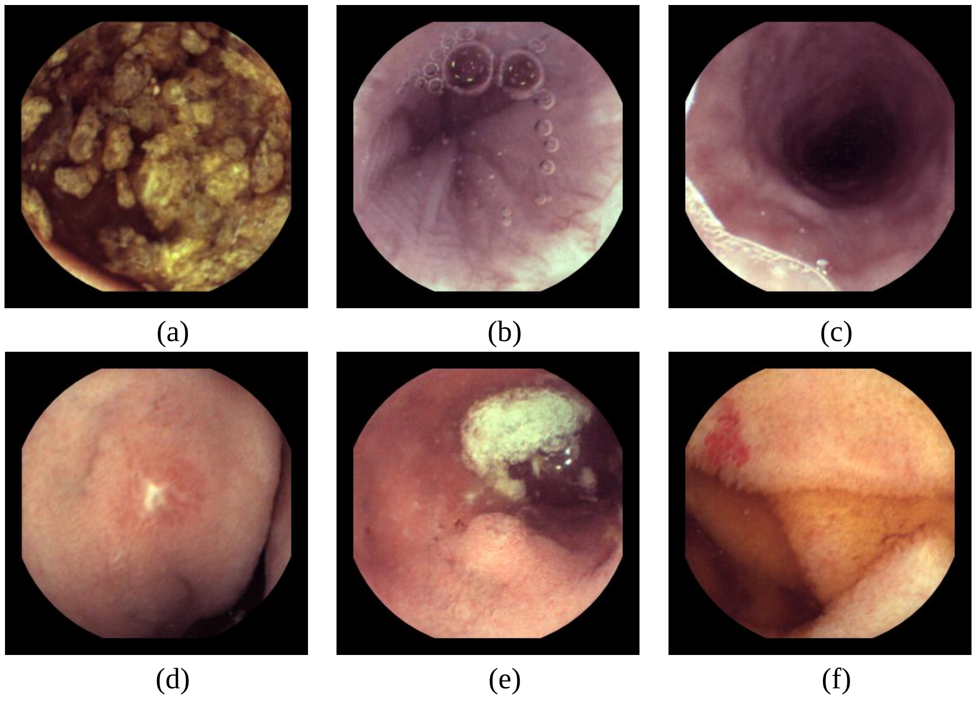


**Figure 4.10** Sample images from the KID dataset. (a) Debris, (b) Bubbles, (c) Lumen hole, (d) Inflammation, (e) Polypoid and (f) Angiectasia.

The classification results per semantic category are presented in **(Figure 4.12)**. It can be noticed that the result for debris are significantly higher than the results of bubbles and lumen hole. The reason is that the most video frames in KID dataset had debris as content compared to the number of video frames that had bubbles and/or lumen hole.

It can also be noticed that the classification performance of the CNN is not always better than the BoW-based approaches, although it has been proved effective in the context of endoscopy (Zhang et al. 2016). This could be explained by the diversity of the KID dataset, which includes several different kinds of lesions, whereas the dataset used in (Zhang et al. 2016) included only colorectal polyps.

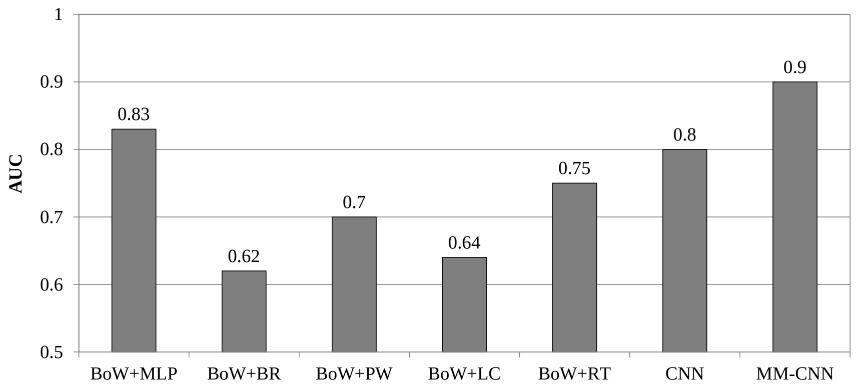


**Figure 4.11** Comparative multi-label lesion detection results for each multi-label method tested.

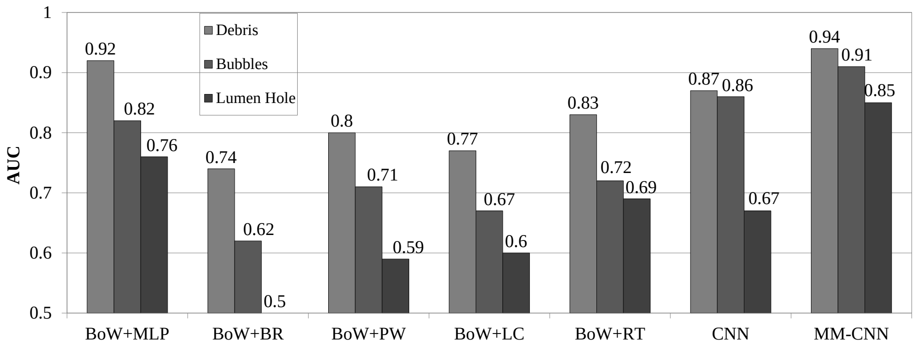


**Figure 4.12** Comparative classification performance results for each semantic label in the KID dataset, for each multi-label method.

## 4.4 Substitution of Real with Artificially Generated Endoscopic Images for CNN Training

Over the years, the frequency of Gastrointestinal (GI) tract diseases is increasing (Anon 2018). This is even more apparent in developed countries. A typical examination of the GI tract is Flexible Endoscopy (Muller & Sonnenberg 1995) and its variations. A non-invasive screening technique, called Wireless Capsule Endoscopy (Swain 2008) (WCE) is becoming increasingly popular, as the examination is performed using an ingestible capsule camera, which can capture images throughout the GI tract of the patient. The retrieved images are then examined by a clinician, in a labor-intense process which typically requires 45-90 minutes of video reading. As a result, computer-aided GI tract lesion detection can be employed to simplify the diagnosis process, thus minimizing the possibility of human-error linked clinician weariness.

One of prime indications for performing WCE is the diagnosis or topographic mapping of GI lesions in known and/or suspected Inflammatory Bowel Disease (IBD) (Koulaouzidis et al. 2013).

The most common inflammatory lesions are ulcers, aphthae, mucosal breaks with surrounding erythema, cobblestone mucosa, stenoses and/or fibrotic strictures, and significant mucosal/villous oedema. State-of-the-art CNNs have been developed and applied for abnormality detection (including inflammatory lesions) in WCE achieving a remarkable performance (Iakovidis et al. 2018)(Diamantis et al. 2019). Earlier approaches specifically for inflammatory lesion detection include methods based on handcrafted color features and Bag-of-visual-Words (BoW) with Support Vector Machine (SVM) classifiers (Vasilakakis et al. 2016).

It is known  that the generalization performance of neural networks is linked with the number and variations of samples available for their training (Neyshabur et al. 2017). This is more apparent in CNN architectures mainly due to the large number of free parameters that need to be trained. To cope with this problem, data augmentation techniques such as, scaling and rotation, is typically employed in order to artificially increase the number of samples found in a dataset. While this can enhance the generalization performance of a CNN, the enhancement is limited as the augmented samples are similar to each other. In most applications this can be tolerated as data availability is not a problem. On the other hand, in the medical domain, mainly due to the data privacy regulations, *e.g.*, the General Data Protection Regulation (GDPR) (Voigt & Bussche 2017), obtaining real medical images as training samples, is becoming harder over the years. In addition, existing accessible datasets are often inadequate for use in the training of deep learning applications, mainly due to their limited size and lack of expert annotations (Guibas et al. 2017). The aforementioned issues could be sidestepped with the usage of synthetic data, since those cannot be traced back to patients and can also be produced in abundance. For these reasons, data generation and more specifically image synthesis has been extensively researched.

To cope with the problem of data availability in the medical domain a novel approach is presented. More specifically our methodology employs a combination of a state-of-the-art LB-FCN (Diamantis et al. 2019), as it has been recently evolved into a lightweight, more efficient, classifier (LB-FCN *light*) (Diamantis et al. 2019), which will be discussed in detail in Section 5 of this thesis, and a Generational Adversarial Network (GAN) (Goodfellow et al. 2014) for training data generation. The GAN was trained to perform non-stationary texture synthesis (Zhou et al. 2018), to generate small bowel WCE images, with and without inflammatory lesions. The artificially generated images were then used to train the LB-FCN *light* architecture. LB-FCN *light* architecture was selected as it combines multi-scale feature extraction and significantly low number of free parameters. We then evaluated the performance of the trained model on real images from the KID WCE image (Koulaouzidis et al. 2017) dataset in the context of inflammatory lesion detection.

## 4.4.1 Medical Image Generation

The goal of texture synthesis is the generation of new samples perceptually similar to an input texture. Conventional approaches to texture synthesis fall into two categories. Non-parametric methods include pixel-based methods (Efros & Leung 1999), which synthesize a new image pixel by pixel. The value of each pixel in the output is being determined by its neighborhood. Pixel-based methods can be improved by replacing pixels with patches as the synthesis unit (Wei et al. 2009). Until recently (Barnes & Zhang 2017), patch-based methods were widely used. Non-parametric techniques for texture synthesis offer the ability to produce high quality results. However, they cannot learn the underlying model of the data since they essentially rearrange the input image based on local similarity criteria (Jetchev et al. 2016). Thus, they are unable to reproduce large-scale structures, including those found in medical images. Another texture synthesis approach is based on the extraction of statistical descriptors to model a texture. A new texture is synthesized by finding an image with matching descriptors such as by optimizing a Gaussian white noise image (Portilla & Simoncelli 2000). While it yields good results on some texture types, in most cases fails to cope with highly inhomogeneous textures.

In recent years, deep learning-based texture synthesis approaches have gained popularity. As opposed to conventional approaches, deep learning methods are capable of discovering models that describe the complex natural world, without the need of hand-crafted features (LeCun et al. 2015). The method presented in (Gatys et al. 2015) is the first to use a deep neural network for texture synthesis. It utilizes a CNN to capture an input texture's spatial statistics by taking advantage of the CNN's powerful feature space. Follow-up works (Ulyanov et al. 2016; Johnson et al. 2016) improve the approach of (Gatys et al. 2015) in terms of speed.

Since the conception of GANs, several variations have been developed (Mirza & Osindero 2014; Odena et al. 2017; Odena 2016; Denton et al. 2015; Radford et al. 2015) able to produce high-quality, natural looking images that can be mistaken for real ones when assessed by human observers. In (Isola et al. 2017) the effectiveness of GANs for image-to-image translation tasks is demonstrated. In the field of medical image generation, GANs have been successfully employed for a variety of tasks, such as the generation of computer tomography (CT) images from their corresponding magnetic resonance (MR) images (Nie et al. 2016), the transformation between different MR image modalities (Nie et al. 2018) and the prediction of PET images from abdominal CT scans for highlighting liver tumors (Ben-Cohen et al. 2017). Similarly a GAN was used in (Calimeri et al. 2017) to generate MR slices of the human brain. To assess the effectiveness, the authors used quantitative and human-based human-based evaluations of generated images. In the context of Cardiac Magnetic Resonance (CMR) image generation, (Zhang et al. 2017) proposed a GAN named Semi-Coupled GAN (SCGAN). The authors proposed a semi-supervised framework to identify CMR images with incomplete Left Ventricle (LV), aiming to ease the process of manual, identification process which is a relatively time-consuming task. The two-stage framework consists of the SCGAN which, generates samples used to extract high-level features from the CMR images. The features are then used to detect missing basal and apical slices. Realistic brain MR image generation using a DCGAN (Radford et al. 2015), was also proposed in the work of (Bermudez et al. 2018). (Beers et al. 2018) used PGGANs (Karras et al. 2017) to synthesize high resolution medical images. The authors evaluated the performance in two domains; fundus photographs exhibiting vascular pathology associated with retinopathy of prematurity (ROP), and multi-modal MR images of glioma. A PGGAN was also used in the work of (Bowles et al. 2018), yet the generated images were used a data augmentation technique, in an effort to mitigate the medical image data scarcity problem. Similarly, recently, (Kaur et al. 2021) used a DCGAN to generate brain MR images that were later used to increase the training data and thus the classification performance of a CNN architecture.

Image-to-image translation has also been used to map binary retinal vessel trees, reconstructed with an adversarial autoencoder (Makhzani et al. 2015), to photorealistic RGB retina fundi images (Costa et al. 2018). In (Guibas et al. 2017), a two-stage pipeline is presented, in which a GAN produces the segmentation masks for retina fundi images, while a second GAN learns the transformation between the binary masks and the fully-colored images. In the work of (Shin et al. 2018), a conditional GAN is presented, able generate colon polyp images to improve polyp detection performance. GANs have also been used for synthetic segmentation images of the lungs and heart in chest X-ray scans (Dai et al. 2017). The work of (C. Han et al. 2018) adopts the DCGAN and WGAN (Arjovsky et al. 2017) variations to create realistic brain MR images. In (Frid-Adar et al. 2018), a DCGAN and an ACGAN (Odena et al. 2017) are trained to produce synthetic liver lesions, which are then used for data augmentation in order to improve the performance of a CNN trained on liver lesion classification. Recently, (Marzullo et al. 2021), used a Pix2Pix GAN(Isola et al. 2017) to perform image-domain translations in order to synthesize realistic laparoscopic images.

While GANs have been used to increase the available samples in the training datasets as a method to increase the generalization performance of CNNs, to the best of our knowledge, no work has been done to investigate their generalization performance when the training dataset consists of only generated images. In this work we investigate the generalization capabilities of state-of-the-art CNN (Diamantis et al. 2019) architecture when trained solely with generated WCE GI tract images on the problem of inflammatory conditions detection in real images, the results of which are promising.

## 4.4.2 Proposed Methodology

The proposed methodology is based on two components; the classifier and the data generator. The first component is a state-of-the-art lightweight CNN architecture, named LB-FCN *light* (Diamantis et al. 2019) which is discussed in detail in Section 5. The second component of our methodology uses a GAN, which performs non-stationary texture synthesis (Zhou et al. 2018). Its architecture is illustrated in **(Figure 4.13)**. As input, it uses sample patches from the available images and artificially generates new images that follow the input textural pattern. Non-stationary texture synthesis was selected because the generated images mimic well, the non-stationary patterns that appear in GI tract images. More specifically (Zhou et al. 2018), generalizes the original GAN architecture (Goodfellow et al. 2014) by having a fully convolutional generator to learn common patterns from a $k \times k$ block, randomly sampled from the input image, and produce a $2k \times 2k$ image, instead of learning the mapping from a simple uniform distribution to the image space. The resulting image is perceptually similar to a target block of the same size, also cropped from the input image. The output of the generator, along with the real $2k \times 2k$ image, are then provided as input to the discriminator which learns to recognize whether a $2k \times 2k$ image is real or fake. Once trained, the generator can be applied to an image of arbitrary size, effectively synthesizing a new image. The GAN uses a linear combination of adversarial loss $L_{adv}$ (Goodfellow et al. 2014), $L_1$ loss (Janocha & Czarnecki 2017) and style loss $L_{style}$ (Gatys et al. 2015) to optimize the generator Eq. (4.6).

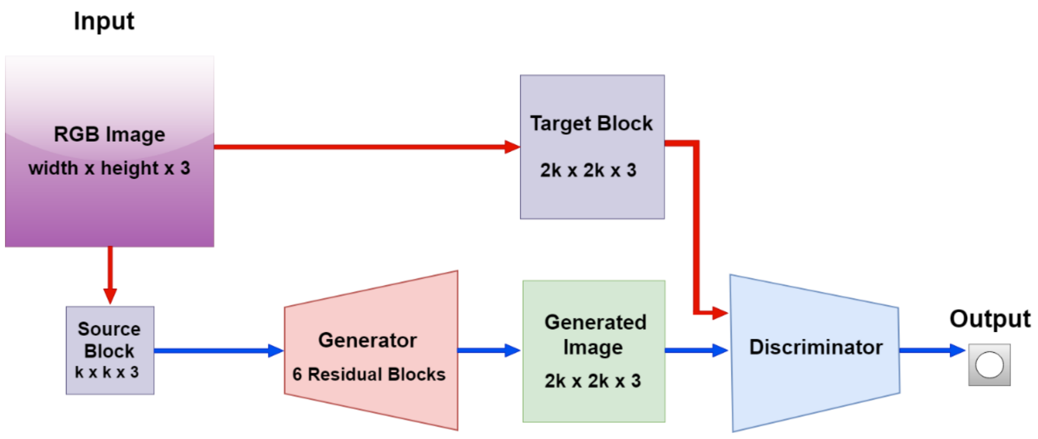


**Figure 4.13** Non-stationary texture synthesis (Zhou et al. 2018) GAN architecture. The generator receives an input of k×k×3 size and expands it to 2k×2k ×3. The discriminator receives an input 2k×2k×3 and tries to identify the validity of it.

$$L_{total} = L_{adv} + \lambda_1 + L_1 + \lambda_2 + L_2 \tag{4.6}$$

where $\lambda_1$ are $\lambda_2$ are constants with values 100 and 1 respectively, as recommended by the authors (Zhou et al. 2018). The output is primarily affected by the adversarial loss, while $L_1$ and style loss help reduce artifacts.

### 4.4.3 Evaluation Methodology and Results

The performance of the proposed methodology was evaluated in the context of WCE image inflammatory condition detection in the small bowel of the GI tract. For this reason we used the publicly available KID (Koulaouzidis et al. 2017) Database 2. In total, the database contains 1778 normal images and 574 images containing various abnormalities such as, polypoids, vascular and inflammatory conditions. The size and variations of images that appear in the database, made it an appropriate choice to serve as a baseline for our experiments. For the purpose of our experiment, we selected a subset of the KID dataset which contains normal images from the small bowel (728 images) of the GI tract along with images that contain inflammatory lesions (227 images).

In order to diversify the output from a single input, during testing 6 random image patches are selected from the original input, which are then used as the input for the generator. Output from the trained generator is illustrated in (**Figures 4.15, 4.16)**. For our experiments, we trained the adversarial expansion GAN on 728 non-pathologic images from the small bowel and 227 images with GI tract inflammatory conditions. We then randomly chose one of the 6 alternative results to be used in the training of LB-FCN *light*.

To investigate the effect of training with fake images, artificially generated by the GAN, we evaluated the classification performance of LB-FCN *light* architecture in two experiments. In the first experiment we trained the model with fake images, and evaluated its performance on real images, while on the second experiment, we trained and tested the model with real images. To limit the bias, in both experiments stratified 10-fold cross-validation technique was employed. In this procedure the dataset is split into 10 disjoined subsets from which 9 are used for training and 1 is kept for testing. The process is then repeated 10 times, each time keeping a different subset for testing. To assure that both of our experiments are comparable, the same image subsets were used upon testing. To assess and visualize the results of our experiments we used the Receiver Operating Characteristic (ROC) curves, which represent the tradeoff between True Positive (TPR) and False Positive (FPR) Rates under different decision thresholds. To measure the classification performance of both of our experiments, we choose to use the Area Under ROC (AUC) (Fawcett 2006), as it is insensitive to the class imbalance that was present to our dataset.

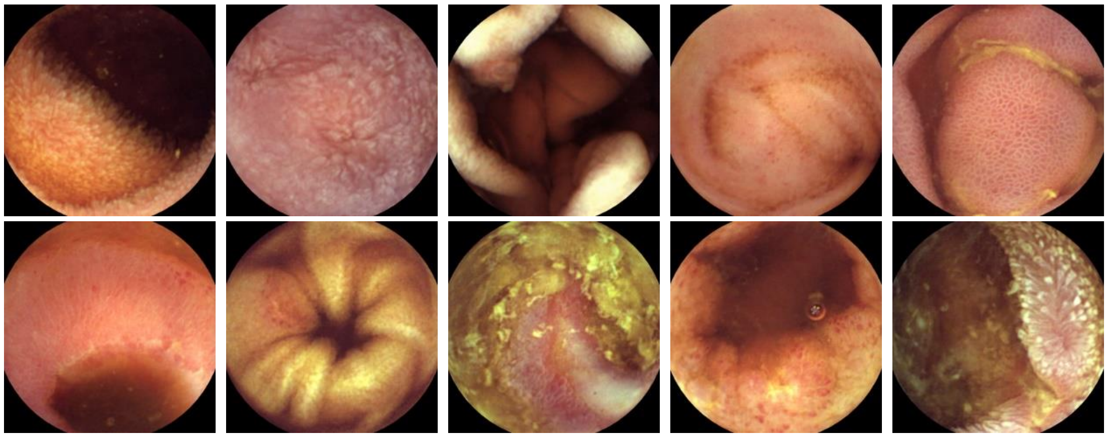


**Figure 4.14** Sample images from the KID Database. The first row contains healthy small bowel images. The second row contains images of various inflammatory conditions.

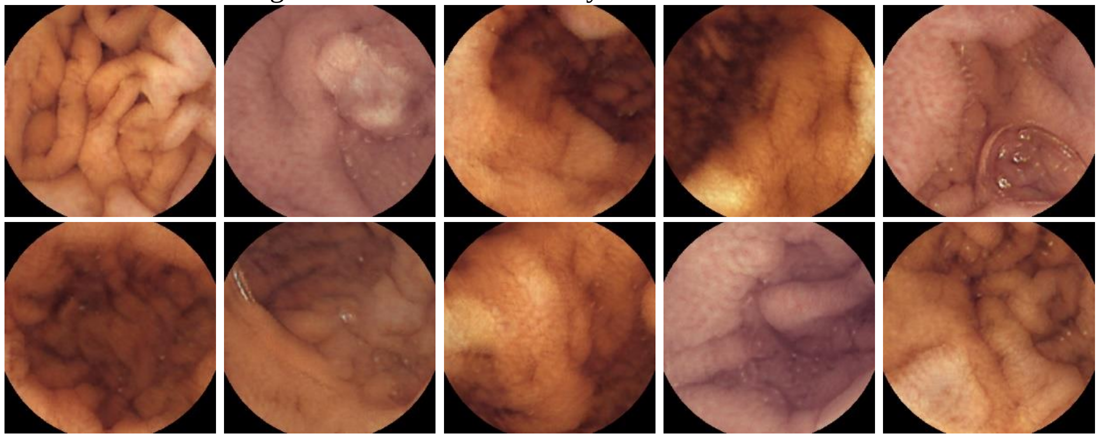


**Figure 4.15** Sample generated small bowel images using the non- stationary texture synthesis GAN (Zhou et al. 2018).

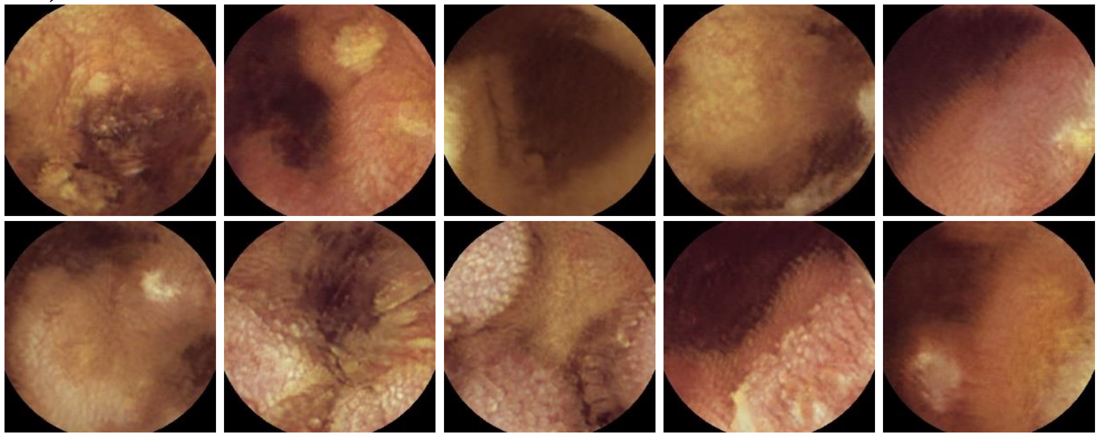


**Figure 4.16** Sample generated images with inflammatory conditions using the non-stationary texture synthesis GAN (Zhou et al. 2018).

The LB-FCN *light* architecture was implemented using the popular open source Python library Keras (Gulli & Pal 2017) with TensorFlow (Abadi et al. 2016) framework backend. The training was conducted using two high-end GPUs (GTX-1080 TI) with each one having 3584 CUDA (Nickolls et al. 2008) cores. To prevent the network from overfitting, the early stopping technique was employed. The training of (Zhou et al. 2018) GAN, was also conducted using the same equipment and implemented using the open source python framework Pytorch (Paszke et al. 2019).

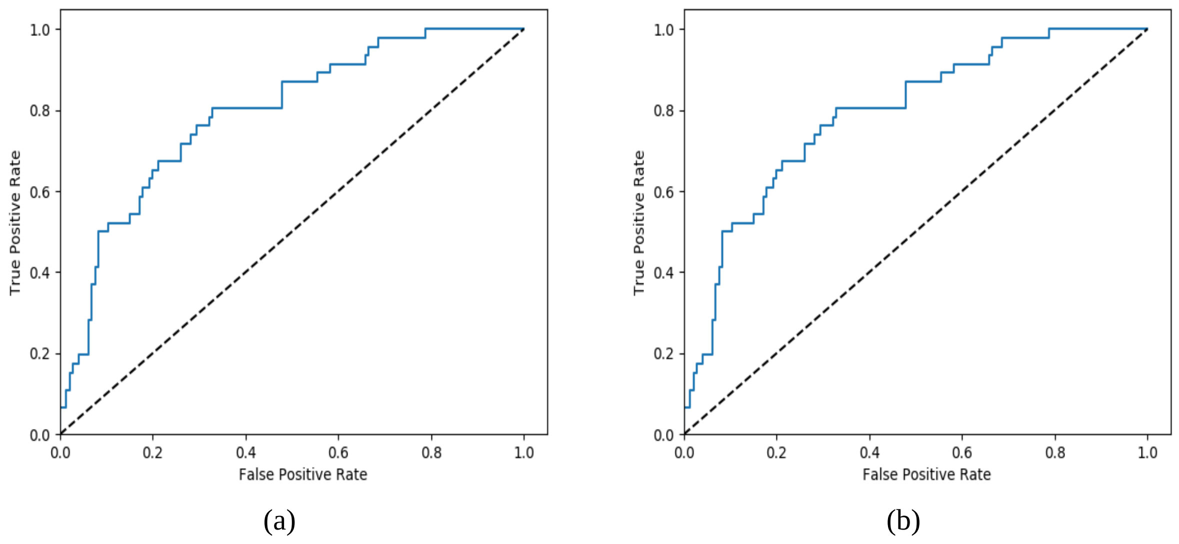


**Figure 4. 17** ROC obtained by 10-fold Cross-Validation on LB-FCN light architecture trained using (a) artificially generated images and (b) real images.

Training the LB-FCN *light* architecture using generated images and evaluating on real images resulted into a 79.1% AUC **(Figure 4.17a)** while training and testing using real images resulted into 90.9% AUC **(Figure 4.17b)**. It is clear that the classification performance of the LB-FCN in the first case is lower. However, by comparing it to the results of other recent approaches on the same dataset, it can be considered comparable. In (Vasilakakis et al. 2016) the BoW-based methodology using features extracted from the CIE-Lab color space, resulted in a performance ranging from 77% to 81%. Therefore, the results obtained can be considered as promising, since the proposed approach is only a first attempt to tackle the presented problem.

We believe that the lower AUC of LB-FCN *light* trained using generated images can be attributed to the lower quality of the generated images with inflammatory conditions **(Figure 4.16)**. More specifically we notice that although the non-stationary texture synthesis GAN (Zhou et al. 2018) was able to capture the small bowel texture and produce high quality real-like images **(Figure**

**4.15),** when inflammatory conditions were present, the generated images were of lower quality. From a medical viewpoint, the main problem affecting their quality is the clarity of pathology; although the fake images with normal content are generally satisfactory, the fake images including pathologies look like taken from a procedure with either unclear/semiopaque luminal content or at least non-translucent. We believe that this is due to the nature of the inflammatory conditions, as in some cases, the conditions are so severe that can harm the uniformity of the overall small bowel texture and thus affecting the overall performance of the generator.

# CHAPTER 5

# MACHINE LEARNING FOR COMPUTER ASSISTED NAVIGATION

This chapter presents the novel contributions of this study towards the use of ML in the context of assistive navigation systems. In the first section it illustrates the Lightweight Look-Behind Fully Convolutional Neural etwork (LB-FCN *light*) which is an extension of the LB-FCN architecture, designed to work in applications that require high inference speeds running on low-end devices, such as mobile devices and embedded systems. Applications of the LB-FCN *light* architecture include staircase detection in outdoor environments (Diamantis et al. 2019) and obstacle recognition in the context of obstacle avoidance for the navigation of visually impaired individuals (Dimas et al. 2020). The second section of this chapter includes a novel digital twin framework for the simulation and evaluation of assistive navigation systems, and its application in the context of a camera-based wearable system for visually impaired individuals in an outdoor cultural space.

## 5.1 Introduction

Today, visual impairment (of any form) affects approximately 16% of the world's population (WHO 2018). The affected individuals deal with various daily challenges, struggling to fit in the modern society. To address this problem, researchers in the fields of medicine, smart electronics and computer science are joining forces to create assistive systems for visually impaired individuals. To date, several designs and components of wearable, camera-enabled systems have been proposed. Recently, we presented a novel solution to this problem that can evolve into an everyday visual aid for people with limited sight or total blindness (Iakovidis et al. 2020). This dissertation has contributed in that investigation, and the solution is now integrated into a first prototype of a wearable smart-glasses system for the visually impaired. The system is equipped with RGB-D cameras, it incorporates efficient deep learning and uncertainty-aware decision-making algorithms, interprets the video scenes, translates them into speech, and describes them to the user through audio.

One of the key components of any assistive navigation system is the detection and recognition of objects and scenes. Due to the advancements of CNNs in the field of CV (Section 2), most modern navigation assistive systems include a deep learning based intelligent module.  An integrated CNN-based framework for object detection was presented in (Sermanet et al. 2013). That framework combined a CNN architecture for feature extraction based on AlexNet (Krizhevsky et al. 2012), named OverFeat, and a regression network to detect multiple bounding boxes around objects in images. A Region-based CNN architecture for object detection was presented in (Girshick et al. 2014) with the name R-CNN. The methodology uses selective search (Uijlings et al. 2013) to extract 2.000 class-agnostic region proposals from each image, which are then resized and feed-forwarded into a pre-trained CNN model to extract features. The extracted features are then used to train a linear Support Vector Machine (SVM) classifier (Theodoridis & Koutroumbas 2009) which classifies the extracted feature representations. Although R-CNN outperformed the OverFeat approach (Sermanet et al. 2013) for object detection, it requires more computational resources. To reduce its computational complexity, the Fast R-CNN (Girshick 2015) was proposed, in which feature maps are extracted from the entire input image. From these feature maps, region proposals are extracted and reshaped into a fixed size, by a technique called Region of Interest (RoI) pooling, so that they can be processed by a fully connected layer. The Softmax function is used to predict the class of the RoI vector while in parallel it computes the offset values for the bounding box of the object.

Another architecture, called Spatial Pyramid Pooling Network (SPPNet) (He et al. 2015b) aimed to cope with the problem of the fixed-size input required by the CNNs which may impact the detection accuracy of the overall model. This was done by implementing a novel spatial pyramid pooling which enabled the network to generate fixed-length image representation regardless of the image size. Compared with R-CNN, SPPNet relies on the same principles, yet it does not have to process 2.000 region proposals per image, as R-CNN does. Each bounding box is classified by an SVM and bounding box regressor. A Faster R-CNN (Ren et al. 2015) achieved real-time object detection capabilities, by removing the selective search used by the previous methodologies.

A methodology for object detection that is fundamentally different from the previous ones was presented in (Redmon et al. 2016). It is called You Only Look Once (YOLO) and it relies solely on a single forward pass of an input image. The image is subdivided using a fixed-size grid, and entered to a CNN that predicts bounding boxes and class probabilities for each box. A saliency-inspired neural network model for object detection was proposed in (Erhan et al. 2014). It predicts a set of class-agnostic bounding boxes along with a single score for each box, corresponding to its likelihood of containing any object of interest. In (W. Liu et al. 2016) an object detector with name Single Shot multibox Detector (SSD) which achieved good balance between computational performance and prediction accuracy. A region-based, Fully Convolutional Network (FCN: a CNN without fully-connected layers) was proposed in (Dai et al. 2016). It relies on the generation of position-sensitive score maps to cope with the dilemma between translation-invariance in image classification and translation-variance in object detection. In (Lin et al. 2017) an object detector for multi-scale object detection was proposed. That detector relies on a feature extractor, named Feature Pyramid Network (FPN), which was designed to improve detection accuracy and speed. In (Redmon & Farhadi 2017) YOLO9000, an extension of the YOLO approach (Redmon et al. 2016), was introduced for real-time object detection, considering 9000 object categories. Newer incremental improvements of the original YOLO architecture include YOLO-v3 (Redmon & Farhadi 2018) and recently YOLO-v4 (Bochkovskiy et al. 2020).  A single-shot object detector, named Deconvolutional SSD (DSSD), was presented in (Fu et al. 2017). It extended SSD by replacing the original VGGNet with a Residual Network (ResNet) (He et al. 2016) for feature extraction. RetinaNet, proposed in (T.-Y. Lin et al. 2018), is a single, unified network composed of a backbone network and two task-specific sub-networks. The backbone network is implemented by a ResNet architecture, used for feature extraction. The first sub-network, performs the classification and the second one, performs bounding box regression. A multi-scale extension of the DSSD network, called Multi-Scale Deconvolutional SSD (MDSSD), has been proposed in (Cui 2018), specifically for small object detection.

The backbone network of single shot detectors heavily affects their computational and classification performance. This is expected, since both the detection and classification steps are based on the quality of the features being extracted from the backbone network. Towards this direction, we presented LB-FCN *light* (Diamantis et al. 2019) which aims to create features of various scales, using multi-scale feature extraction and, reduce the computational complexity, speeding up detector performance, using depthwise separable convolutions. In Section 5.2 the network architecture is presented along with use-cases, benchmarking the network over state-of-the-art.

## 5.2 The Lightweight Look-Behind Fully Convolutional Neural Network Architecture

The Lightweight Look-Behind Fully Convolutional neural network (LB-FCN *light*) (Diamantis et al. 2019) was originally proposed in the context of staircase detection in outdoor environments. The aim of the architecture is to reduce the computational complexity that is typically found in conventional deep CNNs, such as (Simonyan & Zisserman 2014), which suffer from high computational complexity mainly due to their large number of free parameters. As a result, high-end computational equipment such as Graphical Processing Units (GPUs) is needed for both training and testing time, limiting their use in indoor workstations.

Studies such as (Iandola et al. 2016),(Howard et al. 2017; Sandler et al. 2018; Howard et al. 2019) focus their interest in computational complexity reduction of CNN architectures, aiming to enable their usage in mobile and embedded devices. In this context, the tradeoff between computational efficiency and detection performance has been investigated by (Sandler et al. 2018), resulting in a state-of-the-art architecture called MobileNet-v2, extending the original MobileNet-v1 proposed in (Howard et al. 2017). More specifically this architecture keeps the basic principles of depthwise convolutions for the original design enhances it by adding linear bottleneck layers and shortcut connections between each bottleneck. Linear bottleneck layers were utilized as experimental evidence that the non-linear ones were damaging the extracted features between the bottlenecks. As a result of these changes the architecture contains 30% less parameters than MobileNet-v1 while providing a higher accuracy. Recently, we presented LB-FCN (Diamantis et al. 2019) architecture in the context of abnormality detection in medical images. The architecture featured multi-scale feature extraction modules composed of conventional convolutional layers, to better represent the different scales of abnormalities. In addition, look-behind connections were used, which connect the input features to the output of each multi-scale feature extraction module. This was required, so that the high-level features will propagate throughout the network, allowing the network to converge faster and increasing the overall detection accuracy.

The core of LB-FCN *light* architecture is inspired by LB-FCN (Diamantis et al. 2019) and includes modification to enable efficient computations on mobile and embedded devices, while providing a sufficient staircase detection accuracy. More specifically, LB-FCN *light* extends the original LB-FCN design by replacing the multi-scale conventional convolutional layers with depthwise convolutional layers (Chollet 2017). Key features of this architecture include the utilization of multi-scale depthwise separable convolution layers (Chollet 2017) and residual learning (He et al. 2016) connections which help to maintain relatively low number of free parameters, without sacrificing the detection accuracy.

### 5.2.2 The Network Architecture

The design of the LB-FCN *light* architecture follows the FCN (Springenberg et al. 2014) network design, where only convolutional layers are utilized throughout the network. By replacing the fully connected layers, usually found in the classification layer of conventional CNN architectures such as (Krizhevsky et al. 2012; Simonyan & Zisserman 2014), a significant reduction of the number free parameters of the architecture can be achieved. Inspired by the Mobilenet architecture, proposed in (Howard et al. 2017), depthwise separable convolutions (Chollet 2017) are implemented throughout the network to further reduce the complexity of the overall architecture. While in conventional convolution the filters are connected on the entire depth of the input channels, in depthwise separable convolution the filter is applied separately on each channel. To connect the separate filters, the layers are followed by a 1×1 conventional convolution.

The main component of LB-FCN *light* is the Multi-Scale Depthwise Convolution module **(Figure 5.1)** which follows the principles established in (Diamantis et al. 2019). This module is capable of extracting features from parallel depthwise separable convolution layers, each one with a different filter size. More specifically the layers extract features at three different scales: 3×3, 5×5 and 7×7 respectively. The feature maps from each layer are then concatenated forming a multi-scale feature representation of the input which is then followed by 1×1 convolution layer. The architecture features residual connections, which connect the input volume of the multi-scale module using adding operator aggregation with the output of it. This is done in order to preserve the higher-level features extracted from the previous multi-scale blocks throughout the network.

Following the FCN (Springenberg et al. 2014) approach which shows that conventional max pooling operation can be replaced with a convolutional based, we utilized convolutional pooling with filter size 3×3 and stride 2. This introduces another level of non-linearity to the network while keeping the overall architecture logically unified. After each pooling operation the number of extracted filters of each convolutional layer is doubled. In total four multi-scale depthwise convolution modules are utilized in the network with three residual connections as illustrated in **(Figure 5.2)**. For the staircase detection, a softmax layer of two neurons is used as the output of the network.

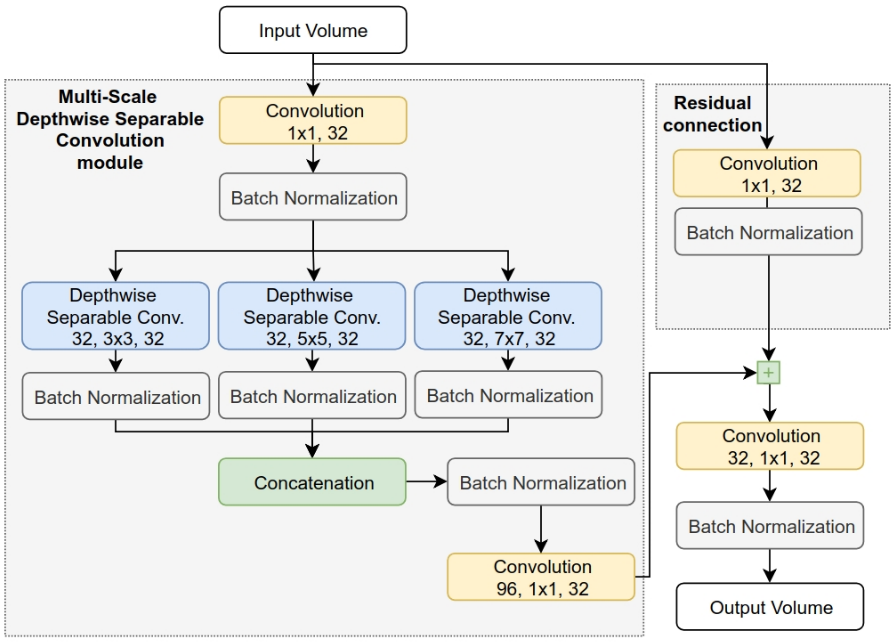


**Figure 5.1** The main building block of LB-FCN *light* architecture.

Throughout the architecture all convolution layers use ReLU activations followed by output batch normalization. The normalization is used so that the output of the convolution layers are centered on zero mean with the unit standard deviation. It has been empirically confirmed that output normalization can contribute to a faster network converge while reducing overfitting phenomenon. As a result of the above no Dropout layer (Srivastava et al. 2014) was used.

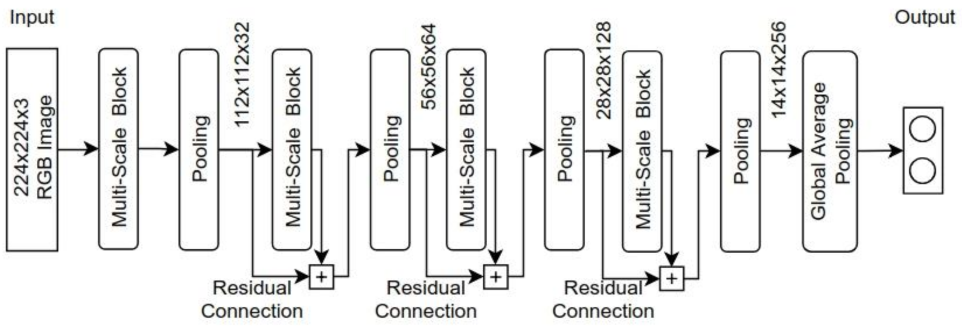


**Figure 5.2** The complete LB-FCN *light* architecture composed of four multi-scale blocks and three residual connections.

While we maintained the multi-scale feature extraction characteristics established in the original LB-FCN (Diamantis et al. 2019) architecture, the change in original filter size selection block increased the overall accuracy of the network. Furthermore, we used conventional ReLU activation functions throughout the network instead of Parametric ReLU that were used in original LB- FCN architecture, which resulted in lower computational complexity without any significant detection performance overhead. The overall improvements made in original LB-FCN architecture, resulted in a significant increase in computational efficiency. As a result, LB-FCN light architecture is capable to efficiently run on mobile and embedded devices.

### 5.2.3 Staircase Detection

Staircase detection in natural images has several applications in the context of robotics and navigation of visually impaired individuals. Previous works are mainly based on handcrafted feature extraction and supervised learning using fully annotated images. In this work we address the problem of staircase detection in weakly labeled natural images, using a novel Fully Convolutional neural Network (FCN), named LB-FCN *light*.

Staircases can be found almost everywhere in different colors, shapes and sizes in both indoor and outdoor environments. Staircases are useful in everyday life; however, they can be seen also as an obstacle for the navigation of humans with disabilities, as well as the navigation of artificial, robotic, agents. The detection of a staircase can be even more difficult in unknown environments, especially for the visually impaired, where there is no previous knowledge about the surroundings, and they can become hazardous. Therefore, staircase detection can be considered as an important component of any system aiming to provide navigational assistance in either indoor or outdoor environments. In controlled, indoor environments, markers, such as augmented reality markers can be used to provide high success rate of staircase detection (Yu et al. 2018). The detection problem usually becomes much harder in outdoor, uncontrolled environments, where different types of staircases of various sizes can be found under various illumination conditions, and can be observed from different viewpoints.

In this chapter we address image-based staircase detection as a pattern recognition problem in the context of embedded and mobile devices. The main challenge is to be able to provide sufficient detection accuracy by utilizing the limited computational resources of such devices, especially in outdoor environments with low latency and limited network accessibility. To address this challenge, we propose a novel lightweight Fully Convolutional neural Network (FCN) architecture as a modification of our recent Look-Behind FCN (LB-FCN) architecture (Diamantis et al. 2019). This novel architecture, named LB-FCN *light*, has significantly fewer free parameters and requires fewer Floating-Point Operations (FLOPs) compared to the previous LB-FCN and state-of-the-art architectures for mobile devices. This was achieved by implementing depthwise separable convolutions throughout the convolutional layers of the network. Also, it enables multi-scale feature extraction and residual learning, making it suitable for multi-scale staircase detection in both indoor and outdoor environments. To evaluate the performance of LB-FCN *light* we created a weakly labeled image dataset, with staircases found in natural images collected from publicly available datasets, *i.e.*, a dataset with semantically labeled images as containing or not containing staircases.

#### 5.2.3.1 Related Work

Staircase detection has been an active research topic in CV and robotics, with an increasing interest nowadays as we are going through the era of ubiquitous computing and pervasive intelligence. One of the first relevant works (Se & Brady 2000) was based on Gabor filters and concurrent line grouping for distant and close staircase detection respectively. In the context of autonomous vehicle navigation, an outdoor descending staircase detection algorithm was presented by (Hesch et al. 2010), based on texture energy, optical flow, and scene geometry features. In the context of computer aided navigation of visually impaired in outdoor environments using a wearable stereo camera, (Lee et al. 2012) utilized Haar features and Adaboost learning providing real-time detection performance. A similar approach that utilizes Haar-like features and an improved staircase specific Viola-Jones (Viola et al. 2001) detector was proposed in (Maohai et al. 2014).

Frequency domain features obtained by ultrasonic sensors were investigated in (Bouhamed et al. 2013), to detect and recognize floor and staircases in electronic white cane. A wearable RGB-D camera mounted on the chest of a visually impaired individual, was used in (Pérez-Yus et al. 2014), where an indoor environment for staircase detection and modeling was proposed. Their approach is capable of providing information for the presence and location along with the number of steps of staircases. Recently an indoor staircase detection framework was proposed in (Ciobanu et al. 2017), utilizing depth images, capable of running on mobile devices. The approach is based on the detection and clustering of image patches that have the surface vectors pointing to the top direction. In addition, information from the Inertial Measurement Unit (IMU) sensor of the device is used to calibrate the surface vectors with the camera orientation. Most of the current staircase detection approaches are supervised, requiring fully annotated training images from controlled environments, *i.e.*, images indicating the location of the staircases within the images. Furthermore, to the best of our knowledge the staircase detection has not been previously investigated to a sufficiently generic extent.

Although deep learning and more specifically CNNs (LeCun et al. 1989) have demonstrated impressive performance in CV applications, especially in natural image classification (Krizhevsky et al. 2012), staircase detection approaches have not been previously reported.

#### 5.2.3.2 The Dataset and Evaluation Methodology

To evaluate the performance of the proposed architecture in the context of natural image staircase detection we have considered two publicly available datasets. The first dataset, named LM+Sun (Tighe & Lazebnik 2010), is a fully annotated natural image dataset obtained from the combination of LabelMe Database (Russell et al. 2008) and SUN dataset (Xiao et al. 2010). The dataset consists of 45,676 images from 232 categories, found in indoor and outdoor environment under various conditions and sizes. For the purpose of our experiment, we used a subset of LM+Sun dataset which includes natural images found in urban and street areas. While the full LM+Sun dataset contains 314 staircase labeled images, most of them are found in indoor environments. Images containing staircases were also found in the urban and street subsets of this dataset, *e.g.*, staircases of buildings that can be directly recognized by a human observer, considering: a) staircases that have at least two steps, and b) staircases covering >15% of the image (in staircases of smaller coverage the steps are not distinguishable; therefore, they cannot be perceived directly as such, without contextual information). To minimize the possibility of a human error in the annotation process, two reviewers separately reviewed and annotated the dataset, and found in total 245 images that include outdoor staircases. To further increase the number of outdoor staircase images, we have created a second dataset named "StairFlickr" which extends LM+Sun staircases with a total of 524 outdoor staircase images. StairFlickr dataset images were obtained from the popular photo management and sharing web application Flickr (Flickr Inc. 2019).

For the purposes of our research, we omitted the fully annotated metadata provided about the staircases in the original LM+Sun dataset. This was performed as our architecture aims for staircase detection on solely weakly-labeled natural images. In total the described dataset includes 5,539 images from which 1,083 images contain staircases[2]. Indicative images from this dataset are

[2] http://enorasi.dib.uth.gr/database/index.html.

illustrated in **(Figure 5.3)**. As it can be observed, the dataset includes various types of staircases found in various positions, sizes, capture from different viewpoints.

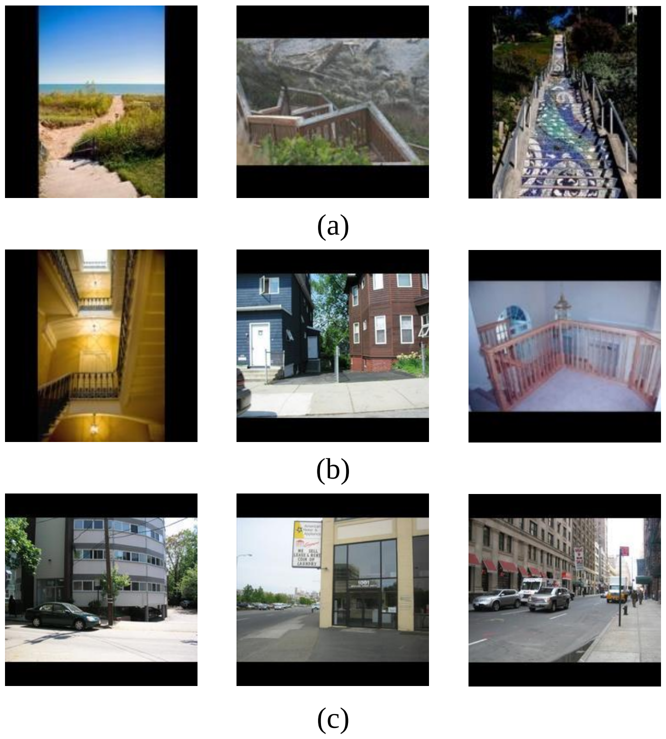


**Figure 5.3** Top: staircases found in StairFlickr dataset. Middle: staircases found in LM+Sun dataset. Bottom: non-staircases images from LM+Sun dataset.

To evaluate the detection performance of the proposed architecture we followed the stratified 10-fold cross-validation (CV) procedure. The dataset was partitioned into 10 stratified subsets from which 9 were used for training and 1 for testing. This was repeated 10 times, each time selecting a different subset, until all folds have been tested. For each evaluation we calculated the accuracy (ACC), specificity (SPC), and sensitivity (TPR) of the trained model.

To better evaluate the classification performance of the trained network, we utilized the Area Under ROC (AUC) measure. AUC measure is a reliable classification performance measure that is insensitive to imbalanced class distributions (Fawcett 2006). This was chosen as the total number of images containing staircases was significantly fewer than the rest of the rest natural images in the dataset.

### 5.2.3.3 Results

We trained the LB-FCN *light* architecture using the images from both Flickr and LM+Sun datasets. As the images differ from each other in both size and aspect ratio we rescaled the dataset to the standardized input size of the network which is 224×224 pixels. To maintain the original aspect ratio of the images, they were padded with zeros to match the network's input dimensions. It is worth mentioning that no further pre-processing step was applied to the images. As the proposed architecture focuses on weakly labeled images, the detailed annotations for the staircases provided by LM+Sun (Tighe & Lazebnik 2010) dataset were ignored. We utilized only the semantic annotations of the images which indicate the presence or absence of staircases.

For the training of the network we utilized the Adam (Kingma & Ba 2014) optimizer with initial learning rate alpha = 0.001 and first and second moment estimates exponential decay rate beta1 = 0.9 and beta2 = 0.999 respectively. For the implementation of the architecture we utilized the Python Keras (Chollet & others 2015) library and the Tensorflow (Abadi et al. 2016) tensor graph framework. The network was trained with mini-batch size of 32 samples on NVIDIA TITAN X GPU, equipped with 3584 CUDA (Sanders & Kandrot 2010) cores, 12 GB of RAM and base clock speed of 1417 MHz. On each fold we utilized the early-stopping technique where a small subset of the training fold was utilized as a validation dataset.

To evaluate the effectiveness in both detection accuracy and computational complexity reduction of LB-FCN *light* architecture we used the MobileNet-v2 (Sandler et al. 2018) as a state-of-the-art architecture for comparison. The results obtained by the two architectures are illustrated in **Table 5.1**.

While the detection performance is slightly higher in case on LB-FCN *light*, the noticeable difference between the two architectures is the computational complexity requirements. **Table 5.2** includes a comparison between the architectures in terms of both the number of trainable free parameters and the total number of required FLOPs. The improvements made on the original LB-FCN design, resulted in a significant reduction of the overall number of FLOPs, from $1.3\times10^{7}$ down to $0.6\times10^{6}$, and reduction of the free parameters of the network, from $8.2\times10^{6}$ down to $0.3\times10^{6}$ respectively.

**Table 5.1** Detection performance comparison, using 10-fold cross-validation, between state-of-the-art MobileNet-v2 (Sandler et al. 2018) and LB-FCN *light* (Diamantis et al. 2019)

| Architecture | AUC (%) | Accuracy (%) | Specificity (%) | Sensitivity (%) |
|---|---|---|---|---|
| LB-FCN *light* | **88.93 ± 1.86** | **91.89 ± 2.12** | **93.80 ± 2.61** | **84.05 ± 3.51** |
| MobileNet-v2 | 87.86 ± 2.11 | 89.99 ± 2.37 | 93.58 ± 2.45 | 83.78 ± 3.22 |

**Table 5.2** Computation complexity comparison between state-of-the- MobileNet-v2 (Sandler et al. 2018) and LB-FCN *light* (Diamantis et al. 2019)

| Architecture | FLOPs ($\times 10^6$) | Trainable Free Parameters ($\times 10^6$) |
|---|---|---|
| LB-FCN *light* | **0.6** | **0.3** |
| MobileNet-v2 | 4.7 | 2.2 |

**Table 5.3** Confusion matrix of LB-FCN *light* (Diamantis et al. 2019)

| | Staircase *actual* | Non-Staircases *actual* |
|---|---|---|
| Staircases *predicted* | 910 | 276 |
| Non-Staircases *predicted* | 173 | 4180 |

### 5.2.4 Obstacle Recognition in the Context of Uncertainty-Aware Visual Perception System for Outdoor Navigation of the Visually Challenged

In the context of Visual Challenged People (VCP) navigation, a novel Visual Perception System (VPS) (Dimas et al. 2020) was proposed for outdoor navigation that can be evolved into an everyday visual aid for VCP. The methodology incorporates deep learning, object recognition models, along with an obstacle detection methodology based on human eye fixation prediction using GANs (Pan et al. 2017) and fuzzy-based risk assessment. The system is integrated in wearable visual perception system and incorporates system architecture for remote task execution for the computationally expensive components of the system.

#### 5.2.4.1 The System Architecture

As the stereoscopic depth aware RGB camera, namely the Intel® RealSense$^{TM}$ D435 was chosen, since it provides all the functionalities needed by the proposed system in a single unit. This component is connected via a USB cable to a BCU of the wearable system. The barebone computer unit (BCU) used in the system was a Raspberry Pi Zero. The BCU orchestrates the communication between the user and the external services that handle the computationally expensive deep learning requirements of the system on a remote cloud computing infrastructure. Another role of the BCU is to handle the linguistic interpretation of the detected objects in the scenery and communicate with the Bluetooth component of the system, which handles the playback operation. For the communication of the BCU component with the cloud computing component, we choose to use a low-end mobile phone that connects to the internet using 4G or Wi-Fi when available, effectively acting as a hotspot device.

For the communication between the BCU and the cloud computing component of the system, we choose to use the Hyper Text Transfer Protocol version 2.0 (HTTP/2), which provides a simple communication protocol, since the messages between the peers are fully encrypted using the SSL/TLS v1.3 protocol. As the entry point of the cloud computing component, we use a load balancer HTTP microservice, which implements a REpresentational State Transfer (RESTful) Application Programming Interface (API) that handles the requests coming from the BCU, placing them in a message queue for processing. The queue follows the Advanced Message Queuing Protocol (AMQP), which enables a platform agnostic message distribution. A set of message consumers, equipped with Graphical Processing Units (GPUs), are processing the messages that are placed in the queue and, based on the result, communicate back to the MPUs using the HTTP protocol. This architecture enables the system to be extensible both in terms of infrastructure, since new works can be added on demand, and in terms of functionality, depending on future needs of the platform.

The VPS component communication is shown in **(Figure 5.4)**. More specifically, the BCU component of the system, receives RGB-D images from the stereoscopic camera at a real-time interval. Each image is then analyzed using fuzzy logic by the object detection component of the system on the BCU itself, performing risk assessment. In parallel, the BCU communicates with the cloud computing component by sending a binary representation of the image to the load balancer, using the VPS RESTful API. A worker then receives the message placed in the queue from the load balancer and performs the object detection task, which involves the computation of the image saliency map from the received images using a GAN. When an object is detected and its boundaries determined, the worker performs the object recognition task using a CNN, the result of which is a class label for each detected object in the image. The worker, using HTTP, informs the MPU about the presence and location of the object in the image along with the detected labels. As a last step, the MPU linguistically translates the object position along with the detected labels provided from the proposed methodology, using the build-in text to speech synthesizer of the BCU. In detail, as an initialization step of the user-system interaction, the system detects and recognizes immediate obstacles found in the scenery, which are communicated to the user. Upon the next iteration, in case of absence of new high-risk obstacles, new notifications are not provided to the user. In the event of detection of a new high-risk obstacle or change with respect to the risk factor of an already detected obstacle, the user is provided with a “stop” notification from the speech module. Additionally, the user is provided with the updated obstacle statuses, *i.e.* spatial location, category and distance. The text-to-speech result is communicated via Bluetooth with the speaker attached to the ear of the user for playback. It is important to mention here that, in case of repeated object detections, the BCU component avoids the playback of the same detected object based on the change of the scenery, which enables the system to prevent unnecessary playbacks.

Furthermore, the functionalities of pause, start and stop of the system are accessible to the user through button click gestures available on the Bluetooth headset of the BCU.

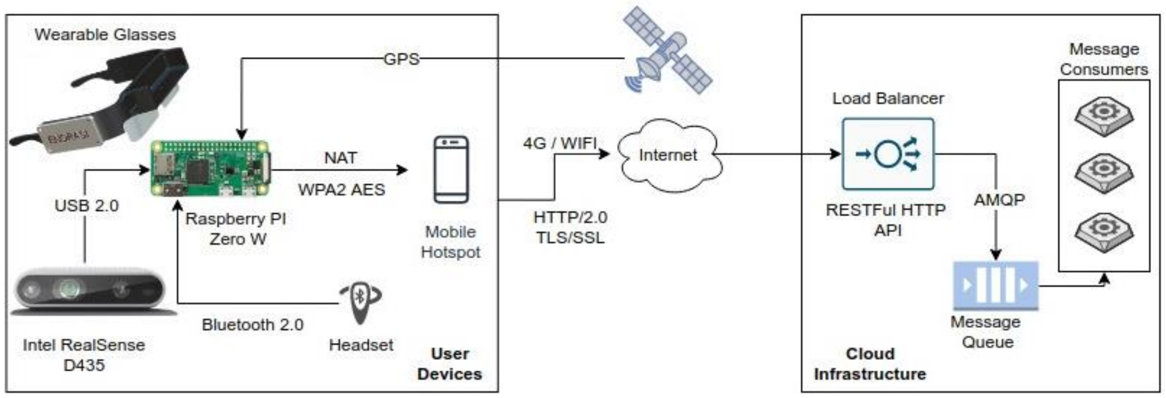


**Figure 5.4** VPS architecture overview illustrating the components of the system along with their interconnectivity.

### 5.2.4.2 The Object Recognition module

Although object detection has a critical role in the safety assurance of VCP, the VPS aims to provide an effective object and scene recognition module, which enables the user to make decisions based on the visual context of the environment. More specifically, object recognition provides the capability to the user to identify what type of object has been detected by the object detection module. Object recognition can be considered as a more complex module compared to object detection, since it requires an intelligent system that can incorporate the additional free parameters required to distinguish between the different detected object.

CNNs have also been used for object and scene recognition tasks in the context of assisting VCP. In the work of (Poggi & Mattoccia 2016), a mobility aid solution was proposed that uses a LeNet architecture for object categorization in 8 classes. An architecture named “KrNet” was proposed in (S. Lin et al. 2018), which relies on a CNN architecture to provide real-time road barrier recognition in the context of navigational assistance of VCP. A terrain awareness framework was proposed in (K. Yang et al. 2018) that uses CNN architectures, such as SegNet (Vijay Badrinarayanan et al. 2017), to provide semantic image segmentation. In the proposed VSP the LB-FCN *light* (Diamantis et al. 2019) was used to classify the bounding boxes of the object detection component, as it offers high classification performance with relatively low computational requirements compared to other mobile oriented architectures, such as (Sandler et al. 2018).

The original LB-FCN *light* architecture was trained on the binary classification problem of staircase detection in outdoor environments. In order to train the network on obstacles that can be found by the VPS, a new dataset named "Flickr Obstacle Recognition" dataset was created **(Figure 5.5)** with images found on the popular social media platform "Flickr" (Flickr Inc. 2019). The dataset contains 1646 RGB images of various sizes that contain common obstacles which can be found in the open space. More specifically, the images are weakly annotated based on their content in 5 obstacle categories; "benches" (427 images), "columns" (229 images), "crowd" (265 images), "stones" (224 images), and "trees" (501 images). It is worth mentioning that the dataset is considered relatively challenging, since the images were obtained by different modalities, under various lighting conditions and different landscapes.

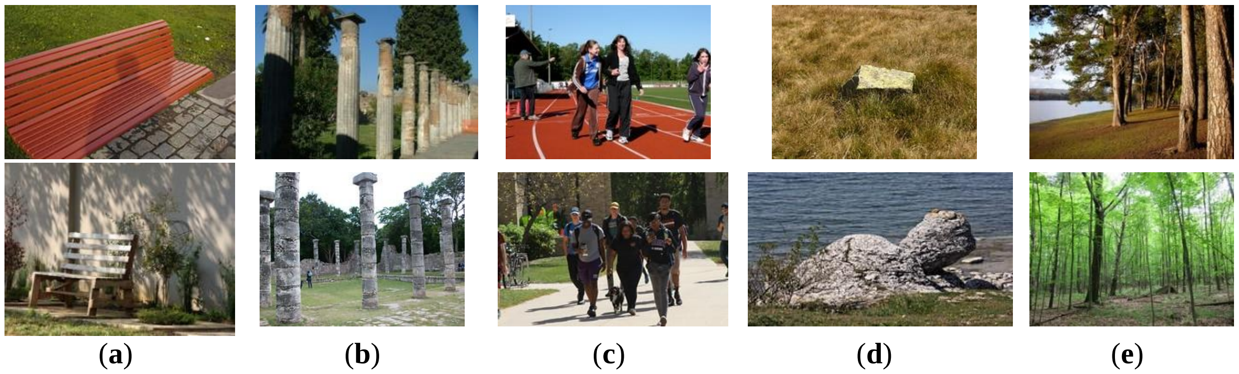


**Figure 5.5** Sample images from the five obstacle categories, (a) "benches", (b) "columns", (c) "crowd", (d) "stones", and (e) "trees" from the "Flickr Obstacle Recognition" dataset.

For the implementation of the LB-FCN *light* architecture, the popular Keras (Chollet & others 2015) python library with the Tensorflow (Abadi et al. 2016) was used as the backend tensor graph framework. To train the network, the images were downscaled to a size of 224×224 pixels and zero-padded where needed to maintain the original aspect ratio. No further pre-processing was applied to the images. For the network training, the Adam (Kingma & Ba 2014) optimizer was used with an initial learning rate of alpha = 0.001 and first and second moment estimates exponential decay as rate beta1 = 0.9 and beta2 = 0.999, respectively. The network was trained using a high-end NVIDIA 1080TI GPU equipped with 3584 CUDA cores (Sanders & Kandrot 2010), 11 GB of GDDR5X RAM, and base clock speed of 1480 MHz.

To evaluate the recognition performance of the trained model, the testing images were composed by the detected objects found by the object detection component of the system. More specifically, 212 obstacles of various sizes were detected. The pre-processing of the validation images was similar to that described above for the training set.

For comparison, the state-of-the-art mobile-oriented architecture named "MobileNet-v2" (Sandler et al. 2018) was trained and tested using the same training and testing data. The comparative results, presented in Table 3, demonstrate that the LB-FCN *light* architecture is able to achieve higher recognition performance, while requiring lower computational complexity, compared to the MobileNet-v2 architecture (**Table 5.4**).

**Table 5.4** Comparative classification performance results between the LB-FCN *light* (Diamantis et al. 2019) architecture and the MobileNet-v2 (Sandler et al. 2018) architecture.

| **Metrics** | **LB-FCN *light*** | **MobileNet-v2** |
|---|---|---|
| Accuracy | **93.8%** | 91.4% |
| Sensitivity | **92.4%** | 90.5% |
| Specificity | **91.3%** | 91.1% |

## 5.3 Digital Twin for Simulation and Evaluation of Assistive Navigation Systems

The assistive navigation of visually impaired individuals requires the development of different algorithms for obstacle detection, recognition, and avoidance, as well as path planning. The assessment and optimization of such algorithms in the real world is a painstaking process that requires repetitive measurements under stable conditions, which is usually difficult to achieve, as well as costly. To this end, digital twin environments can be used to replicate relevant real-life situations, enabling the evaluation and optimization of algorithms through adjustable and cost-effective simulations. This section presents a digital twin framework for the simulation and evaluation of assistive navigation systems, and its application in the context of a camera-based wearable system for visually impaired individuals in an outdoor cultural space. The system incorporates an obstacle avoidance algorithm based on fuzzy logic. The utility and the effectiveness of this framework are demonstrated with an indicative simulation study.

Currently, the rapid development of computer visualization tools and techniques has enabled some advanced visual applications based on virtual reality (VR), which have been used in a wide variety of scientific and industrial fields. VR is a contemporary visualization tool through which realistic virtual environments can be produced to help researchers simplify their workflow. Mainstream VR technologies used in engineering fields, such as architecture or civil engineering, can provide virtual environments with high rendering quality that resemble closely the real environment. More advanced VR tools enable the simulation of human actions or real conditions. Contrary to augmented reality, VR is a completely simulated environment, which does not link with the real world, and it provides an interactive computer-generated environment that enables users to perform several tasks. VR has been successfully used in flight and driving simulators, robot simulators, training in medicine, and production line simulations. Many VR platforms are available nowadays, which are either commercial or open source, and are commonly used to simulate and optimize the navigation of robots or avatars.

Several research groups have been working on systems and devices that are used to assist individuals with different kinds of disabilities, such as visually impaired (VI) individuals (Dimas et al. 2020). VR simulations can be employed to assess the performance of such systems, facilitating their optimization. One of the main aspects related with the assisted navigation of VI individuals is obstacle avoidance. VR simulations can be used to evaluate, in a highly-adjustable and cost-effective environment, different methodologies that can help VI individuals avoid the collision with obstacles. Over the years, several studies with respect to obstacle avoidance have been proposed. Although these systems are effective, when it comes to detecting and avoiding obstacles, they are typically based on methodologies that have been designed for robotic systems (Mohanan & Salgoankar 2018). While in theory such systems can be directly applied to human guidance, they are usually uncomfortable or alien to the individual, rendering them unusable in a day-to-day basis since they fail to address human specific requirements (Ntakolia, Dimas, et al. 2020).

Considering the state-of-the-art developments and to deal with the issue of assistive navigation testing, in this chapter, a novel simulation framework, *i.e.*, a “Digital Twin” (DT), is proposed. The proposed DT combines a simulation environment capable of converting real-world environments into a 3D simulation with a general-purpose obstacle avoidance framework, which enables rapid and reliable prototyping and testing of navigation pipelines. The proposed framework is characterized by its generality since all the components of both the simulation and the obstacle avoidance framework can be changed, facilitating the testing of new algorithms. Furthermore, the implementation of the DT framework was conducted using the Python programming language, which is commonly used by researchers. The source code of the framework and the 3D assets used in the simulations have been released as open source.

## 5.3.1 State-of-the-Art Simulation Environments

Virtual environments implementing the DT concept for navigation have been proposed mainly in the context of robotics, whereas fewer have been proposed in the context of assistive systems for people with disabilities. In both contexts obstacle avoidance comprises an integral component of the respective navigation systems.

### 5.3.1.1 Robot Navigation

There are several available commercial or open-source VR simulation platforms, including Gazebo, Unity, USARSim, V-REP, Choreonoid, Open-HRP, AI2THOR, VirtualHome, Webots, and SIGVerse (Inamura & Mizuchi 2020; Inamura & Mizuchi 2017). These platforms are commonly used to simulate and optimize the navigation of robots or avatars. For example, (S. Zhang et al. 2019) used a simulated environment in Webots to evaluate an algorithm for on-line terrain complexity evaluation according to the touchdown times of swing feet in order to make a quadruped robot obtain the terrain information without using any machine vision system. (Li et al. 2019) proposed a VR environment in Unity to improve the navigation of a mecanum wheel mobile robot. (Y.-J. Han et al. 2018) proposed a humanoid robot navigation algorithm consisting of an image processing and optimization algorithm, which realizes navigation with less computational time than conventional navigation algorithms using map building and path planning processes. They used a VR environment to assess if their algorithm could cope with an environment that changes in real-time. In another study (Pang et al. 2019), analyzed the characteristics of swarm robotic exploration and introduced an improved random walk method, where each robot adjusts its step size adaptively to decrease the number of repeated searches by estimating the density of robots in a virtual environment. In (Inamura & Mizuchi 2020; Inamura & Mizuchi 2017) combined cloud computing and an immersive VR system to perform and measure cognitive and social human-robot interaction in a VR environment. In (Hungerford et al. 2016) proposed an algorithm that allows the coordination of robots in order to fully cover inaccessible portions of Voronoi cells with complete, non-overlapping coverage. The performance of their algorithm was quantified in the Webots simulator using e-puck robots in different environments with obstacles with different characteristics.

For indoor robot navigation, (Guldenring et al. 2020) developed frameworks build on the separation between global and local planners and presented a system to train neural networks for such a local planner component, explicitly accounting for humans navigating the space. More specifically, deep reinforcement learning (DRL)-agents were trained in randomized virtual 2D environments with simulated human interaction. In another research, (Fraichard & Levesy 2020) investigated to what extent the results obtained in a crowd simulation domain could be used to control a mobile robot navigating among people. Their results revealed that all the investigated techniques entailed safety problems, *i.e.*, they would cause collisions in the real world. Finally, (Asiain & Godoy 2020) reviewed navigation approaches for multi-robot systems in VR environments.

#### 5.3.1.2 Assistive Systems

Several studies have been focused on the development of systems and devices that can be used to assist individuals with different types of disabilities, such as VI individuals (Dimas et al. 2020). The performance of such systems and devices can be assessed and optimized through VR simulations. For instance, (Kreimeier & Götzelmann 2019) presented a user study where individuals with visual impairments explored a virtual environment by walking in a VR treadmill and reported the first results from their feasibility study investigating this walk-in-place interaction. In another study, (Moldoveanu et al. 2017) focused on the challenges faced by VI individuals in order to become familiar with the functionalities of electronical devices. Their study demonstrated the importance of training VI individuals for an advanced sensory substitution device. A series of 3D virtual scenes were developed using the advantages of VR in order to ease the training process for the device. In Unity (Tao et al. 2017) developed a validation framework of an indoor navigation system for blind and VI users as a step toward the development of cost-effective indoor way-finding solutions for VI users who require detailed landmark-based navigation instructions that can help them arrive at the chosen destination.

In a different study, (Zhao et al. 2018) created Canetroller, which is a haptic cane controller that simulates white cane interactions, enabling VI individuals to navigate a virtual environment by transferring their cane skills into the virtual world. Indoor and outdoor VR scenes were designed to assess the effectiveness of their controller. VI individuals typically have difficulties in identifying people in crowded environments, and the difficulties may differ depending on the origin of the visual impairment. To examine the potential differences in visual search performance, (Bennett et al. 2018) developed a first-person perspective VR environment integrated with eye-tracking, designed to simulate the dynamic movement of humans in a hallway. The participants were tasked with locating a specific target individual walking among a crowd of people moving in various directions in the hallway. To assess the effect of task difficulty, factors of crowd density and presence of object disorder within the hallway were altered. In general, VI individuals perceive their surroundings differently than those with healthy vision and it is difficult to realize how they perceive their surroundings. To this end, (Stock et al. 2018) introduced a VR platform capable of simulating the effects of common visual impairments, through which a realistic VR representation of actual visual fields obtained from a medical perimeter can be created. From a similar point of view, (McIntosh et al. 2020) investigated whether the experience of an impairment can be usefully simulated for empathetic design, which is of particular importance for the comprehension of proposed designs during the early planning and design phases, without costly and time-consuming use of full participatory processes. They concluded that there is significant potential for the use of VR as an approach to simulate the experiences of certain spaces by VI individuals, enabling empathetic design.

#### 5.3.1.3 Obstacle Avoidance

Together with obstacle detection and recognition, obstacle avoidance is a significant factor regarding the assisted navigation of VI individuals. Through VR, real environments can be simulated for the assessment and optimization of different algorithms that can assist VI individuals avoid collisions with obstacles. For example, (Zapf et al. 2016) used the CRYENGINE development engine (v3.5.6, Crytek, Frankfurt, Germany) to design a virtual pedestrian scenario. In particular, they employed simulated prosthetic vision to evaluate the prospective efficacy of peripheral retinal prostheses for guiding orientation and mobility in the absence of residual vision in comparison to an implant for the central visual field. CRYENGINE modifications included tracking of head movement and orientation through an integrated gyroscope on the head-mounted display, acquiring angular data and tallying of collisions. In another study, (Katz et al. 2012) developed NAVIG to increase the autonomy of VI individuals through a virtual augmented reality system, which assists in route selection and guidance for complex routes through integrating a geographic information system with different classes of objects. While its precision rate is high, it requires an internet connection in real-time. The concept of a virtual environment that allows experiencing unknown locations by real walking while remaining in a controlled environment was introduced in (Kunz et al. 2018). Since the complexity of this virtual environment is controllable, it can be adjusted from abstract training scenarios to real-life situations, such as train stations or airports.

The available literature with respect to obstacle avoidance is primarily focused on the navigation of autonomous systems, such as robots (Xu et al. 2017; Wyrkabkiewicz et al. 2020) and UAVs (Iacono & Sgorbissa 2018; Wang et al. 2020) in known (Pratama et al. 2016; Shi et al. 2010) or unknown environments (Chai et al. 2017; Kumar et al. 2017). These studies have mainly relied on multiple sensors to map the surrounding environment, detect obstacles, and find and plan routes to avoid them, using common and well-tested methods, such as BUG-based algorithms (Ng & Bräunl 2007) and SLAM (Saeedi et al. 2016).

Obstacle avoidance has also been an essential component of systems that aim to assist individuals with vision and kinetic disabilities (Dakopoulos & Bourbakis 2009; Manjari et al. 2020; Tapu et al. 2018). However, these systems are usually based on approaches that have been designed for robotic systems (Mohanan & Salgoankar 2018), which makes them inadequate to meet specific human-based requirements (Ntakolia, Dimas, et al. 2020). For instance, such systems usually require fine control over the movements of the subject, and sometimes require directional and speed changes to be applied in a very short time span, which are unnatural to humans. Trying to bridge the gap between robotics and human guidance systems, in the case of VI individuals, (Weiss et al. 2020) proposed an interesting approach, which is based on reinforcement learning to create navigation agents that mimic the real guidance dogs to which VI individuals are already familiar with, and promising results were reported. Deep learning-based object detectors are commonly used in obstacle avoidance applications. We recently proposed an uncertainty-aware obstacle detection approach (Dimas et al. 2020) (Section 5.2.4). A smartphone-based outdoor obstacle avoidance method using a Single Shot Detector (SSD) (W. Liu et al. 2016) was proposed in (Q. Chen et al. 2019). The authors used the lightweight MobileNetV2 (Sandler et al. 2018) CNN feature extractor as the backbone of the SSD, which was fine-tuned to detect typical road obstacles, such as cars, motorcycles, and pedestrians. Similarly, (Vaishnav et al. 2021) proposed a wearable device in a form of a hat. The device is based on a portable Raspberry PI Zero W platform which is attached to the hat and performs object detection and recognition using the You Only Look Once (YOLO) SSD detector (Redmon et al. 2016). The advantage of the system is its ability to perform detection and recognition tasks both online and offline. This is important especially in outdoor environments where the network availability can be a problem. While these systems can detect objects and obstacles at high framerates, they typically rely on voice commands to inform the user about the presence and the type of the obstacle along with its location with respect to the input sensors. This can be confusing since the user may have to perform multiple direction changes until the obstacle disappears from their field of view. The validation of obstacle avoidance algorithms and the comparison of their performance are crucial requirements for human-centric assistive systems. In theory, real-world experiments would be the preferred choice. In practice, it is relatively hard to maintain an unbiased environment, since the environment in which the algorithms need to be tested must remain unchanged, along with the participants with whom the experiments are conducted. In addition, as the environment remains the same, after a few experiments, the participants will develop memory of their surroundings, introducing positive bias to the results.

### 5.3.2 The Digital Twin Navigation Framework

For the development of the DT assistive navigation framework in this study, the open-source robot simulator Webots was chosen. Webots is a commercial robot simulator developed by Cyberbotics Ltd., which is used in more than 800 universities and research centers worldwide. It supports a wide range of hardware, including GPU rendering acceleration. In addition, it uses the open dynamics engine (ODE) for the detection of collisions and the dynamic simulation of rigid bodies. The ODE library allows the physics of the objects to be simulated. Moreover, in Webots, a large collection of sensors is incorporated, including distance sensors, light sensors, cameras, LiDARs, GPS, accelerometer, and force-sensors (Rosique et al. 2019). The proposed DT simulation framework can be divided into two logical components; the Graphical Simulator and the Obstacle Avoidance Framework.

#### 5.3.2.1 The Graphical Simulator

The graphical simulation framework of the DT aims to create a real-life environment based on data obtained from the real world. More specifically, the simulator is capable of transferring existing marked and unmarked terrains found in maps into the Webots environment, creating a realistic environment on which navigation algorithms can be tested, assessed, and optimized. The components of the DT simulation framework are illustrated in **(Figure 5.6)**.

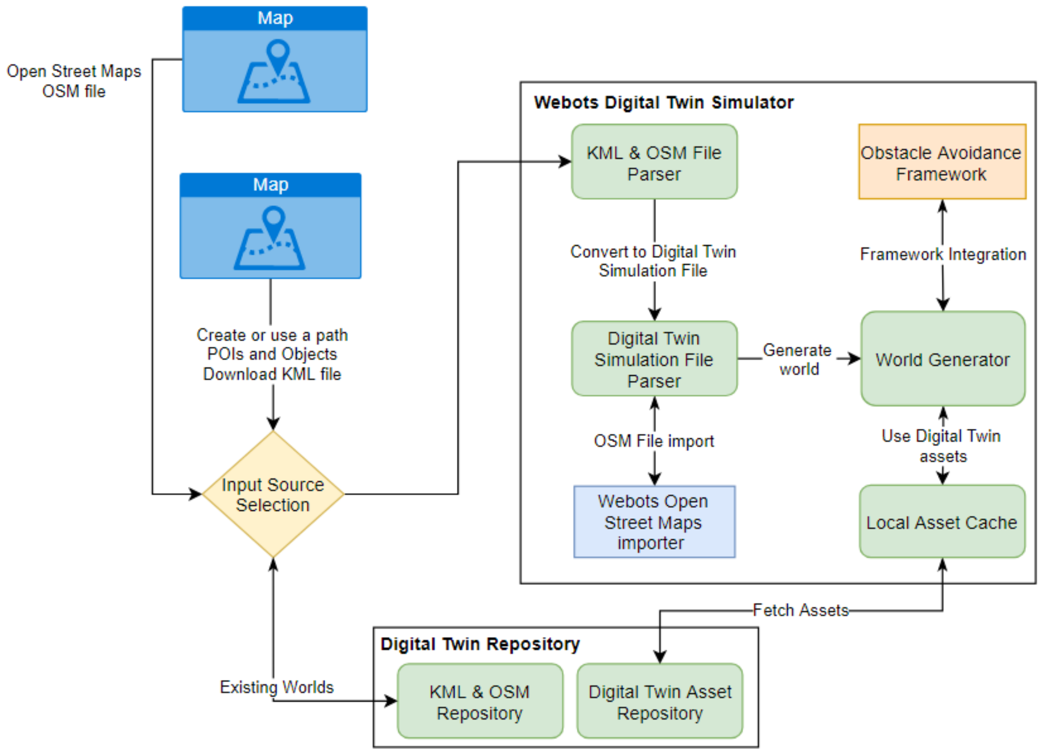

Map
Open Street Maps OSM file
Map
Create or use a path POIs and Objects Download KML file
Input Source Selection
Existing Worlds
Webots Digital Twin Simulator
KML & OSM File Parser
Convert to Digital Twin Simulation File
Digital Twin Simulation File Parser
Generate world
OSM File import
Webots Open Street Maps importer
Obstacle Avoidance Framework
Framework Integration
World Generator
Use Digital Twin assets
Local Asset Cache
Fetch Assets
Digital Twin Repository
KML & OSM Repository
Digital Twin Asset Repository

**Figure 5.6** The graphical simulation framework of the Digital Twin.

The framework provides the capability to import pathways and existing terrains from both Open Street Maps (OSM) and Google Earth, using different sets of importers. In particular, the framework makes use of the OpenStreetMap importer module of Webots to import data from Open Street Maps. Although this module was designed for automobile simulations, it can be used to import arbitrary mapped pathways from any location of the world. Various configuration options are available, including the ability to include roads, lakes, parking spots, trees, *etc.*, effectively enabling fine control of the resulting simulation. Unfortunately, pathways from unmapped or semi-mapped terrains, such as the Historical Triangle of Athens, cannot be imported in the module. To enter such pathways in the proposed simulation framework, the user can parse Keyhole Markup Language (KML) files, which can be obtained from Google Earth. KML files are in XML format, which is a standard developed by Google to express geographic annotations of two- and three-dimensional maps. KML files can be easily obtained from Google Earth by drawing paths and adding annotations to the map. An example pathway is illustrated in **(Figure 5.7)**

The DT simulation framework using a parser is capable of parsing both OSM and KML file formats, translating them into an internal DT simulation file format. This ensures the extensibility of the framework, since, if needed, more file formats can be added in the future. In some cases, such as when arbitrary random paths need to be generated for path planning algorithm testing, OSM and KML files are impractical.  For this reason, the DT parser is capable of parsing files in DOT file format, which is a widely used text file format for arbitrary graph representations.

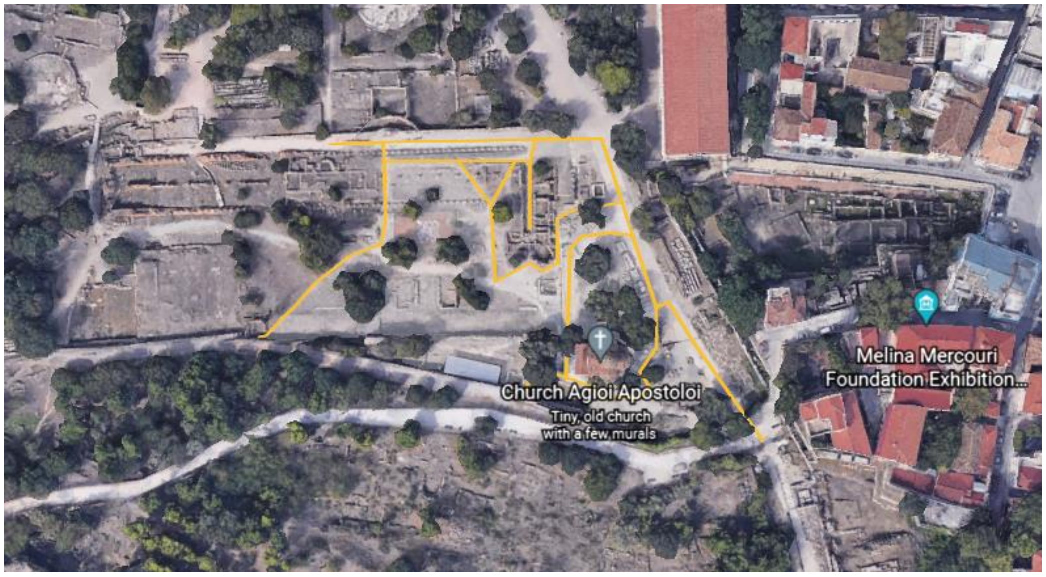

Melina Mercouri Foundation Exhibition...
Church Agioi Apostoloi
Tiny, old church with a few murals

**Figure 5.7** Example path from the Historical Triangle of Athens, outlined in Google Earth.

To setup the simulation, the framework parses the DT simulation file and, using the World Generator (WG) component of the framework, creates dynamically a simulation environment. The WG component controls various parameters of the simulation, including the GPS coordinate system translation, the scale of the simulated area, and the assets and obstacles that can be included. In real-life environments obstacles are not always present. Although obstacles can be added as markers in the KML or OSM files, this is can be a rather tedious and time-consuming process, since the size, location, and obstacle type need to be manually defined. For this reason, the WG component enables the user to dynamically insert obstacles and landmarks in the simulated environment, and control the scale, location, orientation, and type of the objects. This is particularly useful in scenarios where obstacle avoidance algorithms are tested, as it creates an unbiased simulation environment. The WG component of the framework is also capable of including moving obstacles, such as crowds, simulating their movement through a specified algorithm. A variety of crowd movement simulation algorithms have been proposed over the years (Curtis et al. 2016; Kim et al. 2016). For this reason, the WG component uses an abstraction that enables algorithm-independent crowd movement simulation. This is achieved by giving access to both the simulated surrounding and the movement controls of each simulated person individually. To simulate crowd movement, the open-source CROMOSIM Python library is integrated, which includes implementations of a variety of moving simulation algorithms found in (Maury & Faure 2018).

Although Webots includes a variety of built-in graphics, such as trees, benches, roads, stairs, rocks, and houses, in some cases custom objects need to be included in the simulation. For example, to replicate an archaeological site, such as the Historical Triangle of Athens, objects including ancient columns and temples are needed. To achieve this, the WG component is capable of automatically including in the simulated environment objects from external sources that are in WebBot (WBT) file format. Such objects can be designed in computer-aided design (CAD) software or obtained directly from existing CAD libraries and converted in WBT format.

Since designing 3D objects is a relative complex and time-consuming task, an environment in which such assets can be freely shared is highly desirable. For this reason, the presented DT framework includes a publicly available asset repository[3] in which peers can use and share custom 3D models. The asset repository of the framework aims to include both existing simulation files in the Webots "World" file format (WBT) and 3D models in VRML format. An important contribution of the asset repository is that it enables arbitrary simulations to be shared among peers. This allows fast prototyping, but most importantly, testing of different algorithms under the exact same conditions. The WD component of the simulation framework integrates directly with the report asset repository, enabling the framework to fetch and use the files, without requiring manual action. As files can be large, local asset caching and versioning are also included. In some cases, the public sharing of simulation files or custom 3D objects might be not feasible due to license and copyright. To this end, the WD component can use files located in the local hard drive or a private remote repository located in intranet, as long as the files are available through the HTTP protocol.

### 5.3.2.2 The Obstacle Avoidance Framework

The obstacle avoidance framework is an important component of the DT simulation. Although not simulation-specific, since it can be used in real-world scenarios, it can provide, in an abstract and unified way, all the components required for the problem of obstacle avoidance. More specifically, the framework comprises seven individual components which are illustrated in **(Figure 5.8)**

The first component of the framework is the Image Streamer (IS). The component provides a stream of images to the core framework as an object named ImageStreamItem. Along with the image, the object can incorporate metadata, such as a depth map, when the camera allows it, and the GPS location of the image. The framework provides multiple implementations of the IS component, including capturing images from a web camera, *i.e.*, Intel RealSense D435i, capturing images from an RTMP stream and an H264 encoded video. Finally, for the purpose of integration

[3] https://digital-twin.innoisys.com

in the DT simulation environment, an implementation of the IS that uses the RGB and depth cameras of Webots is also included.

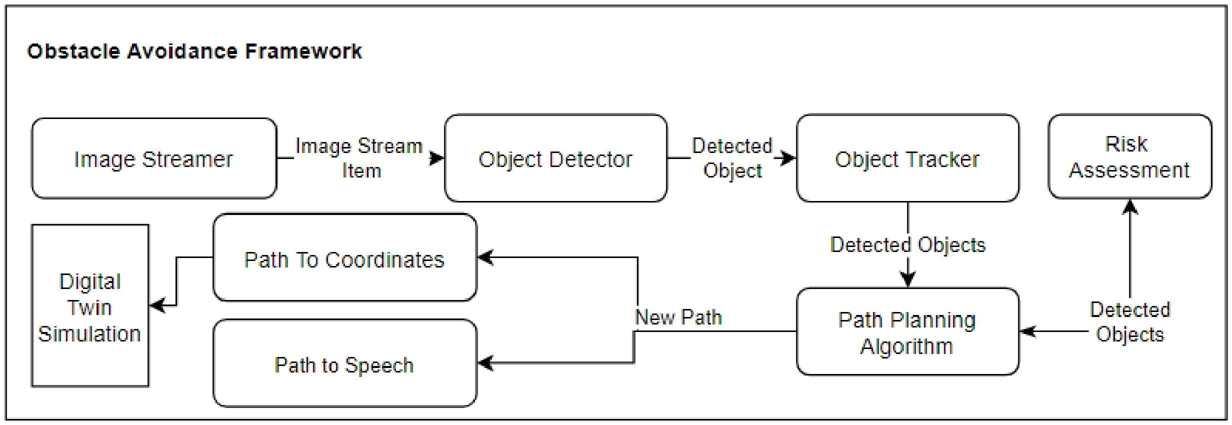


**Figure 5.8** The DT obstacle avoidance framework.

The Object Detector (OD) component of the framework aims to abstract the implementation details of the object detection and recognition algorithm. The component takes as an input one or more ImageStreamItems from the IS component and performs the detection and recognition processes. When one or more objects are detected in the input, the component creates objects that include the bounding box of the object within the image, the label of the recognized object, and the distance and location of the object with respect to the camera. Implementations of the OD component include the YOLOv4 object detector (Bochkovskiy et al. 2020), which is a lightweight deep learning-based object detection and recognition network and the object detection and recognition framework proposed by (Dimas et al. 2020). The latter is a complex two-stage object detection framework specifically designed to be used to assist visually challenged individuals to navigate in unknown environments. Although the OD component can be used directly in the Webots simulation environment, the performance of the object detection and recognition component cannot be validated directly on 3D models. To this end, the DT framework can associate 3D models with real images, which are automatically presented to the component as soon as the 3D model appears on the camera. A limitation of such a use-case is that the depth map component of the ImageStreamItem would not be associated with the corresponding RGB image.

The third component of the framework is the Object Tracking (OT) component, which receives as input the detected objects from the OD component and tracks them. The DT framework includes an implementation of tracking-by-detection algorithm proposed by (Bochinski et al. 2017) and an integration with the DT Webots simulation environment, in which the objects have an automatically generated unique identifier. It is worth mentioning that this component can be enabled or disabled dynamically, depending on the use-case scenario.

Path planning is an essential part in the obstacle avoidance framework, since its role is to estimate if a detected object is an obstacle that must be avoided and to find the optimal alternative path. For this reason, the path planning component of the framework is divided into two sub-modules; the path planning algorithm and the risk assessment module. When an object is detected, the path planning component initially identifies the collision risk of the individual and, based on that, the path planning algorithm handles the navigation.

When a new path is selected from the path planning component, it needs to be translated into the local coordinate system of the framework. To abstract this process, the Path to Coordinates (PtC) component has been introduced into the framework. The abstraction here is required since this component translates the new path into coordinates of the DT Webots simulation framework, and it is used as the output integration point of the obstacle avoidance framework of the DT system.

The last component of the framework is the Path to Speech (PtS) component, which acts as the integration point between users and the avoidance system output. It aims to abstract the algorithm used to translate the path into a form that humans can understand and interact with the surrounding environment.

Finally, the framework is able to capture all the details of the obstacle avoidance process into an output video along with the details in a log format that can be examined after the simulation has been completed. This enables the DT user to run parallel simulations and test different algorithms and environments, the results and behavior of which can be examined subsequently.

### 5.3.3 Simulation Studies

Vision-based navigation and accessibility for VI individuals in indoor places of cultural interest, such as museums, has been extensively investigated (Alkhafaji et al. 2016; Shah & Ghazali 2018) (Alkhafaji et al. 2016; Shah and Ghazali 2018). On the other hand, the accessibility of outdoor sites of cultural interest has been less explored despite the significance of such a venture. In the ENORASI project (Iakovidis et al. 2020; Dimas et al. 2020), a pre-commercial digital system has been investigated to assist the VI individuals navigate safely in outdoor environments of cultural interest, *e.g.*, archeological sites. While providing information concerning the sights in a descriptive way, the system supports the user with audible guidance and instructions for obstacle avoidance. The system comprises mainly a stereoscopic CV system for depth assessment through visual sensors incorporated in a wearable device, emotion-aware speech interaction through a microphone and earphones, and communication with a customized GPS-enabled mobile processing unit (MPU), such as a smart-phone or a tablet. The core advantage of the system is its robust performance based solely on visual sensors, without augmentation from additional sensors, such as ultrasound, LiDAR, and IMU sensors.

More specifically, the proposed system is based on image, video, and audio processing and analysis methods, which include CV algorithms for automatic object recognition and estimation of their distance from the user, as well as decision-making algorithms. The processing and analysis of the acquired data is conducted in the MPU and in a remote server through a computational cloud environment. The tasks performed in the MPU include obstacle detection, which is the most critical task associated with user safety, critical speech-based communication with the user, and object or scene recognition. On the other hand, more complex computational processes, such as decision making with respect to route planning and obstacle avoidance, are performed on remote servers accessible through the cloud. Assessing such methodologies in real word scenarios, is a challenging task, since it requires repetitive on-site measurements under changing conditions (*e.g.*, weather, lighting, crowds) which can be non-deterministic, as well as costly, since it may involve a considerable human effort and hardware adaptations.

To evaluate the proposed DT simulation framework, a series of simulations of vision-based navigation of VI individuals in the outdoor environment of the Historical Triangle of Athens were conducted. This geolocation was selected since it includes a variety of complex pathways that are not fully mapped in neither Google Earth nor Open Street Maps, making the navigation of VI individuals challenging.

### 5.3.3.1 The Simulation Environment

To demonstrate the simulation capabilities of the proposed DT framework an unmapped area of the Historical Triangle of Athens was chosen. The Triangle is composed of multiple routes and small pathways of historical significance. The selected pathway is illustrated in **(Figure 5.7)**, in which the yellow lines correspond to the route that was manually drawn in the Google Earth web application. Subsequently, the selected route was downloaded as a KML file, which was then imported using the DT simulation framework in the Webots platform. An obstacle-free route reconstructed using the route obtained from the KML file is illustrated in **(Figure 5.9)**

In unmapped regions, such as the one presented in **(Figure 5.10a**), obstacles are not included in the map. In such cases, random obstacles of any form can be added to the simulation. In **(Figure 5.10b)**, an example route with randomly set obstacles is presented. Although the position, the size, and the type of the obstacles included are randomly selected, a universal random seed can be used, which guarantees the same simulation properties in multiple experiments. Another important characteristic of the DT simulation is the scale in which the environment will be created. For example, in **(Figure 5.10)**, it can be noticed that the selected scale of the simulated route is much smaller than the actual one obtained from Google Earth. When simulating large-scale environments, this can help to reduce the runtime resources required by the simulation and help the researcher to identify problems of the testing algorithm faster.

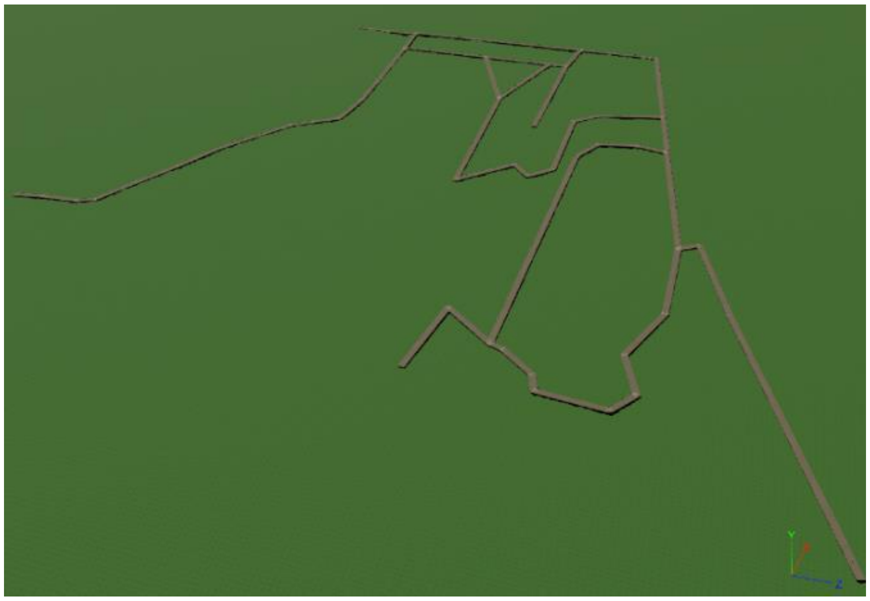
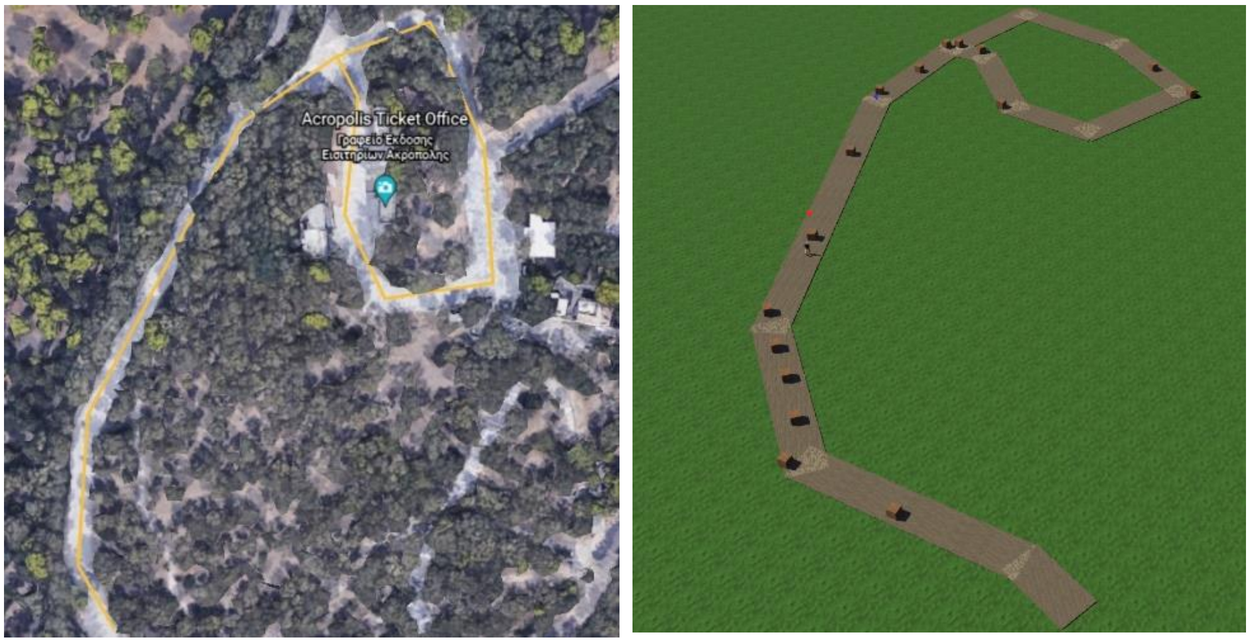


**Figure 5.9** A DT route reconstruction of a route obtained from Google Earth, as illustrated in **(Figure 5.7)**



(a) (b)

**Figure 5.10** (a) Simple route selected from Google Earth; (b) Reconstructed route in smaller scale with randomly positioned obstacles along the route.

## 5.3.3.2 The Obstacle Avoidance Algorithm

To demonstrate the capabilities of the obstacle avoidance framework, an experiment using the route illustrated in **(Figure 5.10)** was conducted, to assess a simple local path planning algorithm for VI human navigation. The Webots environment provides an RGB camera along with a depth camera that can be used as an input for the image streamer of the obstacle avoidance framework. Using the depth information obtained from the image streamer along with the name of the object, which is provided as meta-data information along with the image stream item, an object detector was created. Using the track-by-detection approach (Bochinski et al. 2017), the object tracker of the framework tracks the obstacles throughout the simulation. This is done to avoid multiple path changes from the local path planning algorithm, since the same obstacle is visible in multiple frames. Subsequently, the detected obstacles are passed to the path planning algorithm for risk assessment. The path planning algorithm navigates the individual considering the following criteria:

- the distance from the obstacle;
- the location of the obstacle with respect to the individual;
- the angle required to by-pass the obstacle

More specifically, the risk assessment algorithm uses the bounding boxes of the detected obstacles to calculate the distance from the individual. The distance is measured by averaging the depth map of the bounding box. In some cases, noisy depth points might exist in the depth map, which can affect the measurement performance. To compensate for this, before the distance calculation, the standard deviation of the depth map is computed, and outliers are removed. Thus, the distance $s_o$ from an obstacle $o$ enclosed by a bounding box $B$ of size $w \times h$, where $w$ is the width and $h$ the height of the bounding box, can be expressed as:

$$s_o = \frac{\left(\sum_{i=1}^{w}\sum_{j=1}^{h}\begin{cases}\mu \ \ , \ \mu - \sigma > B_{ij} > \mu + \sigma \\ B_{i,j}, \ \mu - \sigma \leq B_{ij} \leq \mu + \sigma\end{cases}\right)}{(w \cdot h)} \tag{5.1}$$

where $\mu$ is the mean depth of the elements of $B$ and $\sigma$ is the standard deviation. It has been found that, when the outliers are considered, the average distance error is reduced from ±0.7 m to ±0.2 m.

To calculate the angle under which the individual can avoid the obstacle, the algorithm determines the angles from the left and right sides of the bounding box with respect to the individual. Since the horizontal and vertical field of view (FoV) of the camera is known for the simulated camera (Intel RealSense D435i), *i.e.*, 86 and 57, respectively, the horizontal angle $h_\varphi$ and vertical angle $v_\varphi$ of a pixel $p$ can be calculated as follows:

$$h_\varphi^p = \left(\frac{\left(p_x - \frac{W}{2}\right)}{\left(\frac{W}{2}\right)}\right)\left(\frac{H_{FoV}}{2}\right) \tag{5.2}$$

$$v_{\varphi}^{p} = \left(\frac{\left(p_y - \frac{H}{2}\right)}{\left(\frac{H}{2}\right)}\right)\left(\frac{V_{FoV}}{2}\right) \tag{5.3}$$

where $I$ denotes an image with width $W$ and height $H$, and $p_x$ and $p_y$ are the positions of the pixel $p$ in the image.

Using the outermost pixel of each side of the bounding box, the algorithm calculates the angle of the left and right side of the obstacle according to Eq. (5.2), as we are only interested in the horizontal angle. Similarly, the location of the obstacle with respect to the target GPS coordinates is determined by calculating the location of the obstacle, which is calculated by translating the coordinates of the individual using the angle and distance of the previous step. This process is repeated for all detected obstacles that are within a specific distance $c$. This threshold is a hyper-parameter of the algorithm and is selected based on the application needs. In the case of VI individuals, according to (Ntakolia, Dimas, et al. 2020), this is defined as any object within the distance of 2.5 m.

In contrast to robots, humans usually interpret the surrounding environment in verbal, vague terms; that is, instead of expressing turns using degrees, they use verbal terms, such as “small”, “large”, or “medium” turn. Furthermore, in unknown, outdoor environments, obstacle detection and depth estimation contain uncertainty, which is introduced by sources such as greylevel ambiguity, noise introduced by the sensor, and vagueness of image features (Chacón M 2006). For these reasons, the path planning algorithm uses fuzzy logic to determine the optimal path for navigating the individual around the obstacle. This process involves the fuzzification of the crisp values, the number of obstacles $n$, the distance $s$ from the obstacle, and the turn angle $a$ of the individual into the fuzzy domain as $\tilde{n}_v, \tilde{s}_v$ and $\tilde{a}_v$, respectively. Given each crisp value, the uncertainty can be modeled by fuzzy sets:

$$\tilde{n}_v = \left\{\langle x, \mu_{n_{v,}}(x)\rangle \mid x \in U_n\right\}, v = \{few, moderate, many\} \tag{5.4}$$

$$\tilde{s}_v = \left\{\langle x, \mu_{s_{v,}}(x)\rangle \mid x \in U_s\right\}, v = \{short, moderate, long\} \tag{5.5}$$

$$\tilde{a}_v = \left\{\langle x, \mu_{a_{v,}}(x)\rangle \mid x \in U_a\right\}, v = \{small, medium, large\} \tag{5.6}$$

The representing overlapping value intervals can be expressed linguistically, *e.g.*, “small”, “medium”, or “large” in the case of the angle, “short”, “moderate”, or “long” in the case of the distance, and “few”, “moderate”, or “many” in the case of the number of obstacles. $U_n, U_s$, and $U_a$ represent the universes of discourse for the fuzzy sets defined for the number of obstacles Eq. (5.4), the distance Eq. (5.5), and the angle Eq. (5.6), respectively. The quality of a path $p$ is also

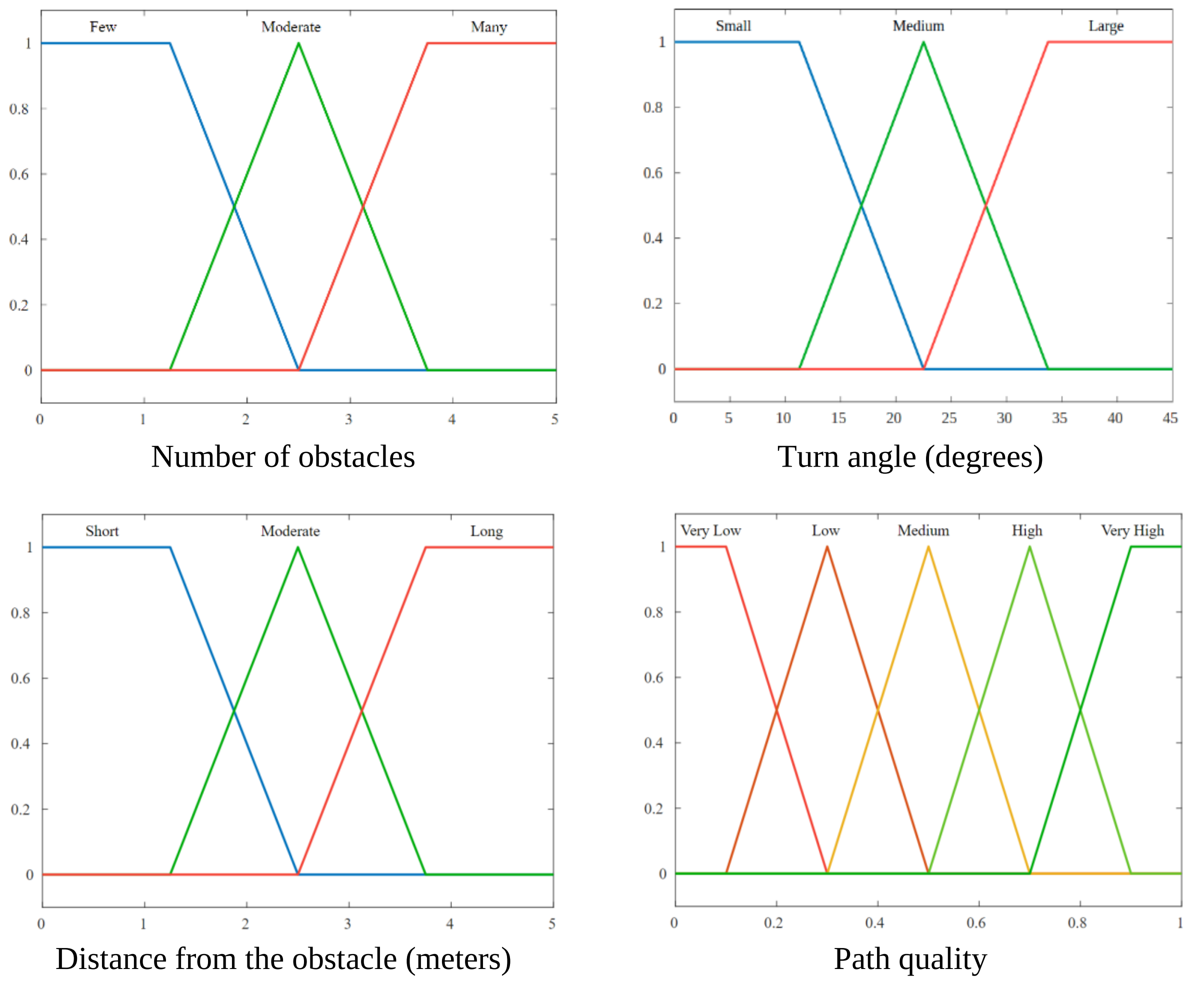


**Figure 5. 11** Visual representation of the membership functions of the four fuzzy sets.

represented by fuzzy sets, which can be linguistically expressed as "very low", "low", "medium", "high", "very high" in the fuzzy domain $\tilde{p}_v$.

$$\tilde{p}_v = \left\{\langle x, \mu_{p_{v,}}(x)\rangle \mid x \in U_p\right\}, v = \{very\ low, low, medium, high, very\ high\} \tag{5.7}$$

where $U_p$ is the respective universe. The membership functions, illustrated in **(Figure 5.11)**, were selected based on the possible values of each variable, such that $U_n = [0,5]$, $U_s = [0,5]$, $U_a = [0,45]$, and $U_p = [0,1]$. The quality of the path is a real number ranging from very low (0) to very high (1). The turn angle ($U_a$) and distance from the obstacle ($U_p$) are measured in degrees and meters, respectively. To determine the best path for avoiding the obstacle, for each side, the algorithm translates the crisp numerical values of the number of obstacles, distance of the individual, and turn angle that the individual must perform, into the fuzzy domain.

**Table 5.5** Fuzzy rules of local path planning algorithm

| Rules | Obstacles | | Turn | | Distance | | Quality path |
|---|---|---|---|---|---|---|---|
| Rule 1 | Few | AND | Small | AND | Short | → | Very high |
| Rule 2 | Few | AND | Small | AND | Moderate | → | High |
| Rule 3 | Few | AND | Medium | AND | Short | → | High |
| Rule 4 | Moderate | AND | Small | AND | Short | → | High |
| Role 5 | Moderate | AND | Small | AND | Moderate | → | High |
| Rule 6 | Few | AND | Small | AND | Long | → | Medium |
| Rule 7 | Few | AND | Medium | AND | Moderate | → | Medium |
| Role 8 | Moderate | AND | Small | AND | Long | → | Medium |
| Rule 9 | Moderate | AND | Medium | AND | Short | → | Medium |
| Rule 10 | Moderate | AND | Medium | AND | Moderate | → | Medium |
| Rule 11 | Many | AND | Small | AND | Short | → | Medium |
| Rule 12 | Many | AND | Small | AND | Moderate | → | Medium |
| Rule 13 | Many | AND | Small | AND | Long | → | Medium |
| Rule 14 | Many | AND | Medium | AND | Short | → | Medium |
| Rule 15 | Few | AND | Medium | AND | Long | → | Low |
| Rule 17 | Few | AND | Large | AND | Moderate | → | Low |
| Rule 18 | Few | AND | Large | AND | Long | → | Low |
| Rule 19 | Moderate | AND | Medium | AND | Long | → | Low |
| Rule 20 | Moderate | AND | Large | AND | Short | → | Low |
| Rule 21 | Many | AND | Medium | AND | Moderate | → | Low |
| Rule 22 | Many | AND | Large | AND | Short | → | Low |
| Rule 23 | Moderate | AND | Large | AND | Moderate | → | Very Low |
| Rule 24 | Moderate | AND | Large | AND | Long | → | Very Low |
| Rule 25 | Many | AND | Medium | AND | Long | → | Very Low |
| Rule 26 | Many | AND | Large | AND | Moderate | → | Very Low |
| Rule 27 | Many | AND | Large | AND | Long | → | Very Low |

The quality of the path is determined using the fuzzy rules defined in (**Table 5.5**) and the Mamdani inference methodology (Gopal 2019). The quality of each path $p$ is then defuzzyfied into a crisp value by calculating the Center of Gravity (CoG) as follows:

$$p_{t_{quality}} = \frac{\sum_{i=1}^{k} p_{t_i} \cdot \mu_{p_v}(p_{t_i})}{\sum_{i=1}^{k}(\mu_{p_v}(p_{t_i}))} \tag{5.8}$$

where $t$ represents the side (left or right), $k$ the number of subareas obtained by the rule inference, and $p_{t_i}$ the value of $p_t$ in the center of the area $i$.

Subsequently, the path with the highest quality is selected. To navigate the individual to the new location, (5.1) and (5.2) are calculated again, taking into consideration the minimum distance $c$ at which the individual can safely bypass the obstacle. Then, the new location is translated into verbal instructions, informing the individual about the angle and the distance. When the obstacle disappears from the image, the algorithm re-calculates the path according to the coordinates of the individual and the target location, informing the individual about the angle change. This process is repeated until target location is reached.

#### 5.3.3.3 Indicative Results

Simulations were performed for different unmapped pathways from the environment of the Historical Triangle of Athens, with a random number obstacles introduced per simulation. The obstacle avoidance algorithm described in the previous paragraph was tested. In each simulation the obstacles were detected by the DT of the ENORASI system and based on the obstacle avoidance algorithm, the agent representing the VI individual was navigated. In every simulation, the agent was safely navigated to its target destination, verifying that the obstacle avoidance algorithm is effective. A visualization of the trajectory followed by the agent in one of the simulated pathways of the Historical Triangle of Athens is illustrated in **(Figure 5.12)**. The blue line indicates the trajectory that would have been followed if the pathway did not contain any obstacles, whereas the red line indicates the actual trajectory of the agent obtained after the obstacle avoidance simulation. This is a representative result, since similar results were obtained in all simulations. Therefore, the simulation results indicate that this algorithm can be used for the safe navigation of VI individuals.

To integrate the path planning algorithm into the simulation, the generated path is translated into the coordinate system of the DT simulation. In parallel, the resulting path is converted to speech using the PtS component of the obstacle avoidance framework. The translation is handled by the Python pyttsx3 library (Natesh 2021), which is a cross platform text synthesis library. Although the simulation is executed in the Webots environment, it is important to be able to view the navigation from the individual’s perspective. For this reason, the obstacle avoidance framework records the entire 3D simulation, including the audio, as perceived by the individual, into an MP4 file for further examination.

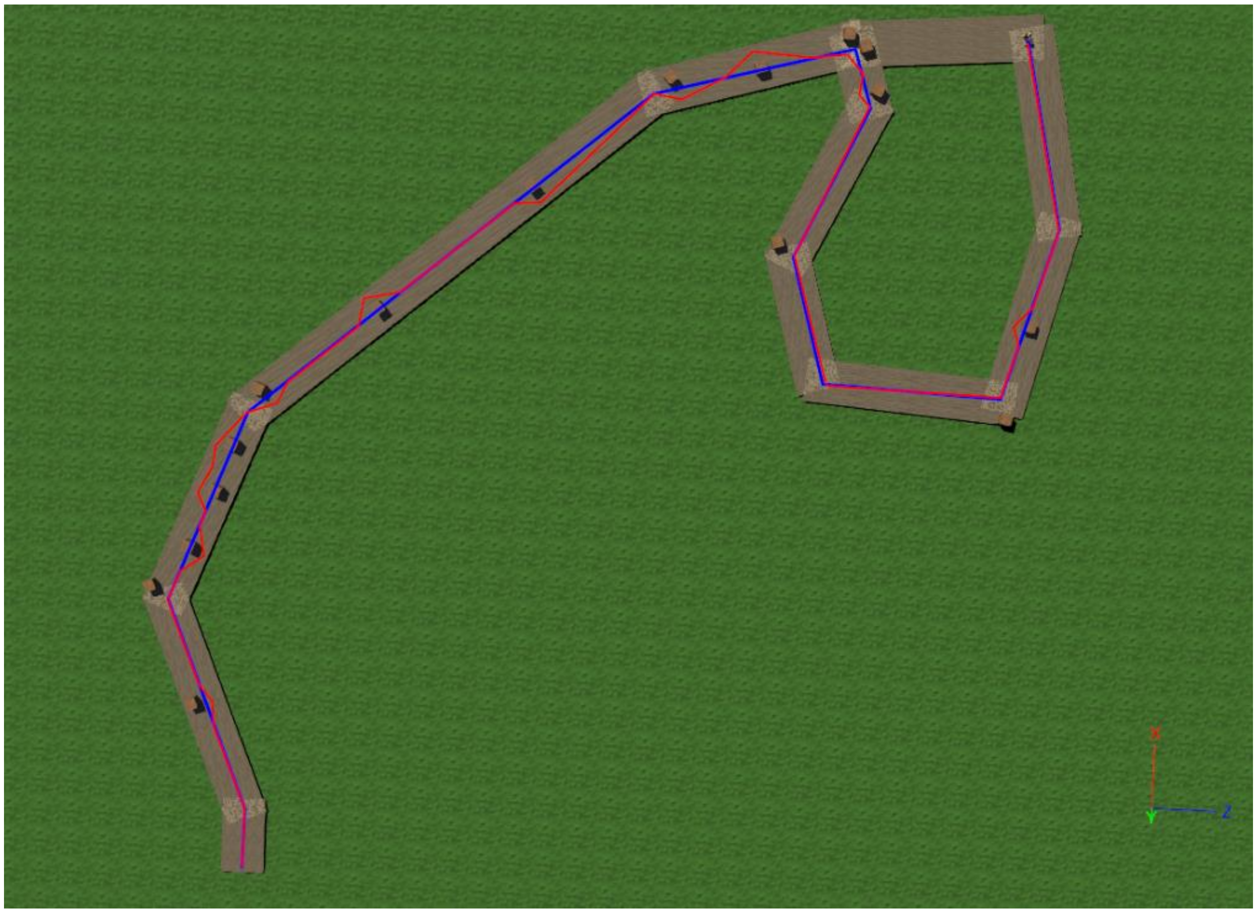


**Figure 5.12** Visual representation of the trajectory followed by the VI individual using the proposed algorithm. The red and blue lines denote the trajectory followed with and without obstacles, respectively.

### 5.3.3.4 Discussion

The VR technology enables researchers to perform multiple experiments in environments that closely resemble the real world. The capability to simulate scenarios free of unpredictable factors, such as weather conditions, allows the re-producibility of experiments, which is difficult and sometimes impossible to be achieved in the real world. The proposed DT framework, which is based on the Webots simulation environment, includes a generalized approach for VR simulations in the context of computer-aided navigation, and aims to provide a general and extensible framework that enables the assessment of algorithms in a standardized and unbiased environment. The experiments conducted in this study demonstrated that the proposed DT can automatically generate VR environments based on real-world environments, such as the ones found in Google Earth and Open Street Maps. When the characteristics of the environment to be simulated are known, the DT framework can introduce 3D objects with a size that is proportional to the real ones, which is especially important for the evaluation of algorithms that take into consideration the spatial size of objects. Although a variety of 3D assets are available, their use in a simulation environment is not always straightforward, mainly due to incompatibility with the platform or licensing issues. Since these assets are essential for the simulation of realistic environments, the above problem is addressed by the DT publicly-available asset repository, in which researchers can share their 3D models, thereby accelerating the process of performing realistic simulations using the DT framework. It should be mentioned that the significance and usefulness of such DT frameworks has been augmented during the COVID-19 pandemic, since outdoor measurements and algorithm verification studies were impossible to be conducted under the social distancing rules and the quarantine measures taken all over the world.

The DT obstacle avoidance framework was developed due to the necessity of evaluating assistive navigation systems for VI individuals. It is composed of abstract interchangeable components that can be used in the context of computer vision-based navigation. Moreover, it can be used by researchers to evaluate different aspects of the navigation pipeline, including obstacle detection, recognition, tracking, and avoidance, in a simulated environment. To enable independent algorithm testing, all components of the pipeline provide simulated equivalents, which can be used in the DT simulation environment. The conducted experiments based on our obstacle avoidance framework illustrate how the DT can be used to develop and assess a simple local path planning algorithm in the context of VI human navigation. The algorithm was developed to address specific human requirements (Ntakolia et al. 2020), and to enable easy navigation around obstacles, without requiring frequent direction changes that are typically met in navigation methodologies made for robotic systems, such as BUG-based algorithms (Ng and Bräunl 2007). Using the DT framework, the algorithm was tested in multiple scenarios of unmapped regions of the Historical Triangle of Athens. The scenarios included the random placement of obstacles along the automatically generated pathways. A major advantage of the proposed DT is that the obstacle avoidance framework is independent, *i.e.*, it can be integrated with the DT simulation environment and can be used without any significant modifications in real-world environments, enabling quick prototyping and testing.

# CHAPTER 6

# ASML: Algorithm-Agnostic Architecture for Scalable Machine Learning

ML applications are growing in an unprecedented scale. The development of easy-to-use machine-learning application frameworks has enabled the development of advanced artificial intelligence (AI) applications with only a few lines of self-explanatory code. As a result, ML-based AI is becoming approachable by mainstream developers and small businesses. However, the deployment of ML algorithms for remote high throughput ML task execution, involving complex data-processing pipelines can still be challenging, especially with respect to production ML use cases. This chapter presents a novel system architecture (Diamantis & Iakovidis 2021) that enables Algorithm-agnostic, Scalable ML (ASML) task execution for high throughput applications. It aims to provide an answer to the research question of how to design and implement an abstraction framework, suitable for the deployment of end-to-end ML pipelines in a generic and standard way. The ASML architecture manages horizontal scaling, task scheduling, reporting, monitoring and execution of multi-client ML tasks using modular, extensible components that abstract the execution details of the underlying algorithms. Experiments in the context of obstacle detection and recognition, as well as in the context of abnormality detection in medical image streams, demonstrate its capacity for parallel, mission critical, task execution.

## 6.1 Introduction

Deep learning growth has triggered the appearance of frameworks for easy development of ML-enabled applications. Many of these frameworks are supported by tech industry leaders, such as Google, Facebook and Microsoft, which usually provide deep learning Platforms as a Service (PaaS) (Beimborn et al. 2011) or Software as a Service (SaaS) (Waters 2005) on their Cloud Computing infrastructures, specialized in executing deep learning frameworks. For instance, Google, which supports the Tensorflow framework, provides Google Cloud. This is a general purpose cloud computing service, enabled by Tensor Processing Units (TPUs) (Jouppi et al. 2017), offering better performance for deep learning applications that use that framework.

While pre-configured virtual machines and containerized ML solutions exist, they still require a technical understanding of the underlying platform; thus, they are not directly applicable to any production environment. For this reason, SaaS providers, such as Amazon and Google offer in their platforms pre-trained deep learning models for specific use cases, typically through a representational state transfer (RESTful) HTTP (Richardson & Ruby 2008) application programming interface (API). In most cases, it is also possible to deploy pre-trained models such as (V. Badrinarayanan et al. 2017) and (Gao et al. 2021), as long as they are implemented in a supported framework. The flexibility of such services is limited, as while it is relatively easy to get started, it is difficult to efficiently incorporate ML models based on novel components, such as the fuzzy pooling layer proposed in (Diamantis & Iakovidis 2020), or complex ML-based data-processing pipelines, such as pipelines that include image preprocessing, integration of multiple heterogeneous ML algorithms with bidirectional data communication. Such pipelines are frequently met in state-of-the-art pattern analysis applications spanning a variety of domains *e.g.*, web content perception (Tian et al. 2018), obstacle detection and navigation for robotics (Zhou et al. 2017) and assistive technologies (Dimas et al. 2020), realtime analysis of medical image sequences during brain surgery (Fabelo et al. 2016) and gastrointestinal (GI) endoscopy (Iakovidis et al. 2018). However, their deployment in a SaaS context, using current ML frameworks, is far from straightforward, especially when high-throughput capacity is required. Today, to deal with this shortcoming, the implementation of such pipelines usually requires from the client to handle the communication and monitor the status of the ML components and implement preprocessing. However, this is not always possible, *e.g.*, in the case of wearable devices and other low-powered embedded systems.

There has been work towards the development of system architectures and frameworks that aim to encapsulate and abstract the usage of complex business logic for different purposes in various domains. A framework for algorithm agnostic video analysis was proposed in (Iakovidis & Diamantis 2014). In (Wang et al. 2013) a system architecture and platform, called Public-oriented Health care Information Service Platform (PHISP) was presented for personalized healthcare services and support remote health care. A system architecture and a framework for discovering content from the web using a RESTful architecture design was presented in (Fernández-Villamor et al. 2013). For managing big semantic data in real-time an architecture, called SOLID, was proposed in (Martιnez-Prieto et al. 2015). This architecture is characterized by its layered design which isolate the real-time and big data specific responsibilities. In the context of time-complemented and event-driven control models, an architecture offering modularity and flexibility of automation software was presented in (Pang et al. 2014). That architecture unifies the two models, aiming to preserve the expressiveness of event-driven programming along with the determinism of time-driven logic. In the context of Enterprise Internet of Things (EIoT), a multi-device, multi-task management and orchestration reference architecture was proposed in (Ahmad & Kim 2020). The architecture focuses the orchestration on the task-level focusing on the business process modeling of enterprise systems. Similarly for task offloading in IoT applications, an architecture named “EdgeABC” was proposed in (Xiao et al. 2020). The architecture splits the tasks into multiple subtasks based on the application workflow and then uses blockchain algorithm to ensure the integrity of resource transaction data and the profits of the resource provider. In (Kraska et al. 2013) a scalable system for ML task declaration and learning, called “MLBase”, was proposed. That system aims to make ML accessible to broad audience of users, by simplifying the declaration of ML models in a Pig Latin-like (Olston et al. 2008) declarative language and automatic ML algorithm selection. In the context of remote Machine Learning as a Service (MLaaS) (Ribeiro et al. 2015), the “PredictionIO” (Chan et al. 2013) framework, integrated a variety of ML models into a prediction service, access to which is provided using an API and a graphical user interface. In (García et al. 2020) a framework that aims to provide assistance throughout ML task lifecycle, such as training, validation and testing, was proposed with the name “DEEP-Hybrid-DataCloud”. That framework uses a standardized API that enables the functionality of the ML models to be exposed based on known semantics. In (Habiba et al. 2018) a unified component based architecture was proposed primarily focused on utility service deployment in cloud environments. The architecture focused on maximizing the availability of the deployed service with minimum configuration overhead. In (Zhou et al. 2017) a distributed architecture was proposed for motion planning of multi-robotics systems in real-time. Although such architectures can be flexible and sometimes extensible, they are tailored on domain-specific problems, limiting their scope.

ASML architecture addresses the problem of remote high throughput ML task execution involving complex data-processing pipelines. It aims to cope with well-recognized challenges (Schelter et al. 2018; Baier et al. 2019) that include the deployment of ML applications in a generic and standard way through a framework that provides the necessary level of abstraction. This framework is independent from the application domain and implementation details, such as the ML algorithms and the different programming languages used for the implementation of different components within these pipelines. To implement this framework, we propose a novel Algorithm-agnostic Scalable architecture for ML applications (ASML) that combines:

- Algorithm-agnostic architecture design, that enable arbitrary ML applications to be modeled.
- Modular design and extensible components that allow extensibility both in terms of the supported tasks and the input and output of the architecture.
- Highly scalable architecture multi-client and parallel execution task support, enabling SaaS deployment scenarios Synchronous and asynchronous task execution.

No such ML-oriented system architecture has ever been proposed, despite the emerging needs for remote artificial intelligence (AI) services in different application domains. The main contributions of the architecture include:

- It provides an answer to the open research question of how to design and implement an abstraction framework, suitable for the deployment of end-to-end ML pipelines in a generic and standard way.
- It provides technical details and application scenarios that can be used as examples for implementation of other ML application pipelines.
- It provides a performance evaluation indicating its efficiency.

To evaluate the performance and the flexibility of the proposed system architecture, we conducted

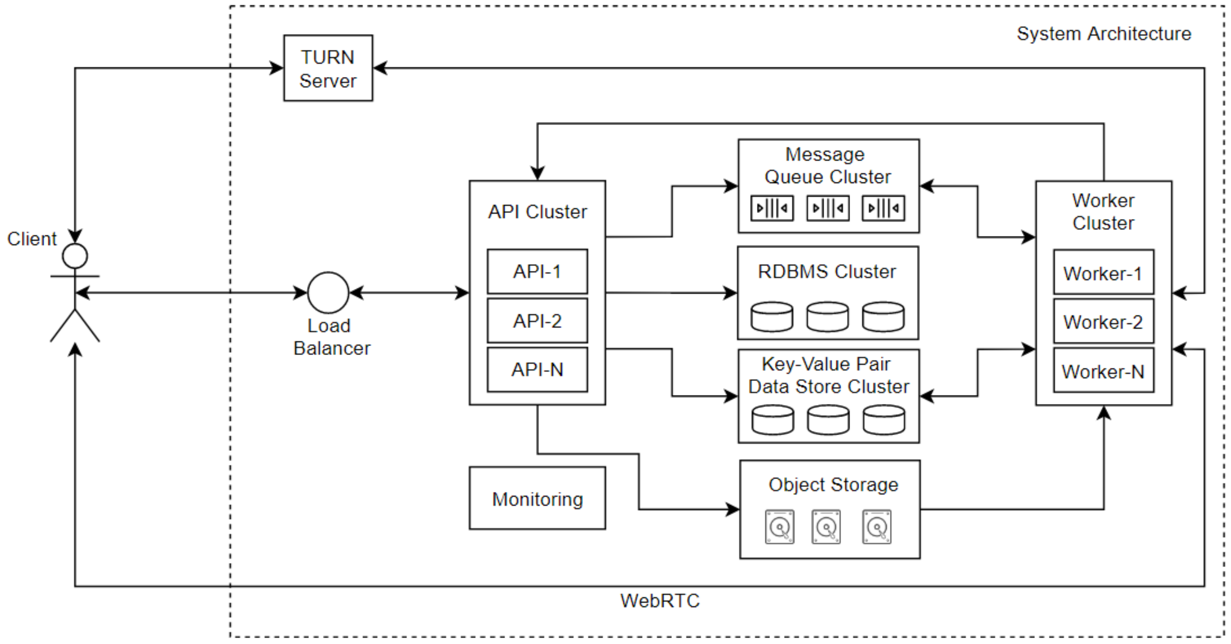


**Figure 6.1** Diagram of the ASML architecture. The monitoring module is connected to all the components of the system architecture. For readability purposes its connections are omitted.

experiments for two SaaS use case scenarios, where pattern recognition is provided as a cloud service. The first use case addresses the complex task of multi-user obstacle avoidance in the context of navigation of visually impaired individuals, using a state-of-the-art obstacle avoidance framework (Dimas et al. 2020). The second use case includes synchronous and asynchronous task execution in the context of abnormality detection in gastrointestinal endoscopy images (Iakovidis et al. 2018). It should be noted that ASML is applied for the first time for the SaaS implementation of these use case scenarios.

## 6.2 The ASML architecture

The proposed system architecture is task-oriented. A task is defined as a self-contained series of actions that is required to be completed to achieve a goal. A goal can be thought as the output of a procedure such as, image classification, object detection, object tracking etc. A task can contain multiple actions that can be executed in parallel or sequentially, depending on the goal needs. When actions are executed sequentially, the execution of the next actions is postponed until the previous ones are completed. Each action defined in the series is receiving the output of all previously completed actions, enabling complex use case scenarios to be defined. When an action in the series results into multiple outputs which are required to be processed separately by the next actions in the series, the task can create multiple tasks to parallelize the process. This is important, especially on high throughput use cases, where a significant performance improvement can be achieved by process parallelization. The input and the output of a task, including data transport, is implemented by *data source handlers*, and the actions are implemented by *processors*. Both data source handlers and processors are instantiations of reusable, abstractly defined, software components. The data of all input and output data source handlers are tagged by unique identifiers, which enable dynamic routing of the data to processors. The routing of both data source handlers and processors, is facilitated by a specialized, reusable software component, called interceptor.

Overall, the ASML architecture **(Figure 6.1)** consists of four logical components:

- The RESTful API, which acts as the entry point of the overall architecture.
- The worker, which handles the task execution.
- The task scheduler.
- The data storage and system monitoring module.

To ensure redundancy, all the components of the ASML architecture can be deployed in a cluster configuration (Bader & Pennington 2001), which considers multiple instances  per component, as illustrated in **(Figure 6.1)**. Assuming that a client has the required access rights to access and dispatch tasks to the proposed system architecture, a typical task flow can be summarized as follows. Initially the client obtains an access token using the API of the architecture. Using this token, the client can make calls to the API to create, cancel, or obtain information about a task. Historical data, such as the output of previous tasks can be obtained using the same API. When a new task is submitted by a client to the API, a record is created in the database of the system containing the task information along with meta data such as, the status of the task, the user who created it etc. In parallel, a record is created in a key-value pair data store, which is used to track the progress and other temporal data about the task lifecycle. This data store offers high throughput read/write operations and is used internally by the system as a temporal meta data storage medium instead of a conventional Relational Database Management System (RDBMS). This design decision was made because temporal data, such as the progress of a task, usually requires frequent updates (can be thousand times per second), which can degrade the time-performance of the system and increase the resource requirements, such as CPU and memory use, of a conventional RDBMS store (Li & Manoharan 2013). The task is then registered to a message queue to be delivered to a worker. At this point a response is issued to the client by the API, containing identification information about the newly created task, which can be used by the client as a reference for future requests, such as tracking the task progress etc. When the task is enqueued, a worker which monitors the message queues, consumes the task and initializes the execution. Depending on the input data source handlers, the processors and the output data source handlers, the worker unravels the task, pulls the appropriate modules and starts the execution of the task. Depending on the task configuration, the worker can communicate directly with the client, receiving and dispatching information or asynchronously inform the client about the progress and the output of the task.

When the task execution is completed the API receives a request from the worker informing about the outcome. At this stage, all temporal data are deleted and the meta data stored in the key-value pair store are permanently written to the RDBMS store.

A more detailed description of the architecture components and their interaction for the implementation of a complex data processing task is provided in the following sections.

### 6.2.1 The Restful API

The API component facilitates the communication with the clients and is the entry point of the ASML architecture. To maintain high compatibility and easy integration with most clients, the API is implemented as a RESTful HTTP service (Richardson & Ruby 2008). To enable ASML architecture to be used in a SaaS deployment, the authentication and authorization of the clients is handled by following the OAuth 2.0 (Hardt & others 2012) protocol. As a result, clients using existing OAuth 2.0 service providers can make use of the architecture without the need of providing their private credentials. Depending on the use case scenario new service providers can be added or disabled dynamically. To ensure high service availability, multiple instances of the API can run simultaneously in an HTTP load balanced environment.

The API exposes four endpoints. The first endpoint is responsible for the creation of a task. The request must contain a payload, which describe the task by identifying the processors, input and output data source handlers that will be used. Along with the payload, the request can contain other parameters, such as the desired priority of the task, the remote callback endpoints that the system will request when the status of the task changes, and flags indicating if a task should be re-processed in case of a failure.

When a task is created, a unique identifier is generated and returned to the client along with two endpoints; one that can be used to track the status of the task and one that can be used for task cancelation. The fourth endpoint can be used by a client to track asynchronously the history of all the tasks that have been created along with their output. While the protocol for the creation of the task depends on the architecture, the requests can be encoded using JavaScript Object Notation (JSON) or Extensible Markup Language (XML), depending on the content type of the HTTP request. This is done to maximize the client compatibility.

As many client applications are nowadays executed on web browsers, the API implements cross-origin resource sharing (CORS) (Van Kesteren & others 2010). Depending on the use case scenario, the API can be equipped with request quote thresholds that can be enforced on per user basis. Such thresholds can be applied on the number of requests that a user can issue within specific time period, resources allocated per user etc. This capability can be used as a pricing schema or to ensure a fair use of the system and prevent denial of service (DOS) (Garber 2000) attacks.

### 6.2.2 The Worker

A worker is an extensible component that can be thought as a handler, subscribed to one or many queues, and it consumes tasks. It is equipped with one or more input and output data source handlers and it can have multiple processors, which are responsible for the execution of several actions **(Figure 6.2)**. As a result, the capabilities of a worker and the information about the queues to which it will subscribe, are derived from the types of actions that it can process. This enables the worker to process pipelines that otherwise would be incompatible to each other, based on their software and hardware dependencies, *e.g.*, a worker could execute a pipeline with two ML processors, one capable of executing models implemented in PyTorch models and the other in Tensorflow. This battles the limitations of "MLBase"-like (Kraska et al. 2013) models, where a single framework must be used. Furthermore, the parallel processor design, enables the implementation of complex use-cases, where more than one models are used in parallel to produce results for the next processor in the pipeline, which is not possible by systems such as (Chan et al. 2013) and (García et al. 2020).

When a task is consumed, a worker initially loads the input data source handlers along with the processors and their output data source handlers and instantiates them using the parameters found in the payload of the task. An example payload with multiple input data source handlers, sequential and parallel processors is illustrated in **(Figure 6.2)**. Upon initialization, it executes the input data source handlers found in the task and passes their output to the processors identified in the payload. A processor may or may not have one or more output data source handlers, which are executed when the processor execution step finishes. In the special case where the output of one processor is needed as an input for the execution of the next one, the processor execution is delayed until all previous processors finish their execution. The output of all processors along with the initial input data source output handler is then piped to the processor as input. In the case where an action can be parallelized, the processor can create new tasks using the API component of the architecture and wait for their output. This enables the worker to use the available resources of the system, when available, and increase its throughput. The scheduling of these tasks is handled by the task scheduling component of the architecture. Considering that processors are re-usable components, not all the outputs of all previous processors are always needed. For this reason, interceptors can be used to select the input of the processors. Finally, when all processors finish their execution, the worker informs the API about the completion of the task.

In all steps of the process, the worker is updating the progress of each processor in a key-value pair database. This is used by the API when a request about the status of the task is received by a client. In some cases, a client might require intermediate information about the execution of some processors; for this reason, the worker can inform the client using remote endpoints after the execution or the failure of a processor.

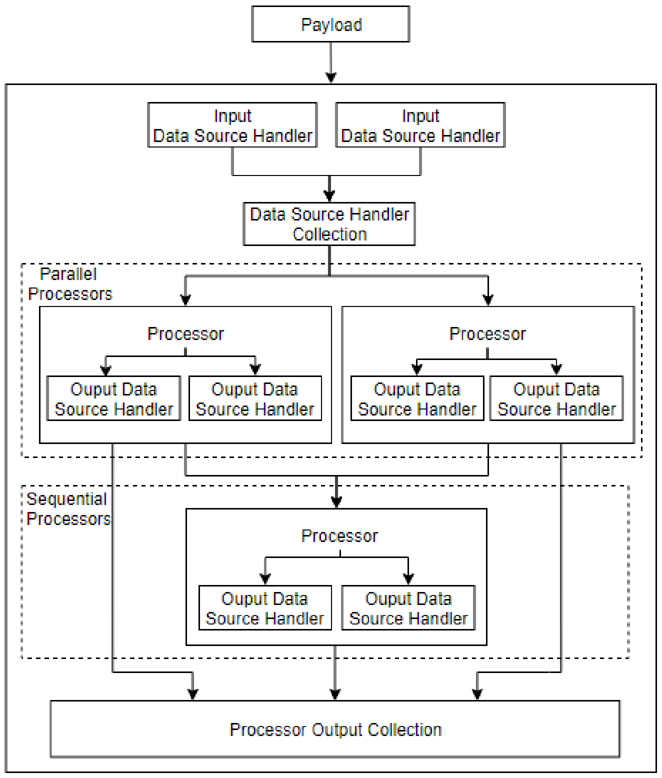

Payload
Input Data Source Handler
Input Data Source Handler
Data Source Handler Collection
Parallel Processors
Processor
Processor
Ouput Data Source Handler
Ouput Data Source Handler
Ouput Data Source Handler
Ouput Data Source Handler
Sequential Processors
Processor
Ouput Data Source Handler
Ouput Data Source Handler
Processor Output Collection

**Figure 6.2** Diagram of a worker with two input data source handlers, parallel and sequential processors with multiple output data source handlers.

While a worker can be extended to support any type of processors and data source handlers, the architecture already contains a series of predefined modules that cover most use case scenarios. The default input and output data source handlers include, HTTP, FTP, SCP, S3, Swift (Sefraoui et al. 2012) protocols along with Web Real-Time Communication (WebRTC) (Bergkvist et al. 2012) and Real-time Streaming Protocol (RTSP) (Schulzrinne et al. 1998) for real-time input and video streaming. ASML architecture is equipped with general purpose processors that enable image, audio and video processing along with ML. For example, in the case of the scenarios described in Section III, the image processor is implemented as a wrapper around the widely used ImageMagick library API. Similarly, the audio and video processor act as wrappers around the FFmpeg library API. The ML processor can be used for inference (not for training) on pre-trained ML models coming from a variety of ML libraries, including Pytorch and Tensorflow frameworks. The later supports the majority of the popular deep learning frameworks such as Tensorflow, Pytorch, CNTK and Darknet and provide variable configuration depending on the use case. The trained models can be provided to the processor using an input data source handler. For extensibility purposes the worker exposes a well-defined API and documentation that can be used for the development of new modules.

Nowadays web applications, *i.e.*, applications that run solely on web browsers, are becoming more and more common. While they offer the flexibility of running on web browsers, which most devices are equipped with, they are limited by the APIs exposed by the browser. For this, common real-time protocols such as the Real-time Messaging Protocol (RTMP) (Parmar & Thornburgh 2012) and the RTSP (Schulzrinne et al. 1998) cannot be used. Recently, web browsers adopted the WebRTC (Bergkvist et al. 2012) standard for real-time audio and video communication.

The ASML architecture supports WebRTC peer-to-peer communication between the workers and the client by using a Traversal Using Relay NAT (TURN) (Mahy et al. 2010) server, since typically, workers and clients are behind a Network Address Translation (NAT) service. Signaling between workers and the WebRTC clients is handled via WebSockets (Fette & Melnikov 2016), which is an open standard for real-time messaging. WebSockets can also be used by the workers to communicate messages to the client in real-time. Authentication and authorization to the TURN server and the WebSockets is handled by the API component of the architecture using OAuth 2.0 Bearer Tokens (Jones & Hardt 2012).

The worker module is designed so that it allows the implementation of any ML application pipeline, as new processors and new input and output data source handlers can be added. In ASML architecture, an ML pipeline implementation can be summarized into four steps:

1) Deploy the pre-trained ML model in the storage module of the architecture
2) Define which input data source handler(s) are going to provide input to the pipeline
3) Configure the ML processor to use the pre-trained model
4) Define one or many output data source handlers which are going to be used as the output of the processor.

Given an ML model created using a common ML framework, such as Tensorflow or PyTorch, the system can automatically load it and use it. Otherwise, a new ML processor should be implemented to enable support of less popular frameworks. In Section 6.3, two complex use-case scenarios are examined along with the steps followed to implement them as ASML pipelines.

### 6.2.3 The Task Scheduler

When a client creates a task using the API component, the tasks are placed in a queue and recorded in a database. The task scheduler acts as an intermediate between the task queue and the API, handling the priority in which the task will be executed. Depending on the priority and the requirements of the task, it will be placed in the appropriate queue. For queuing, the architecture uses Advanced Message Queuing Protocol (AMQP) (Vinoski 2006) compliant servers in cluster mode to ensure redundancy. The requirements of the task are derived from hardware and software dependencies of the task actions. Such requirements can be *e.g.*, the need for a GPU or for a specific software, such as FFmpeg (FFmpeg 2016). Upon the registration of a new worker, its capabilities are announced to the system through the monitoring component. The queues from which the worker consumes messages are then marked as capable of executing tasks with these requirements. Although software dependencies can be included in the list of the worker requirements, we consider that for a given ASML deployment, a subset of libraries or utilities will be available as common resources to all workers.

The priority of a task is determined by the client, upon the creation of a task, and it can be one of the following types: low, normal, high and critical. This flag indicates the urgency of execution of the task and it is used as a method to weight the task priority in the corresponding queue. In case of a critical task, the scheduler guarantees that the task will be executed immediately, whereas for the other types, the task will be executed in a first-in-first-out (FIFO) order. The scheduler performs a series of steps in order to guarantee the execution of critical tasks. Initially the scheduler tries to find an empty queue. If that fails, it communicates with the monitoring component to create and register a new worker. When the resources are saturated, the scheduler checks if a worker with proper capabilities is busy executing a lower priority task. In that case the scheduler places the task in the appropriate queue and signals the worker to halt the execution of the task which is then placed back in the queue. Only tasks with priority marked as low or normal are eligible for halting. In the unlikely event that the scheduler is unable to allocate resources for a new critical task, the task creation will fail, and an error is returned to the client.

Scheduling tasks derived from parallelized actions are considered a special case for the scheduler. These tasks are queued only when workers with the action requirements are idle, otherwise they fail before creation. This enables the workers to continue processing without waiting parallelized actions to be picked by a worker. It can also be used to create scenarios were resources become available after the parallelization, the parent worker can retry to parallelize the action at specific time intervals, this option is only available to tasks with high and critical priorities. To avoid resource stagnation, all tasks that derive from task parallelization are marked with low or medium priority depending on their parent task priority.

### 6.2.4 The Data Storage and System Monitoring Module

The architecture uses two types of data storage; one for heavy Input/Output (I/O) load use cases and one for file storage. For heavy I/O load use cases, Ceph (Weil et al. 2006) is used as a network file system mounted on all components of the architecture. For client file storage, such as pre-trained neural networks, image masks and the output of the workers, an object-store storage is used. The object-store storage enables meta-data to be saved along with the actual files. This property is used by ASML architecture to deal with the problem of  model versioning (Baier et al. 2019), by saving the model version along with the pre-trained models. The redundancy of the network file system is achieved using ZFS (Rodeh & Teperman 2003) filesystem format in RAID-Z2 (the ZFS version of RAID-6) configuration, while the object-store redundancy is handled by software.

Monitoring is integrated at the core of the proposed system architecture and it can be broken down into two categories: one specific to system load monitoring, and one for general events such as hardware failure. In both cases of monitoring, all events are stored and are accessible by the system administrator. The information gathered by the monitoring component include, CPU, memory and GPU. As all components of the architecture are deployed as containers (Rosen 2014), the load monitoring system offers the ability to activate workers or API instances on-demand, depending on the hardware available to the architecture.

The monitoring component monitors the utilization of the system resources in combination with the number of tasks queued for processing and when this number exceeds a configurable threshold, it instantiates a new worker. As tasks are prioritized based on their urgency, the monitoring component is capable of reserving hardware for the execution of critical tasks. High and normal priority tasks are eligible for additional hardware allocation when available, while tasks that are marked as low priority are not. Similarly, tasks that are created by other tasks, typically derived from the parallelization of actions, are also not considered eligible for additional resource allocation in order to avoid resource saturation. This also ensures that parallelized tasks will not get affected by the latency imposed by the hardware resource allocation, such as the virtual machine or container startup. To increase the reliability of the system, the monitoring component takes into consideration worker failures and the corresponding tasks that are processed by these workers. In the case of a failure due to hardware or network issues, the monitoring component will try to allocate new hardware resources to recover. Tasks with critical, high or normal priority and failed due to this error, are automatically prioritized to be assigned on the new allocated hardware. This is important for critical tasks with high-throughput requirements, as their service is not interrupted, while tasks with high and normal priority, experience only the initial latency of the hardware initialization.

In commercial cases, where a cloud provider is used, this feature enables cost-effective deployment, where the resources are allocated according to the real-time needs of the system. As each cloud service provider offer different access to its cloud infrastructure, the load monitoring component generates events in a form of HTTP requests to a configurable endpoint.

The object-store storage of the architecture is accessible by the clients through the API component of the architecture. Upon initial authorization for each client is created, an object container (user space) to which only the authorized client has access to.  A common use case scenario for this storage, is its use as a primary data-store in which a client stores trained ML component, such as trained neural networks, with meta-data information for the trained models. These are used primarily for version control and backward compatibility, which are important and still open issues with respect to the deployment of ML applications (Schelter et al. 2018). For this reason, the object-store component allows the client to issue signed URLs to these resources, granting access to the internal components of the architecture, with the ability to set expiration date for them (TTL). The monitoring component, records access to the object- store storage and all requests, enabling limiting access or bandwidth use of the store configurable on a client base manner.

The described ASML architecture is generic, suitable for the time-efficient SaaS deployment of ML-based data processing applications. In the following section, its capabilities are experimentally demonstrated with two contemporary use cases, where its capacity for real-time video processing is evaluated.

## 6.3 Example Use Cases and Evaluation

To demonstrate the effectiveness of the ASML system architecture, we conducted experiments on two different ML-based data processing use cases.  The first use case considers the problem of obstacle detection and recognition in the context of an assistive system for navigation of Visually Challenged (VC) individuals. Considering that such a system is meant to be used by people with disabilities, it must be accurate, fast, and reliable. The second use case considers the problem of abnormality detection in gastrointestinal (GI) tract videos, in an effort to provide a solution for real-time assistance to the physicians during GI endoscopy. When the endoscopic modality used does not require real-time streaming capabilities, such as in the case of WCE (Iakovidis & Koulaouzidis 2015), we demonstrate how the same processor can be used to process the videos asynchronously.

### 6.3.1 Realtime Obstacle Detection, Recognition and Tracking

Recently we presented a methodology for the detection and recognition of obstacles, and evaluated its effectiveness (Dimas et al. 2020). We consider this as an indicative scenario to show how such a methodology can be implemented using the generic ASML architecture described in Section 6.2, how the ASML-based implementation can be extended by incorporating an obstacle tracking algorithm, and we assess its efficiency and scalability.

### 6.3.1.1 Obstacle Detection and Recognition

The methodology presented in (Dimas et al. 2020), considers that color RGB-Depth (RGB-D) image streams are captured using a stereoscopic camera. Each image is processed by two parallel components and their results are aggregated to determine image regions where high-risk obstacles are located. The first component uses a GAN (Pan et al. 2017) to generate human eye fixations that highlight salient image regions. The second component uses the depth channel of the RGB-D image, to compute three risk maps, representing high, medium and low risk obstacles, based on fuzzy logic. Following the fuzzy aggregation of the outputs of these components, the resulting sub-images corresponding to obstacle regions are provided to a CNN, called Look Behind Fully Convolutional Neural Network (LB-FCN) *light* (Diamantis et al. 2019), to perform the obstacle recognition step. The processing steps required by the methodology (Dimas et al. 2020), are illustrated in **(Figure 6.3)**.

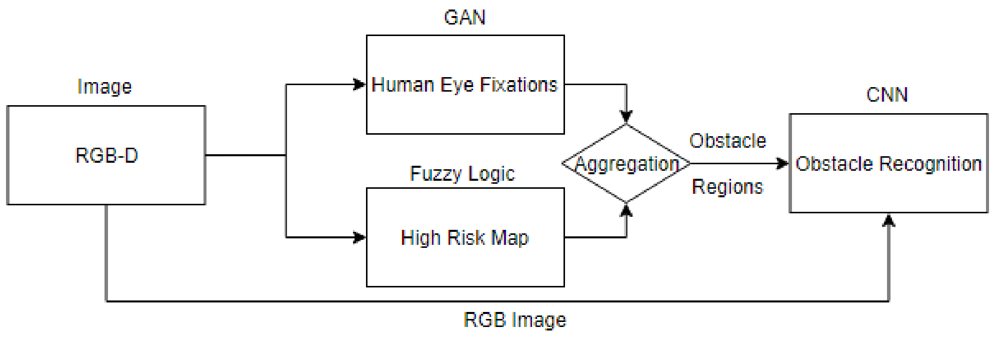


**Figure 6.3** Illustration of the processing steps followed in (Dimas et al. 2020) for obstacle detection and recognition.

The two deep learning inference steps described, are performed on a GPU-enabled server, remotely accessible for the navigation of the VC individuals. Each individual, carries a mobile phone-based wearable system, running a lightweight client application that performs image acquisition and communication with the server. Considering that not all users use the same mobile phones and that native applications are platform-dependent, as a client we considered a conventional web browser. To enable real-time streaming between the client and the system, we used WebRTC capabilities of the architecture, as it is natively supported by all major web browsers. Similarly, for real-time messaging, we used WebSockets to communicate the output of the obstacle detection and recognition back to the client. For the implementation of the obstacle avoidance schema, initially a new processor is required that splits the original RGB-D image into two parts; one with the RGB channels, and one with only the depth information. For the salient region detection and obstacle recognition, the ML processor of the proposed architecture can be used directly; thus, only the pre-trained networks need to be deployed on the object-store storage of the system. For the high-risk map generation, a new processor is required that takes as an input the depth channel of the RGB-D image and computes the high-risk map. For the detection of obstacle subimages, a processor was added following the methodology proposed (Dimas et al. 2020), which takes as input the output of the second the third processor and performs the aggregation **(Figure 6.4, step 4)**. Finally, the obstacle regions along with the RGB image is piped to the ML processor which performs the obstacle recognition **(Figure 6.4, step 5)**. The worker configuration with the corresponding processors, input and output data source handlers is illustrated in **(Figure 6.4)**. Interceptors are used to select the RGB and depth channels for the ML and high-risk map processors. The pre-trained ML models are provided to the worker as input data source handlers from the object-share storage of the architecture, the selection of which is performed using interceptors.

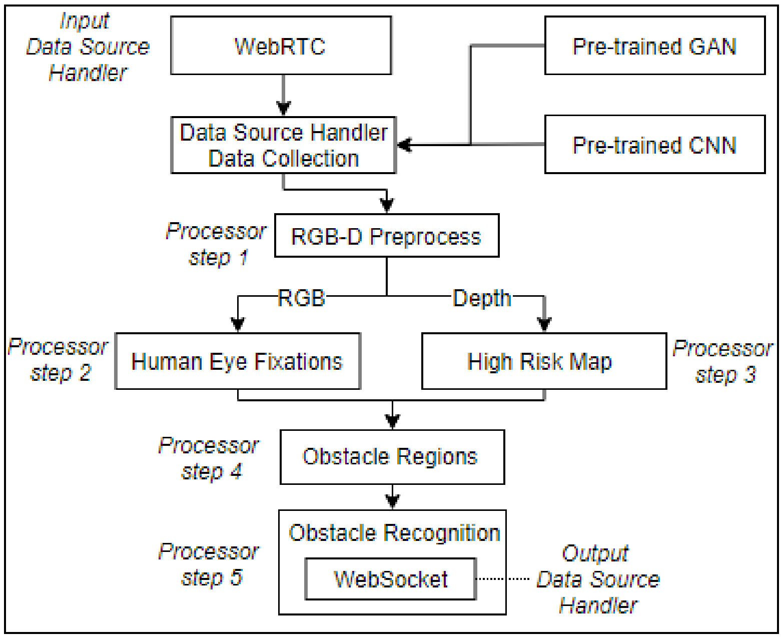


**Figure 6.4** Diagram of a worker implementing the steps required by the obstacle detection and recognition framework (Dimas et al. 2020).

In this use case scenario, the performance of the worker is highly dependent on the number of obstacle regions found by the aggregation processor. As a result, in images where multiple obstacle regions are identified in a single image, the performance of the worker drops exponentially **(Figure 6.5)**. For this reason, we considered parallelization of the object recognition component **(Figure 6.4, step 5)**. To achieve this, the obstacle region processor creates a new task for each detected obstacle region and submits it to the API component for processing, effectively performing the same operation as a client would do. The output of each object recognition task is received by the worker directly, using an HTTP request, produced by the output data source handler of each object recognition task. Each request contains the label of the recognized object along with an identification number which is used by the processor to identify the obstacle region to which the label belongs to. Accounting for the latency that is introduced by the network, this enables higher frame rate compared to the conventional approach where all the information is processed on a single instance.

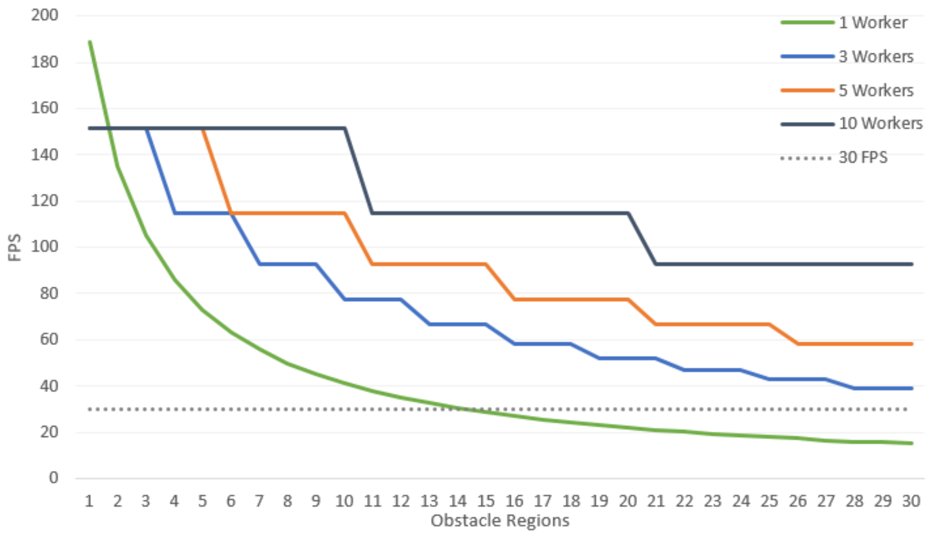


**Figure 6.5** Comparison of frame rate achieved by the proposed architecture using different number of workers and number of obstacle regions. The dotted line illustrates the real-time performance threshold of 30 frames per second (fps).

To perform our experiments and evaluate the performance of the proposed architecture, we used a typical smartphone device with 4 ARM based CPUs and 2 GB of RAM each one paired with Intel RealSense (Keselman et al. 2017) D435i RGB-D camera. To maximize mobile cross-platform compatibility, our experiments were conducted using Google Chrome web browser as the client. For the deployment of the architecture, we used virtualization and more specifically containers through Docker (Merkel 2014). For the HTTP load balancing we used NGINX (Reese 2008) to distribute the incoming requests on multiple instances of API components deployed on lightweight containers. As RDBMS we used master-master deployment of MariaDB. Caching and message queueing was implemented using Redis and RabbitMQ (Dossot 2014) running in cluster mode respectively. The API and monitoring components, the TURN server and the workers are implemented in Go programming language.

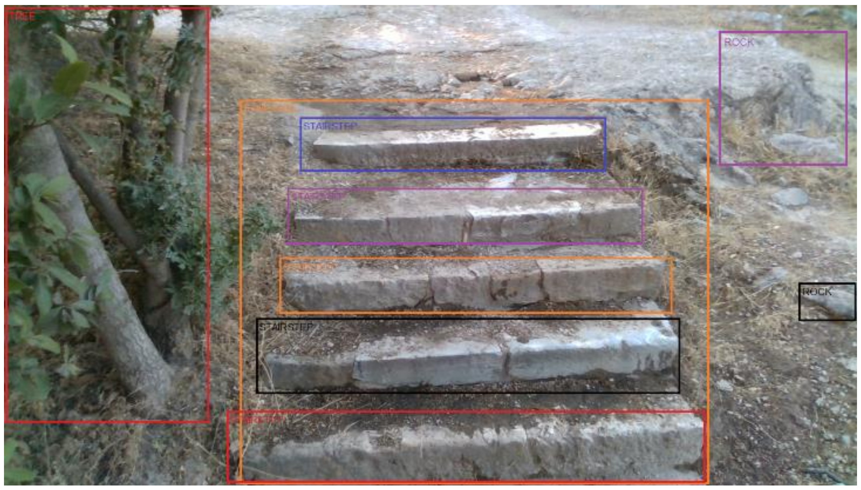


**Figure 6.6** Classification example of 8 obstacle regions using the proposed architecture with 5 workers. Each color corresponds to a different worker.

To demonstrate the performance improvement that the proposed system architecture offers compared to conventional synchronous approaches, we conducted four experiments using different number of workers. In all experiments the workers infrastructure was equipped with an NVIDIA GTX-1080 TI GPU. The implementation of (Dimas et al. 2020) deep learning algorithms was performed using the TensorFlow framework, which enables the ML algorithms to be executed on GPUs, significantly increasing the performance. The results of the frame rate achieved by the proposed architecture using different number of workers are illustrated in **(Figure 6.5)**. The single worker experiment demonstrates the performance of a conventional deployment without the use of the proposed architecture. On average, 3.2ms are required for the obstacle detection task while the obstacle recognition requires 2.1ms for each obstacle region. We found that in a typical scenario, the obstacle region processor detects 12 objects on average per image. Any increase in that number can significantly decrease the performance of a single instance deployment. A visual representation of the parallelized obstacle region classification procedure is illustrated in **(Figure 6.6)**. The performance improvements that the proposed ASML architecture offers become even more apparent when multiple clients are required to be processed in parallel. In this case, a single instance deployment would have been insufficient and would require scaling, which the proposed architectures offer. Communication between multiple workers introduce network latency which can degrade the overall performance. In our experiments, using intranet connections, the network latency was on average 1.3ms. Using the ASML architecture, **(Figure 6.5)** shows that when more than 3 obstacle regions are required to be recognized, the overall performance improvement overcome the network latency.

### 6.3.1.2 Obstacle Tracking

In the context of an obstacle avoidance application, tracking of the objects found in previous frames is important as it can be used to avoid re-informing the user multiple times for the same obstacle, or compute the trajectory of a moving object, such as a vehicle. The methodology presented in (Dimas et al. 2020) does not include tracking; however, several single (Fu & Xu 2019) and multiple (Ciaparrone et al. 2020) object tracking methodologies have been proposed over the years, which can be introduced to the system as an additional processing step. These approaches usually rely on conventional handcrafted features (such as color, shape and texture), deep learning (Fiaz et al. 2019), or follow the tracking-by-detection (Bochinski et al. 2017) approach. In tracking-by-detection methodologies, the state of the algorithm, which contains the history of the detected objects from previous frames need to be preserved in order to be compared with the detected objects of the current frame. This can be challenging when this state must be shared across multiple workers.

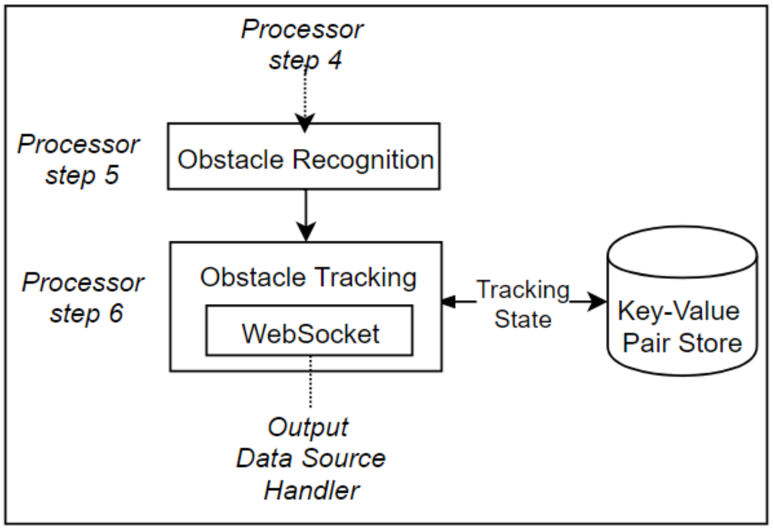


**Figure 6.7** Diagram of a worker extending the obstacle detection and recognition framework (Dimas et al. 2020) methodology with and object tracking (Bochinski et al. 2017).

To overcome this, ASML architecture uses the key-value pair store to share the state across multiple workers. To demonstrate that, we enhanced the obstacle detection methodology proposed in (Dimas et al. 2020) to include obstacle tracking by following the approach proposed in (Bochinski et al. 2017). We followed this approach as it has minimal computational footprint and it relies on detection algorithms with high frame rates, such as the one used in (Dimas et al. 2020). The algorithm of (Bochinski et al. 2017) relies on the fact, that in high frame rate scenarios consecutive frames have significantly overlapping detections. When the intersection-over-union

(IOU) of the consecutive detections is lower than a certain threshold, the detection belongs to the same object.

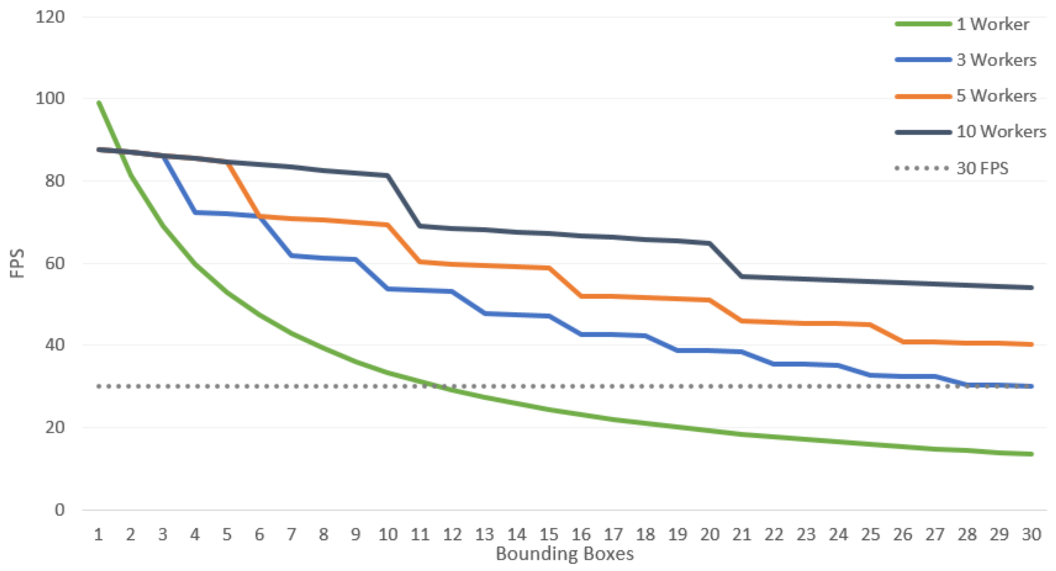


**Figure 6.8** Comparison of frame rate achieved with object tracking by the proposed architecture using different number of workers and number of obstacle regions. The dotted line illustrates the real-time performance threshold of 30 frames per second (fps).

To include the method proposed in (Bochinski et al. 2017) in the obstacle detection pipeline, a new processor was created **(Figure 6.7, step 6)**. This processor accepts as input the detected obstacles and their corresponding classes and outputs their tracking information. This is illustrated in **(Figure 6.7)**. As processors are stateless, the previously computed tracking information, are stored in the key-value pair store. The overhead of this is minimal (on average 4.7ms per frame). This includes the data transmission time, deserialization and post processing data serialization. **(Figure 6.8)** demonstrates how the performance (FPS) is affected when a different set of workers is used in comparison to the number of bounding boxes that need to be tracked.

### 6.3.2 Realtime Multi-User Endoscopic Video Analysis

In the context of computer-aided detection of abnormalities in GI endoscopy, we used the state-of-the-art LB-FCN deep CNN architecture, pre-trained to detect abnormalities in flexible endoscopy (colonoscopy) videos and Wireless Capsule Endoscopy (WCE) (Diamantis et al. 2019; Diamantis et al. 2018). To use this model in the proposed architecture the ML processor of ASML was utilized. Considering that the CNN architecture requires an input with spatial dimensions 224×224 pixels, the video frames received as input, have to be resized accordingly. For this reason, an image processor was used to resize the video frames to the required size using zero-padding to prevent the aspect ratio as suggested in (Diamantis et al. 2019).

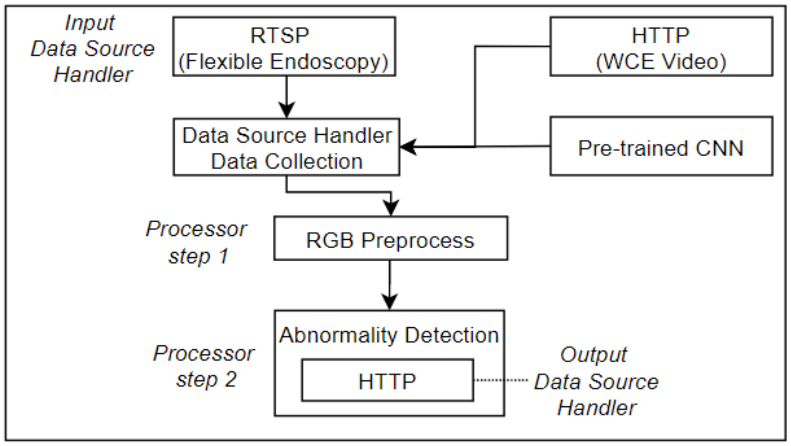

Input Data Source Handler
RTSP (Flexible Endoscopy)
HTTP (WCE Video)
Data Source Handler Data Collection
Pre-trained CNN
Processor step 1
RGB Preprocess
Abnormality Detection
Processor step 2
HTTP
Output Data Source Handler

**Figure 6.9** Diagram of a worker implementing the steps required for the flexible endoscopy and WCE abnormality detection. In flexible endoscopy, the RTSP input data source handler is used while in WCE, the HTTP input data source handler is used to provide the video from the object-store storage.

The same CNN architecture can be used in both flexible endoscopy and WCE abnormality detection applications. In the latter case the videos are not usually streamed in real-time; instead they are obtained only after the capsules are excreted from the GI tract. For this reason, in the case of flexible endoscopy RTSP input data source handler is used, while in the case of WCE the video is stored in the object-store storage and provided to the worker via the HTTP input data source handler. In both cases, when an abnormality is detected by the ML processor, an HTTP request is sent, informing the client about the detection and the frame at which the abnormality was detected. As multiple medical instates can be benefited by such a service, in both cases we considered a SaaS cloud deployment of the ASML, where multiple physicians can access it simultaneously **(Figure 6.9)** illustrates the worker configuration. **(Figure 6.10)** includes samples of WCE image classification results, from the KID (Koulaouzidis et al. 2017) dataset, obtained using the proposed ASML architecture.

To evaluate the real-time performance of the proposed ASML architecture, multi-user experiments were conducted using different number of workers on the problem of flexible endoscopy abnormality detection. In our experiments, on average 2.8ms were required for frame classification using a single worker. **(Figure 6.11)**, shows that when more than 12 endoscopes are streaming in parallel, such a singular deployment becomes insufficient. ASML is capable of scaling horizontally by increasing the number of workers according to the required number of streams and the resources available to the architecture. This can be proven particularly useful in the case of a

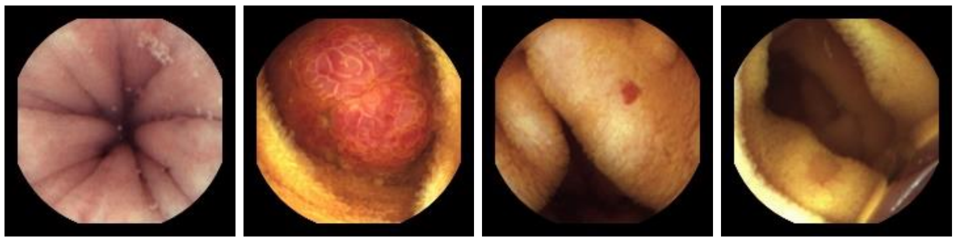


**Figure 6.10** WCE frame classification using the proposed architecture. (a) Normal, (b) Polypoid, (c) Blood, (d) Inflammatory condition.

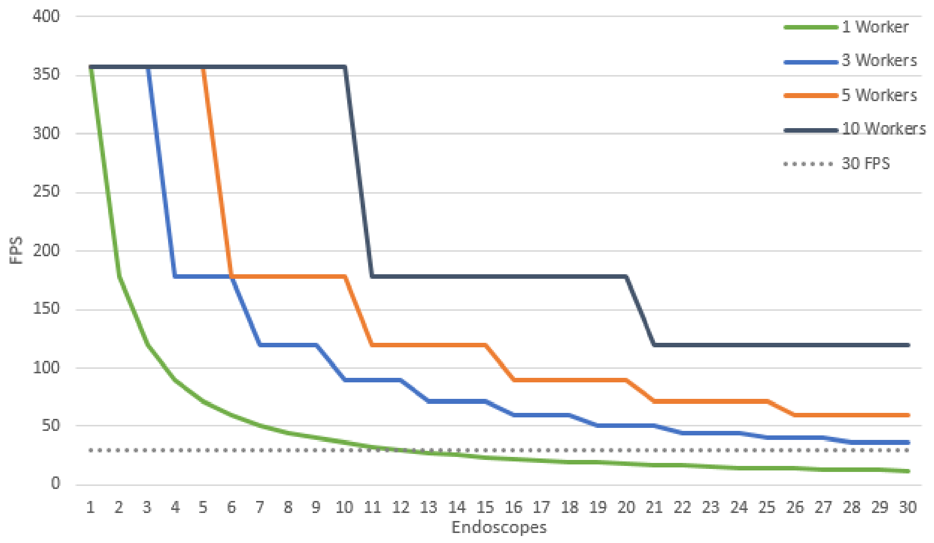


**Figure 6.11** Comparison of frame rate achieved by the proposed architecture using different number of workers and number parallel endoscopes. The dotted line illustrates the real-time performance threshold of 30 fps.

SaaS deployment, where abnormality detection is offered as a service to multiple clients, *e.g.*, several clinics with one or more endoscopy units using the abnormality detection service.

## 6.3.3 User Case Response Time Analysis

To demonstrate the performance of the proposed architecture design we conducted two experiments based on the presented use cases. In both experiments three workers were used to measure the system’s response times over a period of time. In the case of obstacle detection, recognition and tracking **(Figure 6.12)**, the experiment recorded the average response time of the workers and the number of bounding boxes found at a 15 second sampling interval, over a period of 30 minutes. To show the behavior of the system in the case of a worker failure, different workers were shortly removed from the system, at different times. **(Figure 6.12)** shows that upon a worker failure, one of the running workers is successfully compensating for the loss, with the expected expense of higher average response time, due to the extra computation overhead. In total the average response time of the system was 28.1ms, and the maximum and minimum response times 48.2ms and 22.1ms respectively.

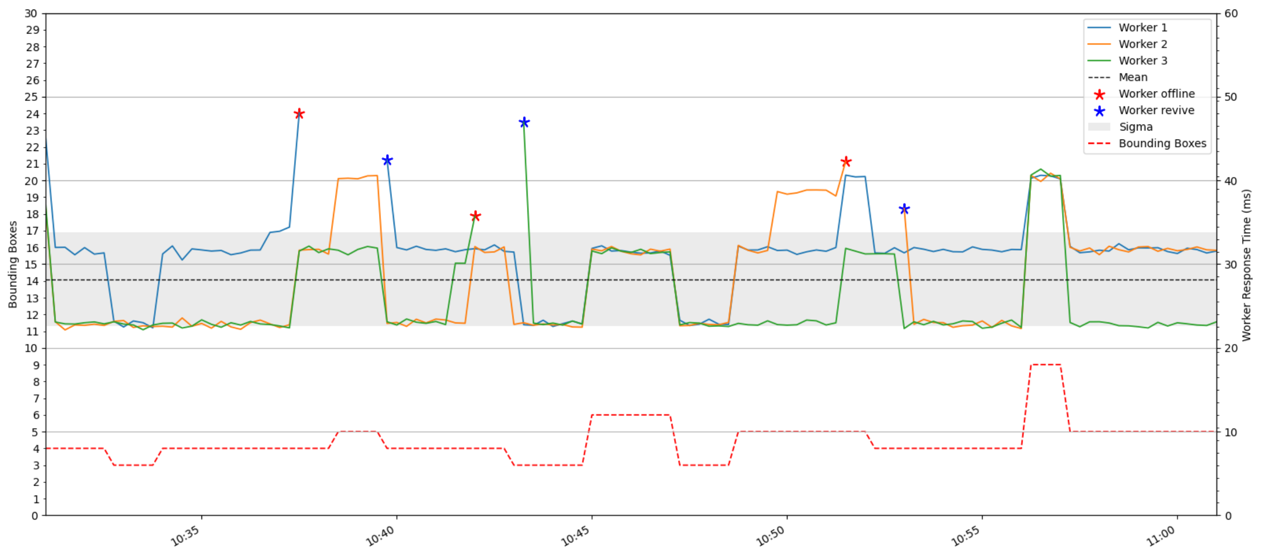


**Figure 6.12** Comparison of worker response times detecting and tracking different number of bounding boxes corresponding to obstacles, sampled over a period of 30 minutes. The graph illustrates how the system behaves when a worker is removed or added back to the system. The sampling points are linearly interpolated for visualization purposes.

A similar system behavior can also be observed in the second use case scenario. For the abnormality detection in GI tract images, the experiment measured the average response time of the workers when used by different number of users with a sampling interval of 1 minute for 2 hours. The average response time of the workers was 3.6ms, and the maximum and minimum response times were 6.1ms and 2.4ms, respectively. When each of the two workers was removed from the system for a short period of time, the system had the same behavior as in the first use case. The response time fluctuations that are observed in both use cases **(Figures 6.12, 6.13)**, can be attributed to the network latency, whereas the higher response times are due to worker initialization.

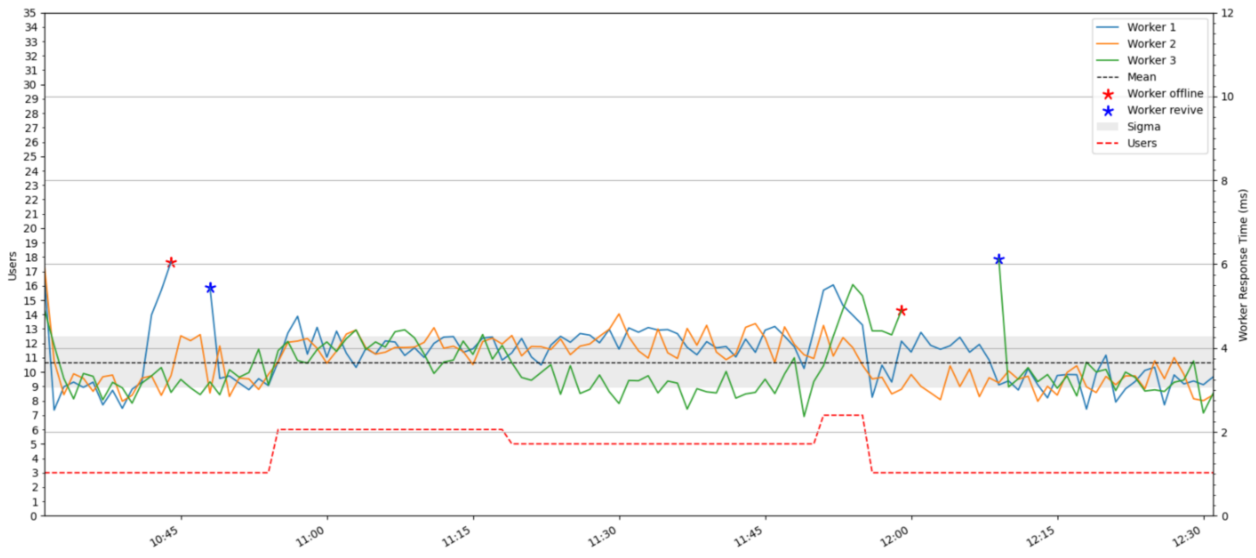


**Figure 6.13** Comparison of worker response times for abnormality detection in GI tract images, when used by different number of users, sampled over a period of 2 hours. The graph illustrates how the system behaves when a worker is removed or added back to the system. The sampling points are linearly interpolated for visualization purposes.

## 6.4 Discussion

There has been work towards scalable system architectures and application frameworks that aim to provide scalable task execution. When it comes to ML, the deployment of such systems tends to be complicated and usually coupled to specific domains and use cases. The lack of an abstraction framework for the whole ML pipeline and need for a generic and standard deployment approach has been highlighted in the recent literature (Schelter et al. 2018)(Baier et al. 2019). Architectures such as (Wang et al. 2013) and (Fernández-Villamor et al. 2013), although scalable, are domain specific and thus they do not allow arbitrary ML task declaration and execution. The needs of ML task execution are also not satisfied by the generic architecture proposed in (Ahmad & Kim 2020) as it focuses on task orchestration in EIoT applications which limits the scope to periodic or event-driven task modeling and it does not include an abstraction framework that can be used as a standard solution for ML pipeline task modeling. Although the declarative ML task execution system “MLBase” (Kraska et al. 2013) has the advantage of automatic scaling, the platform is coupled with a specific ML framework and language (Nguyen et al. 2019), which is a limitation for use in production environments. The “DEEP-Hybrid-DataCloud” framework proposed in (García et al. 2020), although it satisfies the needs for a generic ML task deployment and execution environment, it does not offer flexibility in terms of generic ML pipeline modeling, as models with non-standard functionality exposure semantics are not compatible. Furthermore, the architecture presented in (García et al. 2020) does not include any standard ML task input and output handling.

To cope with these issues in the context of remote, high throughput ML task execution, involving complex data-processing pipelines, we proposed ASML as a novel algorithm-agnostic and platform-independent system architecture. The architecture achieves that by:

- Providing a standardized, extensible and unified algorithm-agnostic task-oriented pipeline framework, with interchangeable platform independent processing units;
- Handling the pipeline execution in a highly scalable system architecture;
- Task scheduling for task parallelization.

Access to the architecture is provided through a RESTful API, enabling platform independence. This in combination with the use of open technologies, such as WebRTC, enables thin clients, such as a web browser, to use the system without special software requirements.

The results obtained from the deployment of two state-of-the-art SaaS ML application scenarios indicated that the ASML architecture is suitable for high throughput applications in different domains. The extensibility of ASML architecture was also investigated, where in the first use case the obstacle detection pipeline was extended to include obstacle tracking by the addition of an object tracking processor. There are several other domains where ASML architecture is applicable, including robotics, transportation, and security, *e.g.*, for SaaS deployment of ML-assisted navigation of autonomous robots and vehicles, and recognition of suspicious patterns from multiple surveillance cameras.

As a limitation of the proposed architecture, one could consider its inability to automatically identify the software requirements of each task, which can result into dependency issues. To solve this problem, the tasks are required to include labels to indicate which software dependencies are required by the task. Another limitation of the architecture is that processors are considered as black boxes and only the workers are informed about what input and output can be accepted. As a result, the API has no way of knowing if an input or output data source handler is compatible with a declared processor. This can result into situations where invalid tasks are successfully created and queued to the system, failing later, when they are picked by a worker. Although this can be resolved by including documentation for each processor, we plan to include automation validation prior task execution.

The ASML architecture can utilize platforms specifically designed for parallel process execution in a multi-host environment, such as Spark (Zaharia et al. 2010); however, configuring these platforms still requires advanced technical knowledge and skills. To cope with this issue, within our future research prospects is to extend the proposed system architecture to include automated host clustering, enabling parallel task execution without special configuration. An open access implementation of the proposed system architecture is planned for distribution to the wider research community.

# CHAPTER 7

# Conclusions and Future Research Directions

This doctoral dissertation investigated and developed novel DNN architectures and DNN-based image analysis and synthesis methodologies, in the context of intelligent systems and services with applications in biomedicine and assisted living. The presented work included extensive experimentation for benchmarking and validation on large datasets, most of which are publicly available. This chapter summarizes the conclusions of this research, and indicates directions for future research.

## 7.1 Concluding Remarks per Chapter

***Chapter 2*** provided the theoretical knowledge that was the basic background for the understanding of the methods that were described in the rest of this thesis' chapters along with detailed and literature review of most impactful deep learning models, including the CNNs and GANs.

***Chapter 3*** introduced a novel fuzzy pooling (Diamantis & Iakovidis 2020) operation for CNN architectures, coping with the uncertainty of feature values. Experiments performed on publicly available datasets, show that the proposed methodology significantly increases the classification performance of CNNs, as compared to other state of-the-art pooling approaches. We showed that fuzzy pooling can be used as a drop-in replacement of existing pooling layers, in CNN architectures, increasing the generalization performance. Furthermore, experiments conducted on standard image datasets (Anon n.d.; Gonzalez & Woods 2018), showed that the proposed methodology is able to preserve better the important features of the pooling areas. This was validated both visually and statistically by the higher classification performance obtained using the fuzzy pooling approach.

***Chapter 4*** presented the novel contributions of this thesis in the context of computer-aided endoscopy. More specifically,

Section 4.1 presented a novel CNN architecture (Diamantis et al. 2019), named Look-Behind Fully Convolutional Neural Network (LB-FCN), to deal with the problem of computer-aided human GI tract image classification. To the best of our knowledge none of the existing deep neural network architectures combined in the same way multi-scale feature extraction along with look-behind connections. The overall conclusions that can be derived about LB-FCN architecture can be summarized as follows:

- It has a simplified design following the FCN (Springenberg et al. 2014) architecture approach.
- Its relatively low number of free parameters along with its multi-scale feature extraction capability enables efficient training with either smaller or larger datasets.
- It outperforms state-of-the-art architectures and methods in the detection of different types of abnormalities in images obtained from different endoscopic modalities, including gastroscopy and WCE.

Section 4.2 presented an investigation in the context of cross-dataset abnormality detection in endoscopy images using a novel CNN architecture named Multi-scale Feature extraction CNN (MFCNN) (Diamantis et al. 2018) which was a predecessor of LB-FCN architecture. The generalization capabilities of MFCNN architecture was evaluated on multiple publicly available gastrointestinal image datasets validating that it is able to generalize well even when the training and testing datasets are significantly different.

Section 4.3 investigated multi-label classification methods for a richer semantic interpretation of endoscopy video frames (Vasilakakis et al. 2018). The rationale behind this was that the classification of the video frame contents into multiple semantic categories, could simplify the detection of contents corresponding to abnormalities especially since the presence of intestinal content, such as debris and bubbles, is dominant in parts of the GI tract (Iakovidis & Koulaouzidis 2015). In this context a novel CNN architecture was presented, named “MM-CNN”, for multi-label WCE image classification. The results validate that the effect of using multiple labels can enhance abnormality detection, with MM-CNN achieving higher classification accuracy when compared to state-of-the-art.

Section 4.4 presented a novel approach (Diamantis et al. 2019) to cope with the problem of small number of training data availability in the medical domain. We showed that data generation using non-stationary texture synthesis technique can be used effectively to generate small bowel wireless capsule endoscopy images with and without inflammatory conditions. Furthermore, we explored the generalization performance of the state-of-the-art LB-FCN *light* (Diamantis et al. 2019) architecture trained on fake, artificially generated, images and evaluated its performance on real WCE small bowel images. One could argue that training on synthetic images may create fault detections on real cases. This is likely to happen; however, classifiers create fault detections even when they are trained with real data. Our experiments assessed the capacity of the state-of-the-art LB-FCN *light* classifier to create such fault detections. We have trained the classifier separately using synthetic and real images, and assessed its classification performance on real images. The results showed that the training of the LB-FCN *light* with real images yields better classification performance, than training with artificially generated images. However, using fake images for training the classification performance obtained was comparable with the performance of state-of-the-art approaches based on 522 handcrafted features. To date there is no classifier creating zero false detections on the dataset used in this study. The most important conclusion that can be derived from this study is that it is feasible to substitute real medical images with fake ones and obtain useful, medically relevant, results. Therefore, the medical data providers could use the proposed medical image generation methodology to provide realistic datasets to the information technology scientists without ethical and legal constraints. Today, it is well-known that such constraints are responsible for delays of relevant projects, *e.g.*, by waiting for approvals from ethical committees. This situation is also responsible for the limited public data availability, which, as it has been already pinpointed in the past (Iakovidis & Koulaouzidis 2015), it is also limiting for the essential progress in the research for computer-aided medical decision support system.

***Chapter 5*** presented the novel contributions of this thesis in the context of assistive navigation of visually impaired. Section 5.1 presented a novel lightweight CNN architecture (Diamantis et al. 2019) , named Look-Behind Fully Convolutional Neural Network *light* (LB-FCN *light*), and examine the generalization capabilities of the network in the context of staircase detection in natural images (Section 5.3). To evaluate the performance of the architecture we extended the LM+Sun (Tighe & Lazebnik 2010) natural image dataset with staircase images obtained from Flickr (Flickr Inc. 2019) social network. To the best of our knowledge there has been no existing work in this field that utilize solely weakly-labeled images to detect staircases in the natural images. The key features of the proposed LB-FCN light architecture can be summarized as follows:

- It has a relatively low number of free parameters requiring an also low number of FLOPs, which makes it suitable to be used on mobile and embedded devices.
- It features multi-scale feature extraction design allowing the architecture to detect staircases of various sizes and under difficult conditions, such as natural images.
- Following the FCN (Springenberg et al. 2014) architecture approach it offers a lightweight and logically unified design.
- Compared to MobileNet-v2 (Sandler et al. 2018) network, the proposed architecture offers a relatively lower number of FLOPs and free parameters and a slightly higher detection performance. This makes it attractive for lower-end mobile and embedded devices.

Section 5.2.4 presented an application of LB-FCN *light* architecture in which the model was used as the object recognition component of a two-stage object detector in the context of navigation of visually impaired individuals (Dimas et al. 2020). To train the network, a new dataset was created, named “Flickr Obstacle Recognition” dataset, containing RGB outdoor images from five common obstacle categories (benches, columns, crowd, stones, and trees). A novel system architecture was also presented which enables the computationally intensive object detection and recognition components of the assistive navigation system to be executed on the cloud, in an effort to achieve high inference throughput.

Navigation assistive systems consist of complex pipelines that include object detection, recognition, and tracking, along with path planning and risk assessment algorithms. Assessing and optimizing the performance of such pipelines is a relatively complex and time-consuming task, especially when it is performed in real-world environments, where parameters such as the weather and lighting conditions may vary. The recent COVID-19 pandemic has raised an additional problem, since it has been impossible to assess algorithms that require real-world outdoor measurements, mainly due to the social distancing rules and quarantine measures. Aiming to address this problem Section 5.3 presented a novel Digital Twin simulation and evaluation framework for assistive navigation systems. More specifically, it presented a novel framework that enables the assessment and optimization of navigation assistive systems for visually impaired individuals in a controlled virtual reality environment. The proposed framework is able to simulate real-world environments obtained using Google Earth or Open Street Maps from both mapped and unmapped regions of the world. The DT framework is integrated with a VR repository in which assets, such as 3D objects and existing simulated worlds, can be stored and shared with peers, facilitating the virtualization process. The integrated obstacle avoidance framework is specifically designed for complex, algorithm-agnostic, navigation pipeline assessment. It enables algorithms to be evaluated in both simulated and real-world environment since it is implemented using the Python programming language.

***Chapter 6*** presented a novel algorithm-agnostic and platform-independent architecture for scalable ML named "ASML" (Diamantis & Iakovidis 2021). While there has been work towards scalable system architectures and application frameworks that aim to provide scalable task execution, when it comes to ML, the deployment of such systems tends to be complicated and usually coupled to specific domains and use cases. The lack of an abstraction framework for the whole ML pipeline and need for a generic and standard deployment approach has been highlighted in the recent literature (Schelter et al. 2018; Baier et al. 2019). ASML architecture aims to cope with these issues in the context of remote, high throughput ML task execution, involving complex data-processing pipelines. The architecture achieves that by:

- Providing a standardized, extensible, and unified algorithm-agnostic task-oriented pipeline framework, with interchangeable platform independent processing units.
- Handling the pipeline execution in a highly scalable system architecture.
- Task scheduling for task parallelization.

Access to the architecture is provided through a RESTful API, enabling platform independence. This in combination with the use of open technologies, such as WebRTC, enables thin clients, such as a web browser, to use the system without special software requirements. The results obtained from the deployment of two state-of-the-art SaaS ML application scenarios indicated that the ASML architecture is suitable for high throughput applications in different domains. The extensibility of ASML architecture was also investigated, where in the first use case the obstacle detection pipeline was extended to include obstacle tracking by the addition of an object tracking processor. There are several other domains where ASML architecture is applicable, including robotics, transportation, and security, *e.g.*, for SaaS deployment of ML-assisted navigation of autonomous robots and vehicles, and recognition of suspicious patterns from multiple surveillance cameras.

## 7.2 Overall Conclusions

The work presented in this dissertation has identified several research challenges, and it provided solutions on related open issues. The overall conclusions derived from the presented DNN architectures, methodologies and applications can be summarized as follows:

- Fuzzy pooling can be used to tackle the uncertainty that is naturally propagated from the input layer to the feature maps of the hidden layers through convolutions. Tackling with such uncertainty improves the quality of features selected by the CNN pooling layers, and contributes to the overall improvement of the generalization performance of the trained network.
- The LB-FCN *light* paradigm indicates that CNNs with a low number of free parameters combined with multi-scale feature extraction and residual learning, can generalize well, even when the training samples are limited.
- It is feasible to use GANs to substitute real medical images with synthetic ones, which can be used for CNN training, especially when real data are limited, and obtain useful, medically relevant, results.
- Remote intelligent image and video analysis services have become possible with ASML, in a generic and standardized way, even with a near-real time performance.
- Using digital twins for system testing can be an effective tool, especially when real-world experimentation is difficult or costly.

## 7.3 Future Plans and Research Directions

Considering the advancements in the context of deep learning, the proposed methods can be further evolved and extended in respect to both their efficiency and effectiveness.

In respect to the Fuzzy Pooling methodology presented in (Diamantis & Iakovidis 2020), future work includes optimization of the current implementation to fully exploit GPU-level parallelism. This will enable us to perform larger-scale experimentation with very large datasets, such as ImageNet (Deng et al. 2009) using deeper CNN architectures, such as (Simonyan & Zisserman 2014). Other interesting research perspectives include the extension of the learnable set of network parameters to include the parameters for the fuzzy rules, and the extension of the proposed approach using generalized fuzzy sets, such as intuitionistic fuzzy sets.

The methodologies presented in Chapter 4, although they have been evaluated in the context of biomedical applications and more specifically in the context of computer-aided endoscopy, can be used in wide range of applications such as natural image analysis. Topics such as coping with the large number of free parameters and overfitting are still open in deep learning applications. Towards these issues, we plan to apply systematic experiments on architecture variations of LB-FCN while applying it to larger, even more diverse datasets of human GI tract images. Furthermore, the future research directions is to extend the proposed architecture to enable localization of the abnormalities through supervised learning using weakly labeled images, and the identification of the different types and subtypes of abnormalities. Similarly, we plan to extend LB-FCN *light* architecture to enable weakly-labeled localization of the staircases and other object classes within the natural images. Although LB-FCN *light* is primarily designed for outdoor mobile use, is has also been used in medical domain applications, such as in bone metastasis image classification (Ntakolia, Diamantis et al. 2020), with promising results. Towards object localization, the first steps have already been done, as LB-FCN *light* architecture has been used as the feature extractor for the YOLO-v3 (Redmon & Farhadi 2018) architecture with promising results.

In the context of artificially generated endoscopic images, the presented methodology was only a first, preliminary, approach to cope with the problem of artificial medical image generation for effective training of classifiers without real data. The results obtained from this study are promising, and can be improved. To this direction we are planning to enhance the GAN, to better represent the abnormalities. Due to the advancements in GAN architectures, we consider using different variations of GANs, such as (Karras et al. 2020). Furthermore, we plan apply this technique to a larger variety of GI tract lesions, such as polypoids and vascular conditions.

The Digital Twin simulation framework future work includes the expansion of the VR asset repository since the design of new 3D models using CAD or equivalent software is a relatively time-consuming task. Although it is possible to evaluate detection and recognition algorithms using real-world images, the simulation capabilities in the 3D space are still limited. To this end, we aim to extend the framework to automatically translate the spatial information found in objects obtained from RGB-D images in the 3D space of the virtual environment.

Finally, in the context of the ASML architecture, limitation of the architecture, one could consider its inability to automatically identify the software requirements of each task, which can result into dependency issues. To solve this problem, the tasks are required to include labels to indicate which software dependencies are required by the task. Another limitation of the architecture is that processors are considered as black boxes and only the workers are informed about what input and output can be accepted. As a result, the API has no way of knowing if an input or output data source handler is compatible with a declared processor. This can result into situations where invalid tasks are successfully created and queued to the system, failing later, when they are picked by a worker. Although this can be resolved by including documentation for each processor, we plan to include automation validation prior task execution. The architecture can utilize platforms specifically designed for parallel process execution in a multi-host environment, such as Spark (Zaharia et al. 2010); however, configuring these platforms still requires advanced technical knowledge and skills. To cope with this issue, within our future research prospects is to extend the proposed system architecture to include automated host clustering, enabling parallel task execution without special configuration. An open access implementation of the proposed system architecture is planned for distribution to the wider research community.

Deep learning continues to evolve at a rapid pace. Future research directions include:

- Methodologies that improve the generalization performance of DNNs with minimal training samples. Such methods include advanced data-augmentation techniques, such as the use of GANs to synthesize images for training data expansion.
- DNNs with minimal computational footprint, which is important for applications in low-powered embedded devices, such as the ones powering the upcoming Internet of Things (IoT). The first step towards that was the minimization of the number of free parameters of the network. While effective, such methods can degrade the overall generalization performance. Methods such as automated DNN architecture search and post-training neuron ablation based on their contribution are promising as they can enable networks to grow deeper.
- The use of fuzzy logic in deep learning can be an effective tool to increase classification performance of existing networks. An interesting direction is to use fuzzy logic in other components of CNN architectures, such as the convolution layer and input pre-processing.
- Although the photo-realistic image synthesis performance of GANs is remarkable, they are still relatively hard to train. Problems such as, convergence detection and model collapsing are still open. Towards coping with such problems, more advanced loss functions need to be investigated.

# APPENDIX

## LIST OF PUBLICATIONS IN JOURNALS

- **D. E. Diamantis** and D. K. Iakovidis, "ASML: Algorithm-Agnostic Architecture for Scalable Machine Learning," *IEEE Access*, vol. 9, pp. 51970-51982, 2021, doi: 10.1109/ACCESS.2021.3069857. (IF 3.7)

- X. Dray, D. Iakovidis, C. Houdeville, R. Jover, **D. Diamantis**, A. Histace, and A. Koulaouzidis, "Artificial intelligence in small bowel capsule endoscopy - current status, challenges and future promise.," *Journal of Gastroenterology and Hepatology*, vol. 36, no. 1, pp. 12–19, Jan. 2021. doi: 10.1111/jgh.15341 (IF 3.4)

- **D. Diamantis** and D. Iakovidis, "Fuzzy Pooling," *IEEE Transactions on Fuzzy Systems*, doi: 10.1109/TFUZZ.2020.3024023. (IF 9.5)

- G. Dimas, **D. E. Diamantis**, P. Kalozoumis, and D. K. Iakovidis, "Uncertainty-Aware Visual Perception System for Outdoor Navigation of the Visually Challenged," *Sensors*, vol. 20, no. 8, p. 2385, 2020. doi:10.3390/s20082385 (IF 3.3)

- **D. E. Diamantis**, D. K. Iakovidis, and A. Koulaouzidis, "Look-behind fully convolutional neural network for computer-aided endoscopy," *Biomedical Signal Processing and Control*, vol. 49, pp. 192–201, 2019. doi:10.1016/j.bspc.2018.12.005 (IF 3.1)

- M. Vasilakakis, **D. Diamantis**, E. Spyrou, A. Koulaouzidis, D. K. Iakovidis, "Weakly supervised multilabel classification for semantic interpretation of endoscopy video frames," *Evolving Systems* 11, 409–421 (2020). doi:10.1007/s12530-018-9236-x

## LIST OF BOOK CHAPTERS

- D. K. Iakovidis, **D. Diamantis**, G. Dimas, C. Ntakolia, and E. Spyrou, "Digital Enhancement of Cultural Experience and Accessibility for the Visually Impaired," in *Technological Trends in Improved Mobility of the Visually Impaired*, S. Paiva, Ed. Cham: Springer International Publishing, 2020, pp. 237–271. doi: 10.1007/978-3-030-16450-8_10

- **D. E. Diamantis**, P. G. Kalozoumis and D. K. Iakovidis "Digital Twin for Assistive Navigation", in *Digital Twins for Digital Transformation and Innovation in Industry: applications and future perspectives*. A.L.Hassanien, A. Darwish, Ed. (Accepted)

## LIST OF PUBLICATIONS IN PEER REVIEWED INTERNATIONAL CONFERENCES

- **D. E. Diamantis**, A. E. Zacharia, D. K. Iakovidis and A. Koulaouzidis, "Towards the Substitution of Real with Artificially Generated Endoscopic Images for CNN Training," in *Proc. IEEE International Conference on Bioinformatics and Bioengineering (BIBE)*, 2019, pp. 519-524, doi: 10.1109/BIBE.2019.00100

- **D. E. Diamantis**, D.-C. C. Koutsiou, and D. K. Iakovidis, "Staircase Detection Using a Lightweight Look-Behind Fully Convolutional Neural Network," in *Proc. Engineering Applications of Neural Networks (EANN)*, 2019, pp. 522–532, doi:10.1007/978-3-030-20257-6

- **D. Diamantis**, D. K. Iakovidis and A. Koulaouzidis, "Investigating Cross-Dataset Abnormality Detection in Endoscopy with A Weakly-Supervised Multiscale Convolutional Neural Network," in *Proc. 25th IEEE International Conference on Image Processing (ICIP)*, 2018, pp. 3124-3128, doi: 10.1109/ICIP.2018.8451673

## REFERENCES

Abadi, M. et al., 2016. Tensorflow: A system for large-scale machine learning. In *12th USENIX symposium on operating systems design and implementation OSDI 16)*. pp. 265–283.

Abnormal, E., 2015. Available at: https://endovissub-abnormal.grand-challenge.org.

Ahmad, S. & Kim, D., 2020. A multi-device multi-tasks management and orchestration architecture for the design of enterprise IoT applications. *Future Generation Computer Systems*, 106, pp.482–500.

Alkhafaji, A. et al., 2016. A survey study to gather requirements for designing a mobile service to enhance learning from cultural heritage. In *European Conference on Technology Enhanced Learning*. Springer, pp. 547–550.

Anon, 2018. Digestive Disorders & Gastrointestinal Diseases | Cleveland Clinic. Available at: https://my.clevelandclinic.org/health/articles/7040-gastrointestinal-disorders.

Anon, *USC University of Southern California - SIPI Image Dataset - Misc http://sipi.usc.edu/database/database.php?volume=misc*, USC. Available at: http://sipi.usc.edu/database/database.php?volume=misc.

Arjovsky, M. & Bottou, L., 2017. Towards principled methods for training generative adversarial networks. *arXiv preprint arXiv:1701.04862*.

Arjovsky, M., Chintala, S. & Bottou, L., 2017. Wasserstein generative adversarial networks. In *International conference on machine learning*. PMLR, pp. 214–223.

Asiain, J. & Godoy, J., 2020. Navigation in Large Groups of Robots. *Current Robotics Reports*, 1(4), pp.203–213.

Atlas, T.G., 2018. Available at: https://www.gastrointestinalatlas.com/videos/.

Bader, D.A. & Pennington, R., 2001. Applications. *The International Journal of High Performance Computing Applications*, 15(2), pp.181–185.

Badrinarayanan, V., Kendall, A. & Cipolla, R., 2017. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 39(12), pp.2481–2495.

Badrinarayanan, V., Kendall, A. & Cipolla, R., 2017. SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(12), pp.2481–2495.

Baier, L., Jöhren, F. & Seebacher, S., 2019. Challenges in the deployment and operation of machine learning in practice. In *27th European Conference on Information Systems (ECIS)*. Stockholm-Uppsala,Sweden, pp. 1–15.

Barnes, C. & Zhang, F.-L., 2017. A survey of the state-of-the-art in patch-based synthesis. *Computational Visual Media*, 3(1), pp.3–20.

Bay, H., Tuytelaars, T. & Van Gool, L., 2006. Surf: Speeded up robust features. In *European conference on computer vision*. Springer, pp. 404–417.

Bazarevsky, V. et al., 2019. Blazeface: Sub-millisecond neural face detection on mobile gpus. *arXiv preprint arXiv:1907.05047*.

Beers, A. et al., 2018. High-resolution medical image synthesis using progressively grown generative adversarial networks. *arXiv preprint arXiv:1805.03144*.

Behnke, S., 2003. *Hierarchical neural networks for image interpretation*, Springer.

Beimborn, D., Miletzki, T. & Wenzel, S., 2011. Platform as a service (PaaS). *Business & Information Systems Engineering*, 3(6), pp.381–384.

Bengio, Y. et al., 2007. Greedy layer-wise training of deep networks. *Advances in neural information processing systems*, 19, p.153.

Bennett, C.R. et al., 2018. Assessing Visual Search Performance in Ocular Compared to Cerebral Visual Impairment Using a Virtual Reality Simulation of Human Dynamic Movement. In *Proceedings of the Technology, Mind, and Society*. ACM.

Bergkvist, A. et al., 2012. Webrtc 1.0: Real-time communication between browsers. , 91. Available at: https://www.w3.org/TR/2021/REC-webrtc-20210126/.

Bergstra, J. & Bengio, Y., 2012. Random search for hyper-parameter optimization. *Journal of machine learning research*, 13(2).

Bermudez, C. et al., 2018. Learning implicit brain MRI manifolds with deep learning. In *Medical Imaging 2018: Image Processing*. International Society for Optics and Photonics, p. 105741L.

Bernal, J. et al., 2015. WM-DOVA maps for accurate polyp highlighting in colonoscopy: Validation vs. saliency maps from physicians. *Computerized Medical Imaging and Graphics*, 43, pp.99–111.

Berthelot, D., Schumm, T. & Metz, L., 2017. Began: Boundary equilibrium generative adversarial networks. *arXiv preprint arXiv:1703.10717*.

Bochinski, E., Eiselein, V. & Sikora, T., 2017. High-speed tracking-by-detection without using image information. In *14th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*. IEEE, pp. 1–6.

Bochkovskiy, A., Wang, C.-Y. & Liao, H.-Y.M., 2020. Yolov4: Optimal speed and accuracy of object detection. *arXiv preprint arXiv:2004.10934*.

Bouhamed, S.A., Kallel, I.K. & Masmoudi, D.S., 2013. Stair case detection and recognition using ultrasonic signal. In *Telecommunications and Signal Processing (TSP), 2013 36th International Conference on*. IEEE, pp. 672–676.

Boureau, Y.-L., Ponce, J. & LeCun, Y., 2010. A theoretical analysis of feature pooling in visual recognition. In *Proceedings of the 27th international conference on machine learning (ICML-10)*. pp. 111–118.

Bowles, C. et al., 2018. Gan augmentation: Augmenting training data using generative adversarial networks. *arXiv preprint arXiv:1810.10863*.

Brock, A., Donahue, J. & Simonyan, K., 2018. Large scale GAN training for high fidelity natural image synthesis. *arXiv preprint arXiv:1809.11096*.

Calimeri, F. et al., 2017. Biomedical data augmentation using generative adversarial neural networks. In *International conference on artificial neural networks*. Springer, pp. 626–634.

Chacón M, M.I., 2006. Fuzzy logic for image processing: definition and applications of a fuzzy image processing scheme. *Advanced Fuzzy Logic Technologies in Industrial Applications*, pp.101–113.

Chai, X. et al., 2017. Obstacle avoidance for a hexapod robot in unknown environment. *Science China Technological Sciences*, 60(6), pp.818–831.

Chan, S. et al., 2013. PredictionIO: a distributed machine learning server for practical software development. In *Proceedings of the 22nd ACM international conference on Information & Knowledge Management*. pp. 2493–2496.

Chang, C., 2011. A library for support vector machines.[(accessed on 10 June 2015)]. *ACM Trans. Intell. Syst. Technol*, 2, p.27.

Che, T. et al., 2016. Mode regularized generative adversarial networks. *arXiv preprint arXiv:1612.02136*.

Chen, H. et al., 2017. Automatic content understanding with cascaded spatial–temporal deep framework for capsule endoscopy videos. *Neurocomputing*, 229, pp.77–87.

Chen, L., Chen, C.P. & Lu, M., 2011. A multiple-kernel fuzzy c-means algorithm for image segmentation. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 41(5), pp.1263–1274.

Chen, Q. et al., 2019. Smartphone Based Outdoor Navigation and Obstacle Avoidance System for the Visually Impaired. In *International Conference on Multi-disciplinary Trends in Artificial Intelligence*. Springer, pp. 26–37.

Chen, T. et al., 2019. Self-supervised gans via auxiliary rotation loss. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. pp. 12154–12163.

Chen, X. et al., 2016. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. *arXiv preprint arXiv:1606.03657*.

Chollet, F., 2017. Xception: Deep learning with depthwise separable convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. pp. 1251–1258.

Chollet, F. & others, 2015. Keras.

Ciaparrone, G. et al., 2020. Deep learning in video multi-object tracking: A survey. *Neurocomputing*, 381, pp.61–88.

Ciobanu, A. et al., 2017. Real-time indoor staircase detection on mobile devices. In *Control Systems and Computer Science (CSCS), 2017 21st International Conference on*. IEEE, pp. 287–293.

Clark, A., Donahue, J. & Simonyan, K., 2019. Adversarial video generation on complex datasets. *arXiv preprint arXiv:1907.06571*.

Ben-Cohen, A. et al., 2017. Virtual PET Images from CT Data Using Deep Convolutional Networks: Initial Results. In S. A. Tsaftaris et al., eds. *SASHIMI@MICCAI*. Lecture Notes in Computer Science. Springer, pp. 49–57.

Cong, Y. et al., 2015. Deep sparse feature selection for computer aided endoscopy diagnosis. *Pattern Recognition*, 48(3), pp.907–917.

Cong, Y. et al., 2016. UDSFS: Unsupervised deep sparse feature selection. *Neurocomputing*, 196, pp.150–158.

Costa, P. et al., 2018. End-to-End Adversarial Retinal Image Synthesis. *IEEE Trans. Med. Imaging*, 37(3), pp.781–791.

Cui, L., 2018. MDSSD: Multi-scale Deconvolutional Single Shot Detector for small objects. *arXiv preprint arXiv:1805.07009*.

Curtis, S., Best, A. & Manocha, D., 2016. Menge: A modular framework for simulating crowd movement. *Collective Dynamics*, 1, pp.1–40.

Cybenko, G., 1989. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4), pp.303–314.

Dai, J. et al., 2016. R-fcn: Object detection via region-based fully convolutional networks. In *Advances in neural information processing systems*. pp. 379–387.

Dai, W. et al., 2017. SCAN: Structure Correcting Adversarial Network for Chest X-rays Organ Segmentation. *CoRR*, abs/1703.08770. Available at: http://arxiv.org/abs/1703.08770.

Dai, X. et al., 2019. Chamnet: Towards efficient network design through platform-aware model adaptation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. pp. 11398–11407.

Dakopoulos, D. & Bourbakis, N.G., 2009. Wearable obstacle avoidance electronic travel aids for blind: a survey. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 40(1), pp.25–35.

Daras, G. et al., 2020. Your local GAN: Designing two dimensional local attention mechanisms for generative models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. pp. 14531–14539.

Darken, C. et al., 1992. Learning rate schedules for faster stochastic gradient search. In *Neural networks for signal processing*. Citeseer.

Dean, J. et al., 2012. Large scale distributed deep networks.

Deng, J. et al., 2009. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*. Ieee, pp. 248–255.

Deng, L., 2012. The mnist database of handwritten digit images for machine learning research [best of the web]. *IEEE Signal Processing Magazine*, 29(6), pp.141–142.

Denton, E. et al., 2015. Deep generative image models using a laplacian pyramid of adversarial networks. *arXiv preprint arXiv:1506.05751*.

Van Der Malsburg, C., 1986. Frank Rosenblatt: Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms. In G. Palm & A. Aertsen, eds. *Brain Theory*. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 245–248.

Diamantis, D., Iakovidis, D.K. & Koulaouzidis, A., 2018. Investigating Cross-Dataset Abnormality Detection in Endoscopy with A Weakly-Supervised Multiscale Convolutional Neural Network. In *2018 25th IEEE International Conference on Image Processing (ICIP)*. pp. 3124–3128.

Diamantis, D., Iakovidis, D.K. & Koulaouzidis, A., 2018. Investigating cross-dataset abnormality detection in endoscopy with a weakly-supervised multiscale convolutional neural network. In *2018 25th IEEE International Conference on Image Processing (ICIP)*. IEEE, pp. 3124–3128.

Diamantis, D.E., Zacharia, A.E., Iakovidis, D.K., et al., 2019. Towards the Substitution of Real with Artificially Generated Endoscopic Images for CNN Training. In *2019 IEEE 19th International Conference on Bioinformatics and Bioengineering (BIBE)*. pp. 519–524.

Diamantis, D.E. & Iakovidis, D.K., 2021. ASML: Algorithm-Agnostic Architecture for Scalable Machine Learning. *IEEE Access*, 9, pp.51970–51982.

Diamantis, D.E. & Iakovidis, D.K., 2020. Fuzzy Pooling. *IEEE Transactions on Fuzzy Systems*.

Diamantis, D.E., Iakovidis, D.K. & Koulaouzidis, A., 2019. Look-behind fully convolutional neural network for computer-aided endoscopy. *Biomedical Signal Processing and Control*, 49, pp.192–201. Available at: https://www.sciencedirect.com/science/article/pii/S1746809418303033.

Diamantis, D.E., Iakovidis, D.K. & Koulaouzidis, A., 2019. Look-behind fully convolutional neural network for computer-aided endoscopy. *Biomedical Signal Processing and Control*, 49, pp.192–201.

Diamantis, D.E., Koutsiou, D.-C.C. & Iakovidis, D.K., 2019. Staircase Detection Using a Lightweight Look-Behind Fully Convolutional Neural Network. In J. Macintyre et al., eds. *Engineering Applications of Neural Networks*. Cham: Springer International Publishing, pp. 522–532.

Dimas, G. et al., 2020. Uncertainty-Aware Visual Perception System for Outdoor Navigation of the Visually Challenged. *Sensors*, 20(8), p.2385.

Donahue, J., Krähenbühl, P. & Darrell, T., 2016. Adversarial feature learning. *arXiv preprint arXiv:1605.09782*.

Dossot, D., 2014. *RabbitMQ essentials*, Packt Publishing Ltd.

Dozat, T., 2016. Incorporating nesterov momentum into adam.

Drake, J. & Hamerly, G., 2012. Accelerated k-means with adaptive distance bounds. In *5th NIPS workshop on optimization for machine learning*.

Dray, X. et al., 2021. Artificial intelligence in small bowel capsule endoscopy - current status, challenges and future promise. *Journal of gastroenterology and hepatology*, 36(1), pp.12–19.

Du, K.-L. & Swamy, M.N., 2006. *Neural networks in a softcomputing framework*, Springer Science & Business Media.

Duchi, J., Hazan, E. & Singer, Y., 2011. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(7).

Efros, A.A. & Leung, T.K., 1999. Texture synthesis by non-parametric sampling. In *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*. pp. 1033–1038.

Erhan, D. et al., 2014. Scalable Object Detection using Deep Neural Networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Everingham, M. et al., 2010. The pascal visual object classes (voc) challenge. *International journal of computer vision*, 88(2), pp.303–338.

Fabelo, H. et al., 2016. HELICoiD project: a new use of hyperspectral imaging for brain cancer detection in real-time during neurosurgical operations. In D. P. Bannon, ed. *Hyperspectral Imaging Sensors: Innovative Applications and Sensor Standards 2016*. SPIE, pp. 1–12. Available at: https://doi.org/10.1117/12.2223075.

Fawcett, T., 2006. An introduction to ROC analysis. *Pattern recognition letters*, 27(8), pp.861–874.

Fernández-Villamor, J.I., Iglesias, C.A. & Garijo, M., 2013. A framework for goal-oriented discovery of resources in the RESTful architecture. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 44(6), pp.796–803.

Fette, I. & Melnikov, A., 2016. The websocket protocol (2011). Available at: https://tools.ietf.org/html/rfc6455.

FFmpeg, 2016. ffmpeg tool (Version be1d324). Available at: https://ffmpeg. org.

Fiaz, M. et al., 2019. Handcrafted and deep trackers: Recent visual object tracking approaches and trends. *ACM Computing Surveys (CSUR)*, 52(2), pp.1–44.

Flickr Inc., 2019. Find your inspiration. | Flickr.

Foundation, T.C.C., 2017. Digestive Disorders & Gastrointestinal Diseases | Cleveland Clinic. Available at: https://my.clevelandclinic.org/health/articles/gastrointestinal-disorder.

Fraichard, T. & Levesy, V., 2020. From Crowd Simulation to Robot Navigation in Crowds. *IEEE Robotics and Automation Letters*, 5(2), pp.729–735.

Frid-Adar, M. et al., 2018. GAN-based synthetic medical image augmentation for increased CNN performance in liver lesion classification. *Neurocomputing*, 321, pp.321–331.

Fu, C.-Y. et al., 2017. DSSD: Deconvolutional single shot detector. *arXiv preprint arXiv:1701.06659*.

Fu, J. & Xu, C., 2019. A survey of single object tracking methods. *Nanjing Xinxi Gongcheng Daxue Xuebao*, 11(6), pp.638–650.

Fukushima, K. & Miyake, S., 1982. Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition. In *Competition and cooperation in neural nets*. Springer, pp. 267–285.

Gao, S.-H. et al., 2021. Res2Net: A New Multi-Scale Backbone Architecture. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43(2), pp.652–662.

Garber, L., 2000. Denial-of-service attacks rip the Internet. *Computer*, (4), pp.12–17.

García, Á.L. et al., 2020. A Cloud-Based Framework for Machine Learning Workloads and Applications. *IEEE Access*, 8, pp.18681–18692.

Garner, S.R. & others, 1995. Weka: The waikato environment for knowledge analysis. In *Proceedings of the New Zealand computer science research students conference*. pp. 57–64.

GASTROLAB, 2018. High Resolution Videos and Images: Colon, the Large Bowel. Available at: http://www.gastrolab.fi/videos/.

Gatys, L.A., Ecker, A.S. & Bethge, M., 2015. Texture Synthesis Using Convolutional Neural Networks. In C. Cortes et al., eds. *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*. pp. 262–270. Available at: http://papers.nips.cc/book/advances-in-neural-information-processing-systems-28-2015.

Gholami, A. et al., 2018. Squeezenext: Hardware-aware neural network design. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. pp. 1638–1647.

Girosi, F., Jones, M. & Poggio, T., 1995. Regularization theory and neural networks architectures. *Neural computation*, 7(2), pp.219–269.

Girshick, R., 2015. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*. pp. 1440–1448.

Girshick, R. et al., 2014. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. pp. 580–587.

Gong, X. et al., 2019. Autogan: Neural architecture search for generative adversarial networks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. pp. 3224–3234.

Gono, K. et al., 2004. Appearance of enhanced tissue features in narrow-band endoscopic imaging. *Journal of biomedical optics*, 9(3), pp.568–577.

Gonog, L. & Zhou, Y., 2019. A review: Generative adversarial networks. In *2019 14th IEEE Conference on Industrial Electronics and Applications (ICIEA)*. IEEE, pp. 505–510.

Gonzalez, R.C. & Woods, R.E., 2018. *Digital image processing* 3rd ed., Pearson.

Goodfellow, I., Bengio, Y. & Courville, A., 2016. *Deep Learning*, MIT Press.

Goodfellow, I.J. et al., 2014. Generative adversarial networks. *arXiv preprint arXiv:1406.2661*.

Gopal, M., 2019. *Applied machine learning*, McGraw-Hill Education.

Graham, B., 2014. Fractional max-pooling. *arXiv preprint arXiv:1412.6071*.

Gu, J. et al., 2018. Recent advances in convolutional neural networks. *Pattern Recognition*, 77, pp.354–377.

Gui, J. et al., 2020. A review on generative adversarial networks: Algorithms, theory, and applications. *arXiv preprint arXiv:2001.06937*.

Guibas, J.T., Virdi, T.S. & Li, P.S., 2017. Synthetic Medical Images from Dual Generative Adversarial Networks. *CoRR*, abs/1709.01872. Available at: http://arxiv.org/abs/1709.01872.

Guldenring, R. et al., 2020. Learning Local Planners for Human-aware Navigation in Indoor Environments. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, pp. 6053–6060.

Gulli, A. & Pal, S., 2017. *Deep learning with Keras*, Packt Publishing Ltd.

Gulrajani, I. et al., 2017. Improved training of wasserstein gans. *arXiv preprint arXiv:1704.00028*.

Gupta, J.N. & Sexton, R.S., 1999. Comparing backpropagation with a genetic algorithm for neural network training. *Omega*, 27(6), pp.679–684.

Habiba, M., Islam, M.R. & Ali, A.S., 2018. A component based unified architecture for utility service in cloud. *Future Generation Computer Systems*, 87, pp.725–742.

Häfner, M. et al., 2015. Local fractal dimension based approaches for colonic polyp classification. *Medical image analysis*, 26(1), pp.92–107.

Han, C. et al., 2018. GAN-based synthetic brain MR image generation. In *ISBI*. IEEE, pp. 734–738. Available at: http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=8359997.

Han, K. et al., 2020. Ghostnet: More features from cheap operations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. pp. 1580–1589.

Han, Y.-J., Kim, I.-S. & Hong, Y.-D., 2018. Optimization-based humanoid robot navigation using monocular camera within indoor environment. *ETRI Journal*, 40(4), pp.446–457.

Hardt, D. & others, 2012. *The OAuth 2.0 authorization framework*, RFC 6749, October.

He, J.-Y. et al., 2018. Hookworm detection in wireless capsule endoscopy images with deep learning. *IEEE Transactions on Image Processing*, 27(5), pp.2379–2392.

He, K. et al., 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. pp. 770–778.

He, K. et al., 2015a. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*. pp. 1026–1034.

He, K. et al., 2015b. Spatial pyramid pooling in deep convolutional networks for visual recognition. *IEEE transactions on pattern analysis and machine intelligence*, 37(9), pp.1904–1916.

He, K. & Sun, J., 2015. Convolutional neural networks at constrained time cost. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. pp. 5353–5360.

Hecht-Nielsen, R., 1992. Theory of the backpropagation neural network. In *Neural networks for perception*. Elsevier, pp. 65–93.

Hegenbart, S. et al., 2013. Scale invariant texture descriptors for classifying celiac disease. *Medical image analysis*, 17(4), pp.458–474.

Hesch, J.A., Mariottini, G.L. & Roumeliotis, S.I., 2010. Descending-stair detection, approach, and traversal with an autonomous tracked vehicle. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*. IEEE, pp. 5525–5531.

Hinton, G., Srivastava, N. & Swersky, K., 2012. Neural networks for machine learning lecture 6a overview of mini-batch gradient descent. *Cited on*, 14(8).

Hinton, G.E., 2012. A practical guide to training restricted Boltzmann machines. In *Neural networks: Tricks of the trade*. Springer, pp. 599–619.

Hochreiter, S., 1998. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6(02), pp.107–116.

Hornik, K., Stinchcombe, M. & White, H., 1989. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5), pp.359–366.

Howard, A. et al., 2019. Searching for mobilenetv3. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. pp. 1314–1324.

Howard, A.G. et al., 2017. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*.

Huang, G. et al., 2018. Condensenet: An efficient densenet using learned group convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. pp. 2752–2761.

Huang, G. et al., 2017. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. pp. 4700–4708.

Hubel, D.H. & Wiesel, T.N., 1968. Receptive fields and functional architecture of monkey striate cortex. *The Journal of physiology*, 195(1), pp.215–243.

Hungerford, K., Dasgupta, P. & Guruprasad, K.R., 2016. A Repartitioning Algorithm to Guarantee Complete, Non-overlapping Planar Coverage with Multiple Robots. In *Springer Tracts in Advanced Robotics*. Springer Japan, pp. 33–48.

Iacono, M. & Sgorbissa, A., 2018. Path following and obstacle avoidance for an autonomous UAV using a depth camera. *Robotics and Autonomous Systems*, 106, pp.38–46.

Iakovidis, D.K. et al., 2015. Blood detection in wireless capsule endoscope images based on salient superpixels. In *2015 37th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*. IEEE, pp. 731–734.

Iakovidis, D.K. et al., 2018. Detecting and Locating Gastrointestinal Anomalies Using Deep Learning and Iterative Cluster Unification. *IEEE Transactions on Medical Imaging*, 37(10), pp.2196–2210.

Iakovidis, D.K. et al., 2020. Digital Enhancement of Cultural Experience and Accessibility for the Visually Impaired. In S. Paiva, ed. *Technological Trends in Improved Mobility of the Visually Impaired*. Cham: Springer International Publishing, pp. 237–271. Available at: https://doi.org/10.1007/978-3-030-16450-8_10.

Iakovidis, D.K. & Diamantis, D.E., 2014. Open-Access Framework for Efficient Object-Oriented Development of Video Analysis Software. *Journal of Software Engineering and Applications*, Vol.07No.08. Available at: //www.scirp.org/journal/paperinformation.aspx?paperid=48124.

Iakovidis, D.K. & Koulaouzidis, A., 2015. Software for enhanced video capsule endoscopy: challenges for essential progress. *Nature Reviews Gastroenterology & Hepatology*, 12(3), pp.172–186.

Iandola, F.N. et al., 2016. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and< 0.5 MB model size. *arXiv preprint arXiv:1602.07360*.

Inamura, T. & Mizuchi, Y., 2017. Competition design to evaluate cognitive functions in human-robot interaction based on immersive VR. In *Robot World Cup*. Springer, pp. 84–94.

Inamura, T. & Mizuchi, Y., 2020. SIGVerse: A cloud-based VR platform for research on social and embodied human-robot interaction. *arXiv preprint arXiv:2005.00825*.

Instruments, N., 2013. Peak signal-to-noise ratio as an image quality metric.

Ioffe, S. & Szegedy, C., 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*. PMLR, pp. 448–456.

Isola, P. et al., 2017. Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. pp. 1125–1134.

Janocha, K. & Czarnecki, W.M., 2017. On loss functions for deep neural networks in classification. *arXiv preprint arXiv:1702.05659*.

Jetchev, N., Bergmann, U. & Vollgraf, R., 2016. Texture Synthesis with Spatial Generative Adversarial Networks. *CoRR*, abs/1611.08207. Available at: http://arxiv.org/abs/1611.08207.

Jia, X. & Meng, M.Q.-H., 2016. A deep convolutional neural network for bleeding detection in wireless capsule endoscopy images. In *2016 38th annual international conference of the IEEE engineering in medicine and biology society (EMBC)*. IEEE, pp. 639–642.

Jia, X. & Meng, M.Q.-H., 2017. A study on automated segmentation of blood regions in wireless capsule endoscopy images using fully convolutional networks. In *2017 IEEE 14th International Symposium on Biomedical Imaging (ISBI 2017)*. IEEE, pp. 179–182.

Jia, Y. et al., 2014. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22nd ACM international conference on Multimedia*. pp. 675–678.

Jiao, L. & Zhao, J., 2019. A Survey on the New Generation of Deep Learning in Image Processing. *IEEE Access*, 7, pp.172231–172263.

Jin, J., Dundar, A. & Culurciello, E., 2014. Flattened convolutional neural networks for feedforward acceleration. *arXiv preprint arXiv:1412.5474*.

Johnson, J., Alahi, A. & Li, F.-F., 2016. Perceptual Losses for Real-Time Style Transfer and Super-Resolution. *CoRR*, abs/1603.08155. Available at: http://arxiv.org/abs/1603.08155.

Jolicoeur-Martineau, A., 2018. The relativistic discriminator: a key element missing from standard GAN. *arXiv preprint arXiv:1807.00734*.

Jones, A.J., 1993. Genetic algorithms and their applications to the design of neural networks. *Neural Computing and Applications*, 1(1), pp.32–45.

Jones, M. & Hardt, D., 2012. *The oauth 2.0 authorization framework: Bearer token usage*, Available at: https://tools.ietf.org/html/rfc6750.

Jouppi, N.P. et al., 2017. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th Annual International Symposium on Computer Architecture*. pp. 1–12.

Juang, C.-F. & Hsieh, C.-D., 2010. A locally recurrent fuzzy neural network with support vector regression for dynamic-system modeling. *IEEE Transactions on Fuzzy Systems*, 18(2), pp.261–273.

Juang, C.-F. & Ku, K.-C., 2005. A recurrent fuzzy network for fuzzy temporal sequence processing and gesture recognition. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 35(4), pp.646–658.

Karkanis, S.A. et al., 2003. Computer-aided tumor detection in endoscopic video using color wavelet features. *IEEE transactions on information technology in biomedicine*, 7(3), pp.141–152.

Karras, T. et al., 2020. Analyzing and improving the image quality of stylegan. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. pp. 8110–8119.

Karras, T. et al., 2017. Progressive growing of gans for improved quality, stability, and variation. *arXiv preprint arXiv:1710.10196*.

Karras, T., Laine, S. & Aila, T., 2019. A style-based generator architecture for generative adversarial networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. pp. 4401–4410.

Katz, B.F.G. et al., 2012. NAVIG: augmented reality guidance system for the visually impaired. *Virtual Reality*, 16(4), pp.253–269.

Kaur, S., Aggarwal, H. & Rani, R., 2021. MR Image Synthesis Using Generative Adversarial Networks for Parkinson's Disease Classification. In *Proceedings of International Conference on Artificial Intelligence and Applications*. Springer, pp. 317–327.

Kazeminia, S. et al., 2020. GANs for medical image analysis. *Artificial Intelligence in Medicine*, p.101938.

Kebria, P.M. et al., 2019. Adaptive Type-2 Fuzzy Neural-Network Control for Teleoperation Systems with Delay and Uncertainties. *IEEE Transactions on Fuzzy Systems*.

Kelley, C.T., 1999. *Iterative methods for optimization*, SIAM.

Keselman, L. et al., 2017. Intel realsense stereoscopic depth cameras. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. pp. 1–10.

Van Kesteren, A. & others, 2010. Cross-origin resource sharing. Available at: https://www.w3.org/TR/2010/WD-cors-20100727/.

Kiefer, J., Wolfowitz, J. & others, 1952. Stochastic estimation of the maximum of a regression function. *The Annals of Mathematical Statistics*, 23(3), pp.462–466.

Kiesslich, R. et al., 2005. Confocal laser endomicroscopy. *Gastrointestinal endoscopy clinics of North America*, 15(4), pp.715–731.

Kim, S. et al., 2016. Interactive and adaptive data-driven crowd simulation. In *2016 IEEE Virtual Reality (VR)*. IEEE, pp. 29–38.

Kim, T. et al., 2017. Learning to discover cross-domain relations with generative adversarial networks. In *International Conference on Machine Learning*. PMLR, pp. 1857–1865.

Kingma, D.P. & Ba, J., 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kingma, D.P. & Welling, M., 2019. An introduction to variational autoencoders. *arXiv preprint arXiv:1906.02691*.

Kingma, D.P. & Welling, M., 2013. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*.

Koulaouzidis, A. et al., 2017. KID Project: an internet-based digital video atlas of capsule endoscopy for research purposes. *Endosc Int Open*, 5(6), pp.E477–E483.

Koulaouzidis, A., Rondonotti, E. & Karargyris, A., 2013. Small-bowel capsule endoscopy: a ten-point contemporary review. *World journal of gastroenterology: WJG*, 19(24), p.3726.

Kraska, T. et al., 2013. MLbase: A Distributed Machine-learning System. In *Proc. 6th Biennial Conference on Innovative Data Systems Research (CIDR)*. Available at: http://cidrdb.org/cidr2013/Papers/CIDR13_Paper118.pdf.

Kreimeier, J. & Götzelmann, T., 2019. First Steps Towards Walk-In-Place Locomotion and Haptic Feedback in Virtual Reality for Visually Impaired. In *Extended Abstracts of the 2019 CHI Conference on Human Factors in Computing Systems*. ACM.

Krizhevsky, A., Hinton, G. & others, 2009. *Learning multiple layers of features from tiny images*, Citeseer.

Krizhevsky, A., Sutskever, I. & Hinton, G.E., 2012. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, pp.1097–1105.

Kubat, M., 1999. Neural networks: a comprehensive foundation by Simon Haykin, Macmillan, 1994, ISBN 0-02-352781-7. *The Knowledge Engineering Review*, 13(4), pp.409–412.

Kumar, M. et al., 2015. Fuzzy membership descriptors for images. *IEEE Transactions on Fuzzy Systems*, 24(1), pp.195–207.

Kumar, N., Takács, M. & Vámossy, Z., 2017. Robot navigation in unknown environment using fuzzy logic. In *2017 IEEE 15th International Symposium on Applied Machine Intelligence and Informatics (SAMI)*. IEEE, pp. 279–284.

Kunz, A. et al., 2018. Virtual Navigation Environment for Blind and Low Vision People. In *Lecture Notes in Computer Science*. Springer International Publishing, pp. 114–122.

LeCun, Y. et al., 1989. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4), pp.541–551.

LeCun, Y. et al., 1998. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11), pp.2278–2324.

LeCun, Y., Bengio, Y. & Hinton, G.E., 2015. Deep learning. *Nature*, 521(7553), pp.436–444.

Ledig, C. et al., 2017. Photo-realistic single image super-resolution using a generative adversarial network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. pp. 4681–4690.

Lee, C.-S., Guo, S.-M. & Hsu, C.-Y., 2005. Genetic-based fuzzy image filter and its application to image processing. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 35(4), pp.694–711.

Lee, Y.H., Leung, T.-S. & Medioni, G., 2012. Real-time staircase detection from a wearable stereo system. In *Pattern Recognition (ICPR), 2012 21st International Conference On*. IEEE, pp. 3770–3773.

Li, Y. et al., 2017. Generative face completion. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. pp. 3911–3919.

Li, Y. et al., 2019. Navigation Simulation of a Mecanum Wheel Mobile Robot Based on an Improved Aast Algorithm in Unity3D. *Sensors*, 19(13), p.2976.

Li, Y. & Manoharan, S., 2013. A performance comparison of SQL and NoSQL databases. In *2013 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM)*. IEEE, pp. 15–19.

Liedlgruber, M. & Uhl, A., 2011. Computer-aided decision support systems for endoscopy in the gastrointestinal tract: a review. *IEEE reviews in biomedical engineering*, 4, pp.73–88.

Lim, J.H. & Ye, J.C., 2017. Geometric gan. *arXiv preprint arXiv:1705.02894*.

Lin, S. et al., 2018. KrNet: A kinetic real-time convolutional neural network for navigational assistance. In *International Conference on Computers Helping People with Special Needs*. Springer, pp. 55–62.

Lin, T.-Y. et al., 2017. Feature Pyramid Networks for Object Detection. In *CVPR*. p. 4.

Lin, T.-Y. et al., 2018. Focal loss for dense object detection. *IEEE transactions on pattern analysis and machine intelligence*.

Liu, D.-Y. et al., 2016. Identification of lesion images from gastrointestinal endoscope based on feature extraction of combinational methods with and without learning process. *Medical image analysis*, 32, pp.281–294.

Liu, W. et al., 2017. A survey of deep neural network architectures and their applications. *Neurocomputing*, 234, pp.11–26.

Liu, W. et al., 2016. Ssd: Single shot multibox detector. In *European conference on computer vision*. Springer, pp. 21–37.

Lowe, D.G., 2004. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2), pp.91–110.

Lutz, S., Amplianitis, K. & Smolic, A., 2018. Alphagan: Generative adversarial networks for natural image matting. *arXiv preprint arXiv:1807.10088*.

Ma, N. et al., 2018. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In *Proceedings of the European conference on computer vision (ECCV)*. pp. 116–131.

Maas, A.L., Hannun, A.Y. & Ng, A.Y., 2013. Rectifier nonlinearities improve neural network acoustic models. In *in ICML Workshop on Deep Learning for Audio, Speech and Language Processing*.

Mahy, R., Matthews, P. & Rosenberg, J., 2010. *Traversal using relays around nat (turn): Relay extensions to session traversal utilities for nat (stun)*, Available at: https://tools.ietf.org/html/rfc5766.

Makhzani, A. et al., 2015. Adversarial Autoencoders. *CoRR*, abs/1511.05644. Available at: http://arxiv.org/abs/1511.05644.

Malinowski, M. & Fritz, M., 2013. Learnable pooling regions for image classification. *arXiv preprint arXiv:1301.3516*.

Mamonov, A.V. et al., 2014. Automated polyp detection in colon capsule endoscopy. *IEEE transactions on medical imaging*, 33(7), pp.1488–1502.

Manjari, K., Verma, M. & Singal, G., 2020. A survey on assistive technology for visually impaired. *Internet of Things*, 11, p.100188.

Mao, X. et al., 2017. Least squares generative adversarial networks. In *Proceedings of the IEEE international conference on computer vision*. pp. 2794–2802.

Maohai, L. et al., 2014. A robust vision-based method for staircase detection and localization. *Cognitive processing*, 15(2), pp.173–194.

Martınez-Prieto, M.A. et al., 2015. The solid architecture for real-time management of big semantic data. *Future Generation Computer Systems*, 47, pp.62–79.

Marzullo, A. et al., 2021. Towards realistic laparoscopic image generation using image-domain translation. *Computer Methods and Programs in Biomedicine*, 200, p.105834.

Maury, B. & Faure, S., 2018. *Crowds in Equations: An Introduction to the Microscopic Modeling of Crowds*, World Scientific.

McCulloch, W.S. & Pitts, W., 1943. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4), pp.115–133.

McIntosh, J., Marques, B. & Harkness, R., 2020. Simulating impairment through virtual reality. *International Journal of Architectural Computing*, 18(3), pp.284–295.

Mehta, S. et al., 2018. Espnet: Efficient spatial pyramid of dilated convolutions for semantic segmentation. In *Proceedings of the european conference on computer vision (ECCV)*. pp. 552–568.

Mehta, S. et al., 2019. Espnetv2: A light-weight, power efficient, and general purpose convolutional neural network. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. pp. 9190–9200.

Mehta, S., Hajishirzi, H. & Rastegari, M., 2020. DiCENet: Dimension-wise convolutions for efficient networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Merkel, D., 2014. Docker: lightweight linux containers for consistent development and deployment. *Linux journal*, 2014(239), p.2.

Metz, L. et al., 2016. Unrolled generative adversarial networks. *arXiv preprint arXiv:1611.02163*.

Minsky, M.L. & Papert, S.A., 1988. Perceptrons: expanded edition.

Mirza, M. & Osindero, S., 2014. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*.

Miyato, T. et al., 2018. Spectral normalization for generative adversarial networks. *arXiv preprint arXiv:1802.05957*.

Mohanan, M. & Salgoankar, A., 2018. A survey of robotic motion planning in dynamic environments. *Robotics and Autonomous Systems*, 100, pp.171–185.

Moldoveanu, A.D.B. et al., 2017. Mastering an advanced sensory substitution device for visually impaired through innovative virtual training. In *2017 IEEE 7th International Conference on Consumer Electronics - Berlin (ICCE-Berlin)*. IEEE.

Montana, D.J., 1995. Neural network weight selection using genetic algorithms. *Intelligent Hybrid Systems*, 8(6), pp.12–19.

Moreira, M. & Fiesler, E., 1995. *Neural networks with adaptive learning rate and momentum terms*, Idiap.

Muller, A.D. & Sonnenberg, A., 1995. Prevention of colorectal cancer by flexible endoscopy and polypectomy: a case-control study of 32 702 veterans. *Annals of internal medicine*, 123(12), pp.904–910.

Murata, N., Yoshizawa, S. & Amari, S., 1994. Network information criterion-determining the number of hidden units for an artificial neural network model. *IEEE transactions on neural networks*, 5(6), pp.865–872.

Nair, V. & Hinton, G.E., 2010. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*. pp. 807–814.

Navab, N. et al., 2015. *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III*, Springer.

Nesterov, Y., 1983. A method of solving a convex programming problem with convergence rate O (1/k 2) O (1/k2). In *Sov. Math. Dokl*.

Neyshabur, B. et al., 2017. Exploring generalization in deep learning. In *Advances in Neural Information Processing Systems*. pp. 5947–5956.

Ng, J. & Bräunl, T., 2007. Performance comparison of bug navigation algorithms. *Journal of Intelligent and Robotic Systems*, 50(1), pp.73–84.

Nguyen, G. et al., 2019. Machine learning and deep learning frameworks and libraries for large-scale data mining: a survey. *Artificial Intelligence Review*, 52(1), pp.77–124.

Nickolls, J. et al., 2008. Scalable parallel programming with cuda: Is cuda the parallel programming model that application developers have been waiting for? *Queue*, 6(2), pp.40–53.

Nie, D. et al., 2016. Medical Image Synthesis with Context-Aware Generative Adversarial Networks. *CoRR*, abs/1612.05362. Available at: http://arxiv.org/abs/1612.05362.

Nie, D. et al., 2018. Medical Image Synthesis with Deep Convolutional Adversarial Networks. *IEEE Trans. Biomed. Engineering*, 65(12), pp.2720–2730.

Nowozin, S., Cseke, B. & Tomioka, R., 2016. f-gan: Training generative neural samplers using variational divergence minimization. *arXiv preprint arXiv:1606.00709*.

Ntakolia, C., Diamantis, D.E., et al., 2020. A Lightweight Convolutional Neural Network Architecture Applied for Bone Metastasis Classification in Nuclear Medicine: A Case Study on Prostate Cancer Patients. *Healthcare*, 8(4). Available at: https://www.mdpi.com/2227-9032/8/4/493.

Ntakolia, C., Dimas, G. & Iakovidis, D.K., 2020. User-centered system design for assisted navigation of visually impaired individuals in outdoor cultural environments. *Universal Access in the Information Society*, pp.1–26.

Odena, A., 2016. Semi-supervised learning with generative adversarial networks. *arXiv preprint arXiv:1606.01583*.

Odena, A., Olah, C. & Shlens, J., 2017. Conditional image synthesis with auxiliary classifier gans. In *International conference on machine learning*. PMLR, pp. 2642–2651.

Olston, C. et al., 2008. Pig latin: a not-so-foreign language for data processing. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*. pp. 1099–1110.

Pan, J. et al., 2017. Salgan: Visual saliency prediction with generative adversarial networks. *arXiv preprint arXiv:1701.01081*.

Pan, Z. et al., 2020. Loss Functions of Generative Adversarial Networks (GANs): Opportunities and Challenges. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 4(4), pp.500–522.

Pang, B. et al., 2019. A Swarm Robotic Exploration Strategy Based on an Improved Random Walk Method. *Journal of Robotics*, 2019, pp.1–9.

Pang, C., Yan, J. & Vyatkin, V., 2014. Time-complemented event-driven architecture for distributed automation systems. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 45(8), pp.1165–1177.

Park, S.W. & Kwon, J., 2019. Sphere generative adversarial network based on geometric moment matching. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. pp. 4292–4301.

Parmar, H. & Thornburgh, M., 2012. Adobe’s real time messaging protocol. *Copyright Adobe Systems Incorporated*, pp.1–52.

Paszke, A. et al., 2019. Pytorch: An imperative style, high-performance deep learning library. *arXiv preprint arXiv:1912.01703*.

Peli, E., 1990. Contrast in complex images. *Journal of the Optical Society of America A*, 7(10), pp.2032–2040.

Pennington, J., Socher, R. & Manning, C.D., 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. pp. 1532–1543.

Pérez-Yus, A., López-Nicolás, G. & Guerrero, J.J., 2014. Detection and modelling of staircases using a wearable depth sensor. In *European Conference on Computer Vision*. Springer, pp. 449–463.

Petzka, H., Fischer, A. & Lukovnicov, D., 2017. On the regularization of wasserstein gans. *arXiv preprint arXiv:1709.08894*.

Poggi, M. & Mattoccia, S., 2016. A wearable mobility aid for the visually impaired based on embedded 3D vision and deep learning. In *2016 IEEE Symposium on Computers and Communication (ISCC)*. pp. 208–213.

Portilla, J. & Simoncelli, E.P., 2000. A Parametric Texture Model Based on Joint Statistics of Complex Wavelet Coefficients. *International Journal of Computer Vision*, 40(1), pp.49–70. Available at: http://dx.doi.org/10.1023/A:1026553619983.

Pratama, P.S. et al., 2016. Positioning and obstacle avoidance of automatic guided vehicle in partially known environment. *International Journal of Control, Automation and Systems*, 14(6), pp.1572–1581.

Qian, N., 1999. On the momentum term in gradient descent learning algorithms. *Neural networks*, 12(1), pp.145–151.

Radford, A., Metz, L. & Chintala, S., 2015. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*.

Ravi, D. et al., 2016. Deep learning for health informatics. *IEEE journal of biomedical and health informatics*, 21(1), pp.4–21.

Read, J. et al., 2016. Meka: a multi-label/multi-target extension to weka.

Redmon, J. et al., 2016. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. pp. 779–788.

Redmon, J. & Farhadi, A., 2017. YOLO9000: better, faster, stronger. *arXiv preprint*.

Redmon, J. & Farhadi, A., 2018. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*.

Reed, S. et al., 2016. Generative adversarial text to image synthesis. In *International Conference on Machine Learning*. PMLR, pp. 1060–1069.

Reese, W., 2008. Nginx: the high-performance web server and reverse proxy. *Linux Journal*, 2008(173), p.2.

Ren, S. et al., 2015. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*. pp. 91–99.

Ribeiro, E., Uhl, A. & Häfner, M., 2016. Colonic polyp classification with convolutional neural networks. In *2016 IEEE 29th International Symposium on Computer-Based Medical Systems (CBMS)*. IEEE, pp. 253–258.

Ribeiro, M., Grolinger, K. & Capretz, M.A., 2015. Mlaas: Machine learning as a service. In *2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA)*. IEEE, pp. 896–902.

Richardson, L. & Ruby, S., 2008. *RESTful Web Services*,  O''Reilly Media, Inc. Available at: https://www.oreilly.com/library/view/restful-web-services/9780596529260/.

Robbins, H. & Monro, S., 1951. A stochastic approximation method. *The annals of mathematical statistics*, pp.400–407.

Rodeh, O. & Teperman, A., 2003. zFS - a scalable distributed file system using object disks. In *20th IEEE/11th NASA Goddard Conference on Mass Storage Systems and Technologies, 2003. (MSST 2003). Proceedings*. pp. 207–218.

Rosasco, L. et al., 2004. Are loss functions all the same? *Neural computation*, 16(5), pp.1063–1076.

Rosen, R., 2014. Linux containers and the future cloud. *Linux J*, 240(4), pp.86–95.

Rosique, F. et al., 2019. A Systematic Review of Perception System and Simulators for Autonomous Vehicles Research. *Sensors*, 19(3), p.648.

Roth, K. et al., 2017. Stabilizing training of generative adversarial networks through regularization. *arXiv preprint arXiv:1705.09367*.

Ruder, S., 2016. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*.

Rumelhart, D.E., Hinton, G.E. & Williams, R.J., 1986. Learning representations by back-propagating errors. *nature*, 323(6088), pp.533–536.

Russakovsky, O. et al., 2015. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3), pp.211–252.

Russell, B.C. et al., 2008. LabelMe: a database and web-based tool for image annotation. *International journal of computer vision*, 77(1-3), pp.157–173.

Saeedi, S. et al., 2016. Multiple-robot simultaneous localization and mapping: A review. *Journal of Field Robotics*, 33(1), pp.3–46.

Salakhutdinov, R. & Hinton, G., 2009. Deep boltzmann machines. In *Artificial intelligence and statistics*. PMLR, pp. 448–455.

Sanders, J. & Kandrot, E., 2010. *CUDA by example: an introduction to general-purpose GPU programming*, Addison-Wesley Professional.

Sandler, M. et al., 2018. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. pp. 4510–4520.

Schelter, S. et al., 2018. On challenges in machine learning model management. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, pp.5–15.

Schulzrinne, H., Rao, A. & Lanphier, R., 1998. Real time streaming protocol (RTSP). Available at: https://tools.ietf.org/html/rfc2326.

Se, S. & Brady, M., 2000. Vision-based detection of staircases. In *Fourth Asian Conference on Computer Vision ACCV*. pp. 535–540.

Sefraoui, O., Aissaoui, M. & Eleuldj, M., 2012. OpenStack: toward an open-source solution for cloud computing. *International Journal of Computer Applications*, 55(3), pp.38–42.

Seguı, S. et al., 2016. Generic feature learning for wireless capsule endoscopy analysis. *Computers in biology and medicine*, 79, pp.163–172.

Sekuboyina, A.K., Devarakonda, S.T. & Seelamantula, C.S., 2017. A convolutional neural network approach for abnormality detection in wireless capsule endoscopy. In *2017 IEEE 14th International Symposium on Biomedical Imaging (ISBI 2017)*. IEEE, pp. 1057–1060.

Sermanet, P. et al., 2013. Overfeat: Integrated recognition, localization and detection using convolutional networks. *arXiv preprint arXiv:1312.6229*.

Shah, N.F.M.N. & Ghazali, M., 2018. A systematic review on digital technology for enhancing user experience in museums. In *International Conference on User Science and Engineering*. Springer, pp. 35–46.

Shaham, T.R., Dekel, T. & Michaeli, T., 2019. Singan: Learning a generative model from a single natural image. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. pp. 4570–4580.

Sharma, T. et al., 2019. Fuzzy based Pooling in Convolutional Neural Network for Image Classification. In *2019 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*. IEEE, pp. 1–6.

Shi, C., Wang, Y. & Yang, J., 2010. A local obstacle avoidance method for mobile robots in partially known environment. *Robotics and Autonomous Systems*, 58(5), pp.425–434.

Shin, Y., Qadir, H.A. & Balasingham, I., 2018. Abnormal Colon Polyp Image Synthesis Using Conditional Adversarial Networks for Improved Detection Performance. *IEEE Access*, 6, pp.56007–56017.

Silva, J. et al., 2014. Toward embedded detection of polyps in wce images for early diagnosis of colorectal cancer. *International journal of computer assisted radiology and surgery*, 9(2), pp.283–293.

Simard, P.Y. et al., 2003. Best practices for convolutional neural networks applied to visual document analysis. In *Icdar*. Citeseer.

Simonyan, K. & Zisserman, A., 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

Snoek, J., Larochelle, H. & Adams, R.P., 2012. Practical bayesian optimization of machine learning algorithms. *arXiv preprint arXiv:1206.2944*.

Sommen, F. van der et al., 2016. Computer-aided detection of early neoplastic lesions in Barrett’s esophagus. *Endoscopy*, 48(07), pp.617–624.

Springenberg, J.T. et al., 2014. Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806*.

Srivastava, N. et al., 2014. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1), pp.1929–1958.

Stock, S., Erler, C. & Stork, W., 2018. Realistic simulation of progressive vision diseases in virtual reality. In *Proceedings of the 24th ACM Symposium on Virtual Reality Software and Technology*. ACM.

Sutton, R., 1986. Two problems with back propagation and other steepest descent learning procedures for networks. In *Proceedings of the Eighth Annual Conference of the Cognitive Science Society, 1986*. pp. 823–832.

Swain, P., 2008. The future of wireless capsule endoscopy. *World journal of gastroenterology: WJG*, 14(26), p.4142.

Szegedy, C. et al., 2015. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. pp. 1–9.

Szegedy, C. et al., 2017. Inception-v4, inception-resnet and the impact of residual connections on learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*.

Szegedy, C. et al., 2016. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. pp. 2818–2826.

Tajbakhsh, N., Gurudu, S.R. & Liang, J., 2015. Automatic polyp detection in colonoscopy videos using an ensemble of convolutional neural networks. In *2015 IEEE 12th International Symposium on Biomedical Imaging (ISBI)*. IEEE, pp. 79–83.

Tan, M. et al., 2019. Mnasnet: Platform-aware neural architecture search for mobile. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. pp. 2820–2828.

Tan, M. & Le, Q., 2019. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International Conference on Machine Learning*. PMLR, pp. 6105–6114.

Tan, M. & Le, Q.V., 2019. Mixconv: Mixed depthwise convolutional kernels. *arXiv preprint arXiv:1907.09595*.

Tao, Y., Ding, L. & Ganz, A., 2017. Indoor Navigation Validation Framework for Visually Impaired Users. *IEEE Access*, 5, pp.21763–21773.

Tapu, R., Mocanu, B. & Zaharia, T., 2018. Wearable assistive devices for visually impaired: A state of the art survey. *Pattern Recognition Letters*.

Theodoridis, S. & Koutroumbas, K., 2009. *Pattern Recognition (Fourth Edition)* Fourth Edition. S. Theodoridis & K. Koutroumbas, eds., Boston: Academic Press. Available at: http://www.sciencedirect.com/science/article/pii/B9781597492720500037.

Tian, S. et al., 2018. A Unified Framework for Tracking Based Text Detection and Recognition from Web Videos. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(3), pp.542–554.

Tighe, J. & Lazebnik, S., 2010. Superparsing: scalable nonparametric image parsing with superpixels. In *European conference on computer vision*. Springer, pp. 352–365.

Tulyakov, S. et al., 2018. Mocogan: Decomposing motion and content for video generation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. pp. 1526–1535.

Uijlings, J.R. et al., 2013. Selective search for object recognition. *International journal of computer vision*, 104(2), pp.154–171.

Ulyanov, D. et al., 2016. Texture Networks: Feed-forward Synthesis of Textures and Stylized Images. *CoRR*, abs/1603.03417. Available at: http://arxiv.org/abs/1603.03417.

Vaishnav, D., Rao, B.R. & Bade, D., 2021. Wearable Assistance Device for the Visually Impaired. In *Advances in Machine Learning and Computational Intelligence*. Springer, pp. 667–676.

Vapnik, V., 2013. *The nature of statistical learning theory*, Springer science & business media.

Vasilakakis, M. et al., 2016. Weakly-supervised lesion detection in video capsule endoscopy based on a bag-of-colour features model. In *International workshop on computer-assisted and robotic endoscopy*. Springer, pp. 96–103.

Vasilakakis, M.D. et al., 2018. Weakly supervised multilabel classification for semantic interpretation of endoscopy video frames. *Evolving Systems*, pp.1–13.

Vemuri, A.S., 2019. Survey of Computer Vision and Machine Learning in Gastrointestinal Endoscopy. *arXiv e-prints*, p.arXiv:1904.13307.

Vinoski, S., 2006. Advanced message queuing protocol. *IEEE Internet Computing*, 10(6), pp.87–89.

Viola, P., Jones, M. & others, 2001. Robust real-time object detection. *International journal of computer vision*, 4(34-47), p.4.

Voigt, P. & Bussche, A. Von dem, 2017. The eu general data protection regulation (gdpr). *A Practical Guide, 1st Ed., Cham: Springer International Publishing*.

Wang, D. et al., 2020. UAV environmental perception and autonomous obstacle avoidance: A deep learning and depth camera combined solution. *Computers and Electronics in Agriculture*, 175, p.105523.

Wang, P. et al., 2013. Design and implementation of a web-service-based public-oriented personalized health care platform. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 43(4), pp.941–957.

Wang, S. et al., 2015. Computer aided endoscope diagnosis via weakly labeled data mining. In *2015 IEEE International Conference on Image Processing (ICIP)*. IEEE, pp. 3072–3076.

Wang, S. et al., 2016. Computer-aided endoscopic diagnosis without human-specific labeling. *IEEE Transactions on Biomedical Engineering*, 63(11), pp.2347–2358.

Wang, Z. et al., 2004. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, 13(4), pp.600–612.

Wang, Z., She, Q. & Ward, T.E., 2019. Generative adversarial networks in computer vision: A survey and taxonomy. *arXiv preprint arXiv:1906.01529*.

Waters, B., 2005. Software as a service: A look at the customer benefits. *Journal of Digital Asset Management*, 1(1), pp.32–39.

Wei, L.-Y. et al., 2009. State of the art in example-based texture synthesis. In *Eurographics 2009, State of the Art Report, EG-STAR*. pp. 93–117.

Weil, S.A. et al., 2006. Ceph: A scalable, high-performance distributed file system. In *Proceedings of the 7th symposium on Operating systems design and implementation*. pp. 307–320.

Weiss, M. et al., 2020. Navigation Agents for the Visually Impaired: A Sidewalk Simulator and Experiments. In L. P. Kaelbling, D. Kragic, & K. Sugiura, eds. *Proceedings of the Conference on Robot Learning*. Proceedings of Machine Learning Research. PMLR, pp. 1314–1327. Available at: http://proceedings.mlr.press/v100/weiss20a.html.

WHO, 2018. World Health Organization - Blindness and visual impairement. Available at: http://www.who.int/news-room/fact-sheets/detail/blindness-and-visual-impairment.

Wilcoxon, F., 1947. Probability tables for individual comparisons by ranking methods. *Biometrics*, 3(3), pp.119–122.

Wimmer, G. et al., 2016. Convolutional neural network architectures for the automated diagnosis of celiac disease. In *International Workshop on Computer-Assisted and Robotic Endoscopy*. Springer, pp. 104–113.

Wu, B. et al., 2019. Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. pp. 10734–10742.

Wyrkabkiewicz, K., Tarczewski, T. & Niewiara, L., 2020. Local Path Planning for Autonomous Mobile Robot Based on APF-BUG Algorithm with Ground Quality Indicator. In *Advanced, Contemporary Control*. Springer, pp. 979–990.

Xiao, H., Rasul, K. & Vollgraf, R., 2017. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*.

Xiao, J. et al., 2010. Sun database: Large-scale scene recognition from abbey to zoo. In *2010 IEEE computer society conference on computer vision and pattern recognition*. IEEE, pp. 3485–3492.

Xiao, K. et al., 2020. EdgeABC: An architecture for task offloading and resource allocation in the Internet of Things. *Future Generation Computer Systems*, 107, pp.498–508.

Xie, S. et al., 2017. Aggregated residual transformations for deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. pp. 1492–1500.

Xu, Q.-L., Yu, T. & Bai, J., 2017. The mobile robot path planning with motion constraints based on Bug algorithm. In *2017 Chinese Automation Congress (CAC)*. IEEE, pp. 2348–2352.

Yang, K. et al., 2018. Unifying terrain awareness for the visually impaired through real-time semantic segmentation. *Sensors (Switzerland)*, 18(5), pp.1–32.

Yang, L. et al., 2021. CondenseNet V2: Sparse Feature Reactivation for Deep Networks. *arXiv preprint arXiv:2104.04382*.

Yang, T.-J. et al., 2018. Netadapt: Platform-aware neural network adaptation for mobile applications. In *Proceedings of the European Conference on Computer Vision (ECCV)*. pp. 285–300.

Yao, K., 2013. *Zoom gastroscopy: Magnifying endoscopy in the stomach*, Springer Science & Business Media.

Yi, Z. et al., 2017. Dualgan: Unsupervised dual learning for image-to-image translation. In *Proceedings of the IEEE international conference on computer vision*. pp. 2849–2857.

Yildirim, O. & Baloglu, U., 2019. REGP: A New Pooling Algorithm for Deep Convolutional Neural Networks. *Neural Network World*, 29(1), pp.45–60.

Yu, D. et al., 2014. Mixed pooling for convolutional neural networks. In *International Conference on Rough Sets and Knowledge Technology*. Springer, pp. 364–375.

Yu, T. & Zhu, H., 2020. Hyper-parameter optimization: A review of algorithms and applications. *arXiv preprint arXiv:2003.05689*.

Yu, X. et al., 2018. AR Marker Aided Obstacle Localization System for Assisting Visually Impaired. In *2018 IEEE International Conference on Electro/Information Technology (EIT)*. IEEE, pp. 271–276.

Yuan, Y., Li, B. & Meng, M.Q.-H., 2015. Improved bag of feature for automatic polyp detection in wireless capsule endoscopy images. *IEEE Transactions on automation science and engineering*, 13(2), pp.529–535.

Yüksel, M.E. & Borlu, M., 2009. Accurate segmentation of dermoscopic images by image thresholding based on type-2 fuzzy logic. *IEEE Transactions on Fuzzy Systems*, 17(4), pp.976–982.

Zaharia, M. et al., 2010. Spark: Cluster computing with working sets. *HotCloud*, 10(10-10), p.95.

Zapf, M.P.H. et al., 2016. Assistive peripheral phosphene arrays deliver advantages in obstacle avoidance in simulated end-stage retinitis pigmentosa: a virtual-reality study. *Journal of Neural Engineering*, 13(2), p.026022.

Zeiler, M.D., 2012. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*.

Zeiler, M.D. & Fergus, R., 2013. Stochastic pooling for regularization of deep convolutional neural networks. *arXiv preprint arXiv:1301.3557*.

Zeiler, M.D. & Fergus, R., 2014. Visualizing and understanding convolutional networks. In *European conference on computer vision*. Springer, pp. 818–833.

Zhang, H. et al., 2019. Self-attention generative adversarial networks. In *International conference on machine learning*. PMLR, pp. 7354–7363.

Zhang, L., Gooya, A. & Frangi, A.F., 2017. Semi-supervised assessment of incomplete LV coverage in cardiac MRI using generative adversarial nets. In *International Workshop on Simulation and Synthesis in Medical Imaging*. Springer, pp. 61–68.

Zhang, R. et al., 2016. Automatic detection and classification of colorectal polyps by transferring low-level CNN features from nonmedical domain. *IEEE journal of biomedical and health informatics*, 21(1), pp.41–47.

Zhang, S. et al., 2019. Static Gait Planning Method for Quadruped Robot Walking on Unknown Rough Terrain. *IEEE Access*, 7, pp.177651–177660.

Zhang, X. et al., 2018. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. pp. 6848–6856.

Zhao, Y. et al., 2018. Enabling People with Visual Impairments to Navigate Virtual Reality with a Haptic and Auditory Cane Simulation. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. ACM.

Zhou, Y. et al., 2017. A real-time and fully distributed approach to motion planning for multirobot systems. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 49(12), pp.2636–2650.

Zhou, Y. et al., 2018. Non-stationary texture synthesis by adversarial expansion. *arXiv preprint arXiv:1805.04487*.

Zhu, J.-Y. et al., 2017. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Proceedings of the IEEE international conference on computer vision*. pp. 2223–2232.

Zimmermann, H.-J., 2011. *Fuzzy set theory—and its applications*, Springer Science & Business Media.