Universidade do Minho
Escola de Ciências

Viviana Figueira Magalhães

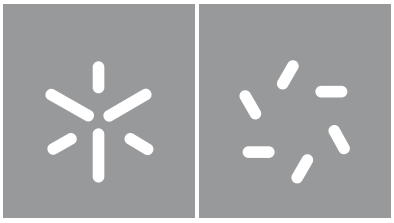
Self-Supervised Learning Techniques for Monitoring Industrial Spaces

janeiro de 2023

Self-Supervised Learning Techniques for Monitoring Industrial Spaces

Viviana Magalhães

UMinho | 2023



Universidade do Minho
Escola de Ciências

Viviana Figueira Magalhães

Self-Supervised Learning Techniques for Monitoring Industrial Spaces

Dissertação de Mestrado
em Matemática e Computação

Trabalho efetuado sob a orientação dos
Professora Doutora Maria Fernanda Pires Costa
Professor Doutor Manuel João Oliveira Ferreira

janeiro de 2023

Acknowledgements

First of all, I would like to thank my supervisors, Professor Doctor Fernanda Costa and Professor Doctor Luís Ferrás for their availability, kindness, and support during the realization of this dissertation.

I also want to express my gratitude to Neadvance and my supervisor Professor Doctor Manuel João Ferreira for the great experience and for suggesting the theme of this thesis. Furthermore, I would like to thank my colleagues at Neadvance for the way they integrated me into the team, for all the kindness, and availability to help, and for constantly asking if it was already done, in the final tough stretch. I especially want to thank Tiago Pinto for his patience with me, for always finding time to help me even on very busy days, for checking up on me constantly, and for all the knowledge and motivation given. I would also like to thank my desk buddy Vitor Figueiredo for all the help, for answering my 1-minute and 1-hour questions, for looking at detail, and for all the kindness, support, and good advice. I was happy to be part of your team!

To all my colleagues who have accompanied me on my academic journey. A special thanks to Vitor Hugo, who accompanied me from day one in this experience at Neadvance. Thank you for all the support and availability, for helping me without me having to ask, for the laughter and, above all, for the positivity that I so much needed.

I want to thank my aunts: Maria, Isabel and Catarina for all the love, support and kind words. To my godfather Alberto, thank you for watching over me up in heaven. A special thank you to my favorite children Valentina, Eva and Tomás for their genuine love and laughter and for reminding me to appreciate the little things.

Thank you to my friends, especially: Adriana, for remembering and being a sun on dark days; Isabel, for being a good friend and making me feel stronger; Helena, for the laughter and reminding me to enjoy the journey; Patrícia, who has been by my side since the 5th grade. Thank you for making all our steps less scary; Ponto, for being a good friend and always finding time; Nacional, for believing in me when I wouldn't believe in myself.

I want to thank my boyfriend Cristóvão for all the support, his daily patience, putting up with all my dramas and nonsense, and for always listening and having something peaceful and nice to say.

To my brothers Philipe and Nando, who make life better, for all the love, laughter and companionship. For cheering me up on weaker days and helping me become the person I am today. A thank you will never be enough.

Last, but most importantly, a big thank you to my mother Martinha, to whom I dedicate all my work. Thank you for always believing in me, for applauding all my victories no matter how small they were, thank you for all the effort that allowed me to be where I am today, thank you for passing on so much joy, positivity and light, and thank you for always being there. For you mother!

Statement of Integrity

I hereby declare having conducted this academic work with integrity. I confirm that I have not used plagiarism or any form of undue use of information or falsification of results along the process leading to its elaboration. I further declare that I have fully acknowledged the Code of Ethical Conduct of the University of Minho.

Resumo

Este documento é uma Dissertação de Mestrado com o título "*Self-Supervised Learning Techniques for Monitoring Industrial Spaces*"e foi realizada e ambiente empresarial na empresa Neadvance - Machine Vision S.A. em conjunto com a Universidade do Minho.

Esta dissertação surge de um grande projeto que consiste no desenvolvimento de uma plataforma de monitorização de operações específicas num espaço industrial, denominada SMARTICS (*Plataforma tecnológica para monitorização inteligente de espaços industriais abertos*). Este projeto continha uma componente de investigação para explorar um paradigma de aprendizagem diferente e os seus métodos - *self-supervised learning*, que foi o foco e principal contributo deste trabalho. O *supervised learning* atingiu um limite, pois exige anotações caras e dispendiosas. Em problemas reais, como em espaços industriais nem sempre é possível adquirir um grande número de imagens. O *self-supervised learning* ajuda nesses problemas, extraindo informações dos próprios dados e alcançando bom desempenho em conjuntos de dados de grande escala. Este trabalho fornece uma revisão geral da literatura sobre a estrutura de *self-supervised learning* e alguns métodos. Também aplica um método para resolver uma tarefa de classificação para se assemelhar a um problema em um espaço industrial.

Palavras-chave: Visão por computador, *Deep Learning*, *Self-Supervised Learning*, *Pretext tasks*, *Contrastive Learning*, Espaços Industriais

Abstract

This document is a Master's Thesis with the title "*Self-Supervised Learning Techniques for Monitoring Industrial Spaces*" and was carried out in a business environment at Neadvance - Machine Vision S.A. together with the University of Minho.

This dissertation arises from a major project that consists of developing a platform to monitor specific operations in an industrial space, named SMARTICS (*Plataforma tecnológica para monitorização inteligente de espaços industriais abertos*). This project contained a research component to explore a different learning paradigm and its methods - self-supervised learning, which was the focus and main contribution of this work. Supervised learning has reached a bottleneck as they require expensive and time-consuming annotations. In real problems, such as in industrial spaces it is not always possible to require a large number of images. Self-supervised learning helps these issues by extracting information from the data itself and has achieved good performance in large-scale datasets. This work provides a comprehensive literature review of the self-supervised learning framework and some methods. It also applies a method to solve a classification task to resemble a problem in an industrial space and evaluate its performance.

Keywords: Computer Vision, Deep Learning, Self-Supervised Learning, Pretext tasks, Contrastive Learning, Industrial Spaces

Contents

List of Figures

List of Tables

List of Abbreviations

AdaGrad Adaptive Gradient Algorithm.

ADAM Adaptive Moment Estimation.

AI Artificial Intelligence.

ANNs Artificial Neural Networks.

BN Batch Normalization.

BP Back-Propagation.

CL Contrastive Learning.

CNNs Convolutional Neural Networks.

CV Computer Vision.

DL Deep Learning.

DNNs Deep Neural Networks.

ELU Exponential Linear Unit.

FCL Fully Connected Layer.

FN False Negatives.

FP False Positives.

ILSVRC ImageNet Large-Scale Visual Recognition Competition.

lr Learning Rate.

LReLU Leaky Rectifier Linear Unit.

MAE Mean Absolute Error.

mAP mean Average Precision.

MCC Matthews Correlation Coefficient.

mIU mean Intersection over Union.

ML Machine Learning.

MLP Multilayer Perceptron.

MSE Mean Squared Error.

Noisy ReLU Noisy Rectifier Linear Unit.

OCR Optical Character Recognition.

PReLU Parametric Linear Unit.

ReLU Rectifier Linear Unit.

ResNet Residual Network.

RL Reinforcement Learning.

RMSE Root Mean Squared Error.

RMSProp Root Mean Square Propagation.

RReLU Randomized Leaky Rectifier Linear Unit.

Semi-SL Semi-Supervised Learning.

SGD Stochastic Gradient Descent.

SL Supervised Learning.

SSL Self-Supervised Learning.

SwAV Swapping Assignments between multiple Views.

Tanh Hyperbolic Tangent.

TL Transfer Learning.

TN True Negatives.

TP True Positives.

UL Unsupervised Learning.

W&B Weights & Biases.

1 Introduction

This dissertation is a research component of a major project that consists of developing a platform to monitor specific operations in an industrial space, named SMARTICS (*Plataforma tecnológica para monitorização inteligente de espaços industriais abertos*). It was developed in a business environment at Neadvance - Machine Vision S.A. company. This component was mainly to research and explore the new paradigm of self-supervised learning and use it to solve a real problem in an industrial space context.

The rise of algorithms capable of detecting, recognizing, and tracking objects in uncontrolled environments, such as indoor and outdoor surveillance systems, traffic control of people and vehicles, intelligent parking, etc., opens a temporal window for the application of these technologies in the logistics areas in industrial environments, allowing a more efficient and "intelligent" management of the factory space. All of these surveillance and monitoring activities are currently performed manually or with CCTV systems that monitor critical areas. CCTV systems are connected to monitors at a control post, where one or more operators manually perceive the monitored areas and detect abnormal situations. In this context, human errors are very common, increasing depending on the quantity of video images and the size of the industrial environment to be monitored.

In addition to the need for constant and more efficient monitoring, maximizing the use of factory space has also become a growing concern, not only because of the high cost of construction and maintenance, but also to optimize the cadence of the production process and act quickly in case of malfunctions.

In this way, the use of "intelligent" logistics solutions, capable of increasing the ambient density, allows reducing the operating and maintenance costs of the production facilities. This type of solution also leads to a reduction in waiting times for production and machine downtime, reducing the interference of people in logistical tasks.

Deep Learning has brought significant development in automated computer vision systems such as object detection [1], image classification [2], and image segmentation [3], which is useful for monitoring these industrial spaces. However, the success of these systems relies on supervised learning that requires a large amount of labeled data. In many situations, it is not possible to acquire a big amount of images, as is the case of industrial spaces. As a result, a large research effort is currently focused on systems that can adapt to new conditions without leveraging a large amount of expensive and time-consuming supervision. One alternative to overcome this is to use the self-supervised learning paradigm [4, 5, 6]. Self-supervised learning constructs feature representations without manual annotations using pretext tasks, which allows models trained in these tasks to extract useful information that can later improve downstream tasks. Further self-supervised learning methods use contrastive learning to push positive instances closer together, and negative ones further apart, in

the embedding space [7, 8]. These methods have achieved great performance closing the gap with supervised learning. Researchers proclaim that the next AI revolution will not be supervised but self-supervised [8].

1.1 Objectives of the Dissertation

The main objective of this dissertation was the implementation of a vision system that could classify if a shelf was empty or not, to assimilate to an industrial space context, in a self-supervised learning setting. To achieve such a goal, the following tasks were performed:

- Study of the self-supervised learning paradigm as well as the state-of-the-art methods;
- Creation of datasets to resemble industrial space problems;
- Training of models, varying parameters;
- Evaluation of the trained models and selection of the best parameters.

1.2 Dissertation's Structure

This dissertation is organized into 6 chapters: Introduction; Basic Concepts; Self-Supervised Learning; Development Tools and Datasets; Experiments and Results; and Conclusions and Future Work. The contents of each chapter are shortly described in the following topics:

- Chapter 1: Introduction

  In this chapter, the problem addressed is presented and the objective of this dissertation. The structure of the dissertation is also highlighted;

- Chapter 2: Basic Concepts

  This chapter covers the theoretical and basic concepts that lead to the focus of the thesis. Starting with the main definitions of Machine Learning, Deep Learning, and Artificial Neural Networks, going further to how Convolutional Neural Networks work. Followed by their hyperparameters and some architectures and finishing with a description of Computer Vision history;

- Chapter 3: Self-Supervised Learning

  This chapter is a central part of this thesis. It describes how Self-Supervised Learning works and explains some state-of-the-art models of this learning paradigm;

- Chapter 4: Development Tools and Datasets

  This chapter presents the tools and datasets used for the development of the self-supervised learning method;

- Chapter 5: Experiments and Results

  In this chapter the different experiments are reported and the results displayed and discussed.

- Chapter 6: Conclusions and Future Work

  In this chapter are presented the conclusions and future work of this thesis.

2 Basic concepts

This chapter serves as a theoretical introduction and covers basic concepts to help understand the theory and the methods applied in this thesis. We start with a broad overview on Machine Learning and it's types. Followed by Deep Learning and Computer Vision. We then enter the scientific background of these concepts and focus on artificial neural networks, especially convolutional neural networks.

2.1 Machine Learning

At the beginning of the 20^{th} century, science fiction introduced the world to the concept of artificially intelligent robots. By the 1950s, we had a generation of mathematicians, scientists and philosophers who had culturally internalized the concept of artificial intelligence in their minds. One of them was Alan Turing, a British mathematician and computer scientist who wondered that if humans use available information and reason in order to solve problems and make decisions, then machines could do the same. Much has evolved since then, and artificial intelligence has already had a profound impact on the world. Weather forecasting, email spam filtering, Google search predictions and image classification are just a few examples. What these technologies have in common are Machine Learning algorithms that allow them to learn and consequently respond in real-time. But what is Machine Learning ?

First of all **Artificial Intelligence** (AI) is a field of computer science concerned with not just understanding but also building intelligent entities — machines that can compute how to act effectively and safely in a wide variety of novel situations [9]. **Machine Learning** (ML) is a subset of AI which allows computers to learn from data without being explicitly programmed. The goal of ML is to design methods that learn using observations of the real world, without explicit definition of rules by humans and improve automatically through experience [10]. A vast set of ML algorithms has been proposed to cover the wide variety of data and problem types.

2.2 Machine Learning Pipeline

A ML pipeline can be divided into three main steps: Data Collection, Data Modelling, and Deployment. The first step is gathering data. Data is information collected to be examined and used to help make decisions [11]. This can be a spreadsheet with multiple rows and columns of information, text, images, or even audio files. The machines learn from the given data, so it is very important to collect good and reliable data so that the ML model can find the right patterns. The next stage is modelling, where a ML algorithm is taught to gain insights from the collected data. And finally, deployment, where the models are deployed into production.

In data modelling, there are 4 phases as shown in the Figure 1: Problem Definition, Data Understanding and Processing, Modelling, and Evaluation.

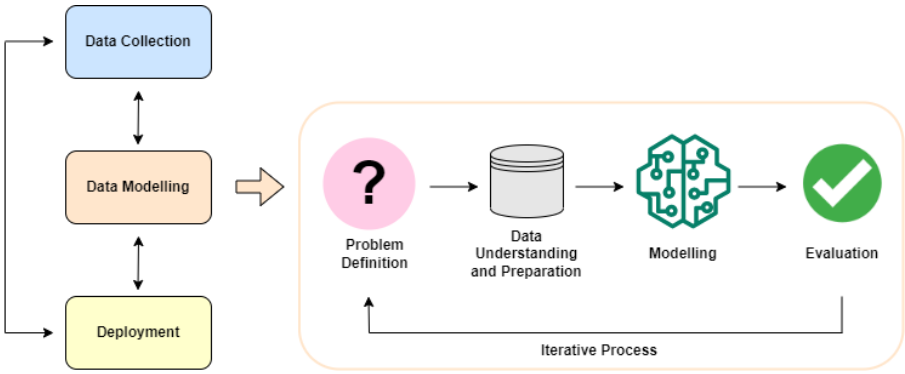


Figure 1: Machine Learning Pipeline

2.2.1 Problem Definition

The first step is aligning the problem you're trying to solve to a machine learning paradigm. These methods can be divided into three primary approaches: supervised learning, unsupervised learning and reinforcement learning [12] (Figure 2).

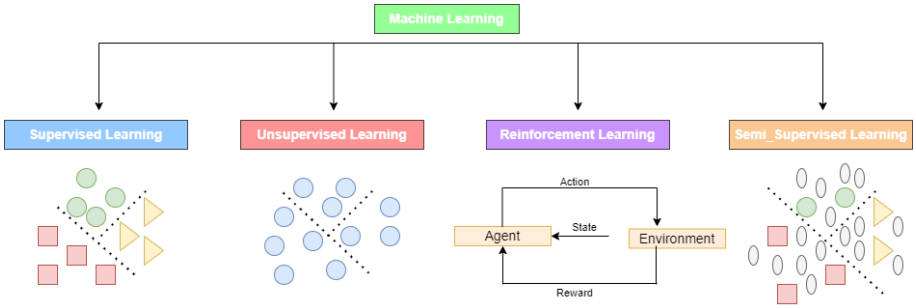


Figure 2: Types of Machine Learning

1. **Supervised Learning**

   Supervised learning (SL) is the most common type of machine learning. In this type of learning, each example in the dataset has a corresponding label. The dataset is the collection of labelled examples $\{(x^{(i)}, y^{(i)})\}_{i=1}^{N}$. Suppose that each element among N is a feature vector $x^{(i)}$, i.e. a vector in which each dimension $j = 1, ..., d$ contains a value describing the instance. This value designated as $x_j^{(i)}$ is called a feature. So if each $x^{(i)}$ in our dataset represents a fruit, then the first feature, $x_1^{(i)}$, might contain the colour of the fruit, the second feature, $x_2^{(i)}$, the weight in grams, $x_3^{(i)}$ the width and so on.

The feature at position j in a feature vector $x^{(i)}$ contains the same type of information in all the examples of the dataset. That is, if $x_2^{(i)}$ contains the weight in grams in an example $x^{(i)}$, then $x_2^{(k)}$ also contains the weight in grams in each example $x^{(k)}$, $k = 1, ..., N$ [13]. The input can be a feature vector, but can also be more complex data such as images, that we will see further on. Each pair $(x^{(i)}, y^{(i)})$ was generated by an unknown function $y = f(x)$. The goal is to find a function h that approximates to the true function f. The issue is not how well the function performs in the training set, but how well it handles inputs x it has not seen before. We evaluate this with a second sample of $(x^{(i)}, y^{(i)})$ pairs, which we call the test set. h generalizes well if it accurately predicts the outputs $y^{(i)}$ of the test set (Figure 3). In SL we can have two types of problems [12]:

(a) **Classification**, which is assigning a label to an unlabelled example where the label belongs to a finite set of categories. It's a binary classification problem if it only has two categories and a multi-class classification problem if it has more than two.

(b) **Regression**, which is predicting a real-valued label given an unlabelled example, where the label is a real or a continuous value.

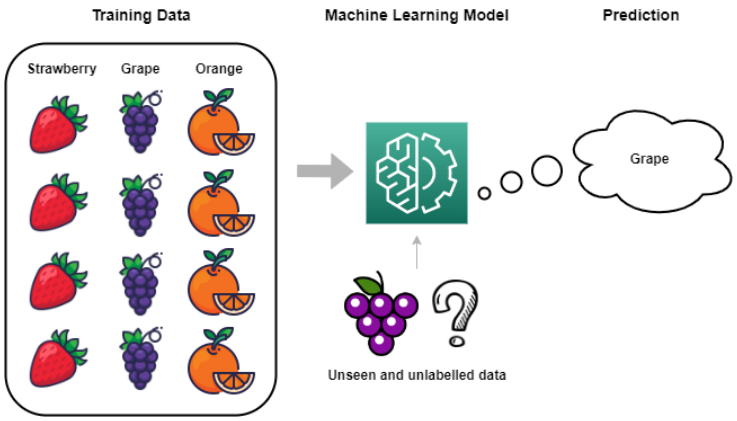


Figure 3: Illustrated Example of a Supervised Learning Workflow - Classification problem

2. **Unsupervised Learning**

In Unsupervised Learning (UL), the dataset is a collection of unlabelled examples $\{x^{(i)}\}_{i=1}^{N}$. Assume again that each $x^{(i)}$ is a feature vector and the goal is to build a model that takes a feature vector as input and transforms it either to another vector or to a value that can be used to solve a practical problem [13]. The machine uses unlabelled data and acts on the information from the data without guidance, grouping the unsorted information according to patterns, similarities and differences without any prior training. It is

defined to find the hidden structure in the data itself. For example, suppose the model receives images of different fruits like bananas, pears, cherries and blueberries that it has never seen before. So the machine has no idea of the characteristics of these fruits, so we cannot categorise them as $'banana'$, $'cherry'$, etc. However, it can categorise them based on their resemblance and put together images of fruits that have the same shape and colour, for example (Figure 4).

Unsupervised learning is classified into two categories [14]:

(a) **Clustering** is applied to group data based on different patterns our machine model finds, such as grouping different fruits.

(b) **Association** is a rule-based ML technique that finds out useful relations between parameters of a large data set, such as: people that buy X also tend to buy Y.

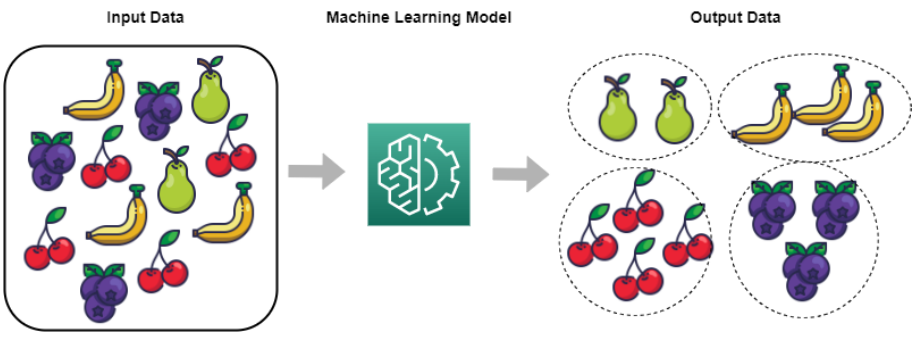


Figure 4: Illustrated Example of an Unsupervised Learning Workflow - Clustering

3. **Reinforcement Learning**

Reinforcement Learning (RL) is based on developing a system that improves its performance by receiving feedback from the environment at each iteration. There is an agent that uses insights from the environment to perform actions with the goal of maximising the cumulative reward. It learns from the environment by interacting with it and receives rewards or punishments based on its actions [15]. Let us take a simple example using Robert Tryon's experiment testing the ability of successive generations of rats in completing a maze [16]. So suppose we want a mouse to complete a maze. In this case, our agent is the mouse, the environment is the maze, the action is the movement of the mouse through the maze, moving left, right, etc., and there is a state that represents the position of the mouse in the maze in each iteration. Eating the pieces of food it finds when it goes the right way is its reward and motivates it to explore further. Its punishment is, for example, getting stuck in a narrow hole. The mouse's goal is to maximise the rewards,

i.e. to eat the largest amount of food (Figure 5).

Reinforcement Learning is a powerful tool that can help increase automation and optimize sophisticated systems such as robotics, autonomous driving tasks and manufacturing.

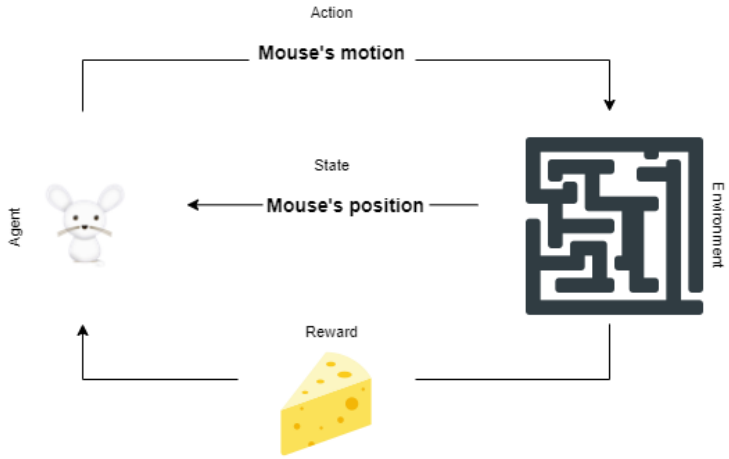


Figure 5: Illustrated Example of an Reinforcement Learning Workflow

It's important to mention that modern research is not limited to these Machine Learning approaches. There are many other types of learning. The ones that are used during this work are:

- **Semi-Supervised Learning**
- **Transfer Learning**
- **Self-Supervised Learning**

Labelled data is often difficult, expensive, or/and time-consuming to obtain, as they require the effort of human annotators. In industrial scenarios it is not always possible to have a large amount of data and when possible they are normally unbalanced. On the other hand, unlabelled data is relatively easy to collect. **Semi-Supervised Learning** (Semi-SL) addresses this problem by using both labelled and unlabelled data to learn from. The portion of labeled examples is usually quite small compared to the unlabelled example. The goal of Semi-SL is to understand how combining labeled and unlabelled data may change the learning behaviour, and design algorithms that take advantage of this combination [17] (Figure 6 (a)).

Humans recognise and apply relevant knowledge from previous experiences when confronted with new tasks. The more a new task is related to a previous experience, the easier it is to solve. In contrast, common ML

algorithms deal with isolated tasks. **Transfer learning** (TL) improves ML by transferring knowledge learned in tasks and using it to improve learning in a related target task. Transfer learning is thus the improvement of learning in a new task through the transfer of knowledge from a related task that has already been learned [18] (Figure 6 (b)).

Self-Supervised Learning is the focus of the thesis and will be fully covered in chapter 3.

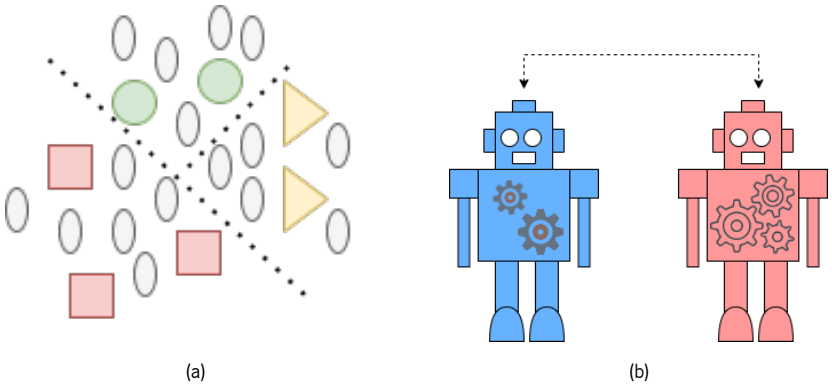


Figure 6: Illustrative Semi-Supervised Learning (a) and Transfer Learning (b)

2.2.2 Data Understanding and Processing

The second step is to examine the data, understand it in the context of the problem, and apply proper pre-processing, which is the process of transforming raw data into clean data suitable for modelling. Data pre-processing depends heavily on the type of data. If we have structured data such as an Excel spreadsheet with numeric and categorical features, there are techniques such as filling in missing values, normalizing the data, removing irrelevant features, converting categorical data into numeric data, or vice-versa [19]. The data can also be natural language text, audio files, or images. If we have images, for example, we can crop or resize them, or even remove images that we believe are not appropriate for the model to learn from. Depending on the problem and the data, we apply different processing and different set of techniques to better suit the data to the problem at hand.

2.2.3 Modelling

We now have clean data ready for modelling which we have to split into sets, as shown in Figure 7:

- **Training set**: A set used to train the model and teach it how to extract and process information;
- **Validation set**: A set used to validate the model in the training process ;

- **Test set**: A set of unseen data used only to assess the performance of the model.

The data can also be split in only training and test sets and in this case the tuning of the model parameters is done on the training set.

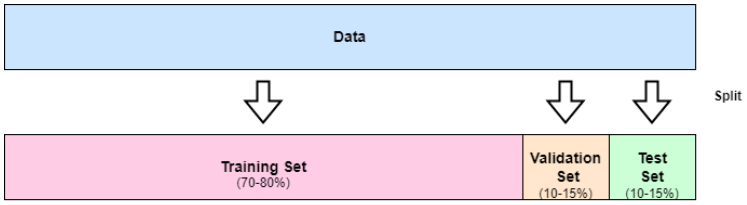


Figure 7: Data Splitting

First, we must choose the model according to the data and the problem at hand. For example, if it is a classification problem with supervised data, we can use a Decision Tree or a Support Vector Machine or K-Means if we want to cluster the data [14]. If it is unstructured data such as images or natural language text, we can choose deep models such as neural networks, which work best.

Once the model is selected, we feed it with the training data. When dealing with high complex models or high amounts of data, the model training requires/asks for a equally high time. Transfer learning reduces this problem. The next step is to improve the results obtained by tuning. Tuning a model involves changing the model's hyperparameters, which depend on the model chosen. Many models have different hyperparameters that can be adjusted. After improving the performance of the model by tuning the hyperparameters, it is time to see how it performs on the test set. This gives an indication of how the model will perform once used in production.

Since the model has never seen data in the test set, evaluating the model on it is a good way to see how well it generalizes. A good model will yield similar results on the training/validation set and on the test set. Overfitting and underfitting are examples of a model not being able to generalize well [20].

Overfitting occurs when a model learns the detail and noise in the training data so well that it negatively impacts the performance of the model on new unseen data. This means that the noise in the training data is picked up and learned as concepts by the model. As these concepts don't apply to the new data, this impacts negatively the models ability to generalize and consequently the model performs poorly on the test set [21]. This can happen for many reasons, such as:

- data leakage, when some of the test data leaks into the training data;
- the model is too complex for the data;

- the data has many features but a small number of training examples;
- too much noise in the training data.

Some solutions are using a simpler model, reducing the dimensionality of examples in the dataset, adding more training data or regularizing the model, and reduce the noise in the training data [13].

Underfitting is the opposite of overfitting. It refers to a model that can neither model the training data nor generalize to new data. An underfit model is not a suitable model and will have poor performance on the training data as it will not be able to learn [21]. This happens for example:

- if the test data is different to the data the model was trained on;
- the model is too simple for the data;
- the features of the data are not giving enough information.

This can be fixed by trying a different or more complex model, improving the existing model through hyperparameter tuning, training the model for longer or reducing the amount of features [13].

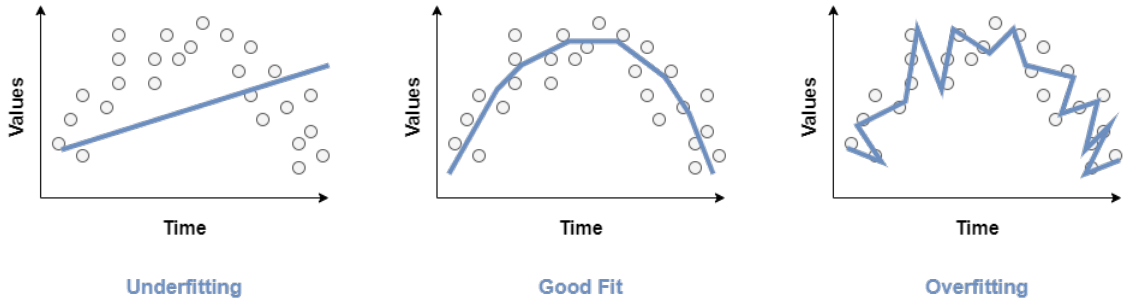


Figure 8: Illustrative examples of models fitting the data

2.2.4 Evaluation

After being trained, the model is evaluated by testing the performance of the model on the test set, which is unseen data. The evaluation is an estimate that we use to talk about how well we think the algorithm may do in practice. To calculate this, we use specific metrics, that depend on the problem at hands.

If it's a regression problem the commonly used metrics are [22]:

- **Mean Absolute Error (MAE)**: measures the average magnitude of errors in a set of predictions. It's the sum of the absolute differences between predictions and actual values.

$$MAE = \frac{1}{n}\sum_{j=1}^{n}|y_j - \hat{y}_j|$$

- **Mean Squared Error (MSE)**: consists of the average of squared differences between the prediction and the actual observation.

$$MSE = \frac{1}{n}\sum_{j=1}^{n}(y_j - \hat{y}_j)^2$$

- **Root Mean Squared Error (RMSE)**: consists of the square root of the average of squared differences between the prediction and the actual observation.

$$RMSE = \sqrt{\frac{1}{n}\sum_{j=1}^{n}(y_j - \hat{y}_j)^2}$$

Here, n is the number of observations, and y_j and $\hat{y}_j$ are the actual observation and the predicted value by the model, respectively.

When dealing with a classification problem, a confusion matrix (Figure 9) is useful to describe the performance of a model, as the values of this table formulate metrics. This table contains:

- **True Positives (TP)**: These refer to positive examples that were correctly labeled by the model.
- **True Negatives (TN)**: These are the negative examples that were correctly labeled by the model.
- **False Positives (FP)**: These refer to negative examples that were incorrectly labeled as positive.
- **False Negatives (FN)**: These are the positive examples that were incorrectly labeled as negative.

Each column in a confusion matrix represents an actual class, while each row represents a predicted class.

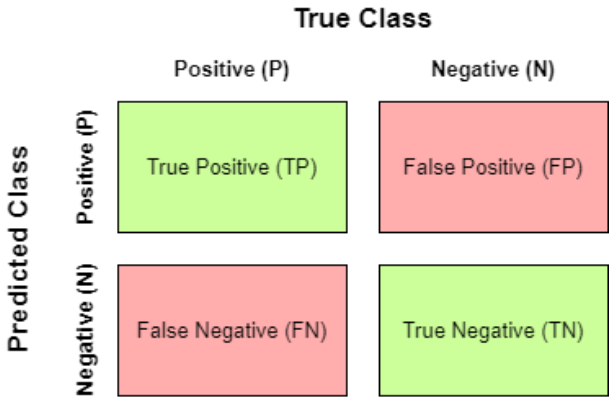


Figure 9: Confusion Matrix for binary classification

Given this let's look at the evaluation metrics [19]:

- **Accuracy** is calculated as the number of all correct predictions divided by the total number of observations. It's the percentage of examples that are correctly classified by the model.

$$\textit{Accuracy} = \frac{TP + TN}{TP + FP + TN + FN}$$

- **Recall** is a measure of completeness that determines the proportion of positive examples that are correctly identified. It is calculated by dividing the true positives by anything that should have been predicted as positive.

$$\textit{Recall} = \frac{TP}{TP + FN}$$

- **Precision** is a measure of exactness that quantifies the number of correct positive predictions made, this is, it determines the percentage of examples labeled as positive that are actually positive. It's calculated by dividing the true positives by anything that was predicted as a positive.

$$\textit{Precision} = \frac{TP}{TP + FP}$$

- **F_1 Score** is the harmonic mean of precision and recall. It can be used to obtain a more balanced view of performance, as it gives equal weight to precision and recall.

$$F_1 Score = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

  The F_β Score where β is a non-negative real number is a weighted measure of precision and recall. It assigns β times as much weight to both metrics.

$$F_\beta Score = \frac{(1 + \beta^2) \times Precision}{\beta^2 \times Precision + Recall}$$

- **Matthews Correlation Coefficient (MCC)**: When dealing with unbalanced datasets, most classifiers are biased towards the largest class [23]. To evaluate the performance of a model in this condition, the metric chosen must be unaffected by the unbalanced dataset issue. An effective solution is *Matthews Correlation Coefficient* (MCC) , a contingency matrix method of calculating the Pearson product-moment correlation coefficient between actual and predicted values [24]. MCC is defined as:

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}$$

  MCC gives a score close to 1 only when the binary predictor was able to predict the majority of positive and negative data instances [24] and -1 with a perfect misclassification.

2.2.5 Iterative Process

After applying these four steps, the model has been trained and given feedback on its performance. Based on the results obtained, we can change aspects of the previous steps to improve the performance of the model. To do this, we can add data, prepare the data differently, try a different model, modify the parameters of the model, and more. Since it is an iterative process, we can change what we think is best, test it, analyse the results and choose the best model.

2.3 Deep Learning

While ML algorithms have been around for a long time, the ability to automatically apply complex mathematical computations to large-scale data is a more recent development. Thanks to the increase of speed and memory of computers, ML techniques evolved to learn from a large bulk of training data. With this boost of computational power we can create neural networks with multiple layers, which are called *deep neural networks* (DNNs) and will be explained in the next chapter. **Deep Learning** (DL) is a subset of ML based on artificial neural networks that extracts high-level features from raw input using a variety of sequential nonlinear transformations organized in multiple layers [25]. Figure 10 illustrates the relation between AI, ML and DL.

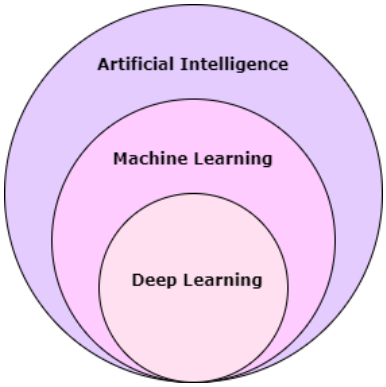


Figure 10: The relation between Artificial Intelligence, Machine Learning and Deep Learning

2.4 Artificial Neural Networks

Artificial Neural Networks (ANNs) are inspired by the human brain and aim to imitate biological neurons and the connection between them.

The biological neural network consists of a large number of interconnected nerve cells called neurons. Neurons have a simple three-part structure consisting of a cell body, a series of fibers called dendrites, and a single

long fiber called the axon (see Figure 11).

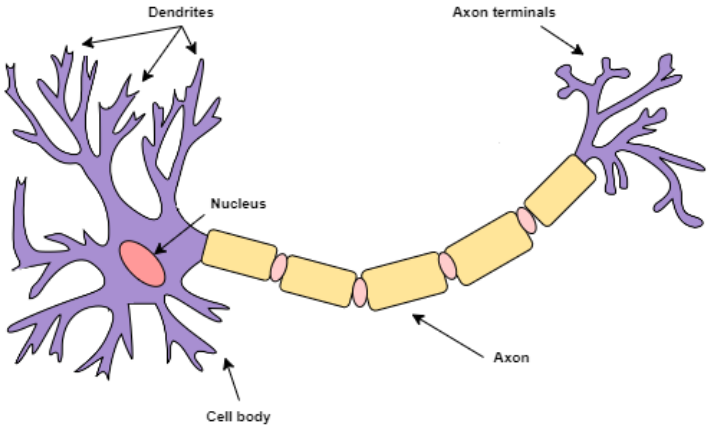


Figure 11: Illustrative biological neuron

The cell body of the neuron, which includes the neuron nucleus, is where most of the neural computation takes place. Neural activity is passed from one neuron to another in the form of electrical impulses that travel along the axon of the neuron by an electrochemical process. The axon can be thought of as a connecting wire. This transport process moves along the cell of the neuron, down the axon then through synaptic junctions across a synaptic space to the dendrites and/or soma of the next neuron at an average speed of 3 m/sec. Since a given neuron may have multiple synapses, a neuron can connect to many other neurons. Similarly, since there are many dendrites, a single neuron can also receive messages from many other neurons. In this way, the biological neural network is interconnected. Not all connections are equally weighted, some have a higher priority than others. Also, some are excitatory and others are inhibitory (to block the transmission of a message). These differences are caused by differences in chemistry and by the presence of chemical transmitters and modulatory substances within and near neurons, axons, and in the synaptic junction [26, 27]. Neuroscientists have discovered that the human brain learns by changing the strength of the synaptic connection between neurons when simulated repeatedly by the same impulse. The human brain is made up of about 100 billion neurons that are connected in complex ways that allow us to learn new tasks and perform regular activities. A single neuron performs only one simple modular function, which is to respond to the nerve activations coming from the transmitter neurons connected to its dendrite and to transmit its activation to the receiver neurons via axons. However, it is the composition of these simple functions that together can express complex functions. The structure of the biological neural system and the way it performs its functions inspired the idea of ANNs [28].

Analogous to the structure of the human brain, an ANN consists of a series of processing units - neurons.

Each neuron is connected to another neuron by means of directed communication links, each with an associated weight, just like biological neurons. ANNs can be described as a directed graph whose nodes correspond to neurons that perform the basic units of computation and whose edges correspond to the connection between the neurons [29, 30].

The basic motivation behind using an ANN model is to extract the most relevant features from the original attributes. By using a complex combination of inter-connected nodes, ANN models are able to extract much richer sets of features [28].

2.4.1 The Simple Perceptron

In 1958, Frank Rosenblatt, an American psychologist notable in the field of artificial intelligence, invented an artificial neuron - the *perceptron* [31]. A perceptron is a feed-forward neural network consisting of a single neuron that can receive multiple inputs and produce a single output. Feed-forward because the information flows only forward through the network from the input to the output. Perceptrons are used to classify linearly separable classes by finding an arbitrary m-dimensional hyperplane in the feature space that separates instances of two classes [27].

Figure 12 illustrates the basic architecture of a perceptron that takes n input attributes: $x_1, x_2, ..., x_n$, with $x_i \in \mathbb{R}$, and produces a binary output $\hat{y} \in \mathbb{R}$. Each attribute x_i is multiplied by a specific weight w_i. The weighted link is used to emulate the strength of a synaptic connection between neurons. These products are summed and fed to a nonlinear function, an activation function Φ. This function determines if the neuron is activated or not, if its value is above a certain threshold. The perceptron has an additional input called the bias. The job of the bias Θ is to shift the activation function to positive or negative values, making adjustments within neurons. Changing the bias value does not change the shape of the activation function, but together with the other weights determines when the perceptron fires. Training the perceptron aims at determining the optimal weights and bias values at which it fires [28, 26]. The general model takes the form:

$$\hat{y} = \Phi\left(\sum_{i=1}^{n} w_i x_i + \Theta\right)$$

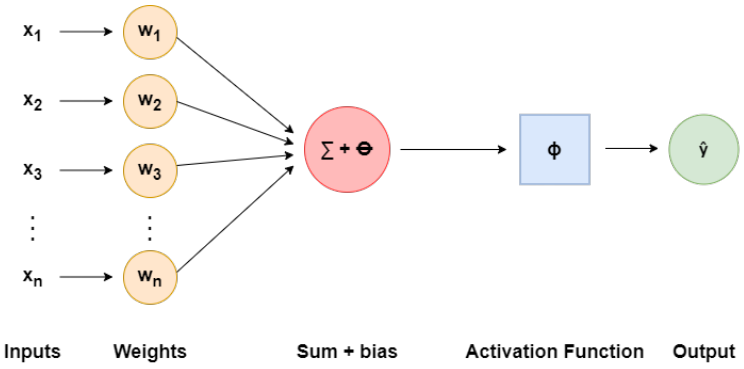


Figure 12: Illustrative simple perceptron

2.4.2 The Multilayer Perceptron

A single perceptron can solve any classification problem for linearly separable classes. If given two nonlinearly separable classes, a single perceptron will fail to solve the problem of classifying them. To solve this type of problem a *multilayer perceptron* (MLP) network is needed. The decision boundaries in a multilayer perceptron network have a more complex geometric shape in the feature space than in a hyperplane [27].

A multilayer perceptron is a feedforward artificial neural network [32]. It generalizes the basic concept of a perceptron to more complex architectures of nodes capable of learning nonlinear decision boundaries. In this architecture, nodes are arranged in groups called layers. These layers are usually organized in the form of a chain so that each layer acts on the outputs of the previous layer. In this way, the layers represent different levels of abstraction that are sequentially applied to the input features [28]. Figure 13 shows a multilayer neural network architecture with a single hidden layer.

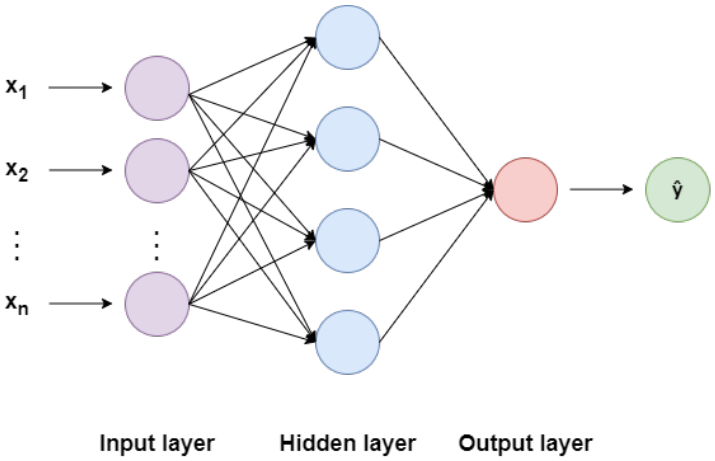


Figure 13: Illustrative example of an artifical neural network with a single hidden layers

There are three types of layers: Input layer, hidden layer, and output layer. The input layer, the first layer of the network, is used to represent attributes from the data. These inputs are fed into the intermediate layers - the hidden layers - which consist of processing units known as hidden nodes. Each hidden node processes the signals it receives from the input nodes or hidden nodes of the previous layer and generates an activation value that is passed on to the next layer. A unit in one layer is connected to all units in the previous and subsequent layers, so they are called fully connected layers. Intuitively, we can think of each hidden node as a perceptron trying to construct a hyperplane, while the output node simply combines the results of the perceptrons to obtain the decision boundary. While the first hidden layer works directly with the input attributes and thus captures simpler features, subsequent hidden layers can combine them and construct more complex features. The use of hidden layers in ANN is based on the general assumption that complex high-level features can be constructed by combining simpler low-level features. The larger the number of hidden layers, the deeper the hierarchy of features learned by the network tends to be. This motivates learning ANN models with long chains of hidden layers known as deep neural networks. Unlike shallow neural networks, which have only a small number of hidden layers, deep neural networks can represent features at multiple levels of abstraction and often require many fewer nodes per layer to achieve similar generalization performance as shallow networks. From this perspective, an MLP learns a hierarchy of features at different levels of abstraction that are eventually combined in the output layer, which processes the activation values from the previous layer to make predictions for the output variables [10, 33, 29].

2.4.3 Convolutional Neural Networks

Convolutional neural networks (CNNs), also called convnets, are a specific type of artificial neural network. They are one of the best learning algorithms for understanding image content and have shown exemplary performance in image segmentation, classification, detection, and retrieval-related tasks [34]. The groundwork for convolutional neural networks goes back to the 1980's and was laid by Fukushima and LeCun et al., which will be detailed in chapter 2.5. However, it only became popular in 2012, when a CNN model called *AlexNet* [2] won the annual ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) [35].

The attractive features of CNN is its ability to exploit spatial or temporal correlations in data and significantly reduce the number of learn-able variables, reducing time and computational cost. For example, in image classification, if one initial layer recognizes edges, a follow up layer can recognize simpler shapes, and the next recognize higher-level features such as faces. The network extracts different abstract features as the input spreads into deeper layers [36].

A typical CNN architecture is generally divided into several learning stages consisting of a combination of

convolutional layers, non-linear processing units, and pooling layers, followed by one or more fully connected layers at the end (Figure 14). In the following, we will explain each stage.

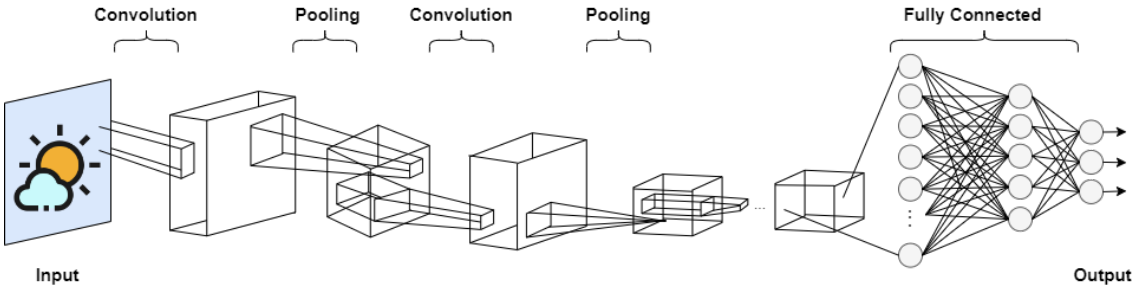


Figure 14: Illustrative example of a Convolutional Neural Network

2.4.3.1 Convolutional Layer

The input image is a matrix of numbers, where each number corresponds to the intensity of a single pixel, ranging from 0 to 255. In the RGB model, the color image consists of three matrices corresponding to the three color channels: red, green, and blue.

These matrices pass through the **convolutional layer**, which is the most important building block in convolutional neural networks. This layer performs an operation called *convolution*, which is a mathematical linear operation between matrices. Since the technique was designed for two-dimensional inputs, multiplication is performed between an array of input data and a two-dimensional array of weights called a *filter* or *kernel*. These kernels are a grid of discrete numbers and typically have a small spatial dimensionality, but are distributed over the entire depth of the input data [37, 10]. Such reduction of dimensions provides moderate invariance in the scale and position of objects. When using images, the inputs have very high dimensions and must be efficiently processed by large CNN models. Therefore, instead of defining convolutional filters that match the spatial size of the inputs, we typically define them to be significantly smaller compared to the input images. This design offers two key advantages: The number of parameters that can be learned is greatly reduced when smaller kernels are used, and small filters ensure that different patterns are learned from local regions corresponding, for example, to different object parts in an image. The size of the filter, height and width, which defines the spatial extent of a region that a filter can change at each convolution step, is called the filter's *receptive field* [10, 38].

A dot product is applied between the filter and the filtered region of the input. A dot product is the sum of the element-wise products of these two matrices, resulting in a single value. Specifically, the filter is applied systematically to each overlapping filtered region of the input, left to right, top to bottom [38].

Repeated application of the same filter to the input image results in a map of activations called a *feature*

map, which indicates the position and strength of a recognised feature in the input. The full feature maps are obtained by using several different kernels. Such a weight-sharing mechanism has several advantages such as it can reduce the model complexity and make the network easier to train.

Each kernel has a corresponding activation map that is stacked along the depth dimension to form the entire output volume of the convolutional layer. This systematic application of the same filter across an entire image is a powerful idea. If the filter is designed to detect a particular type of feature in the input, then the systematic application of that filter to the entire input image provides the filter with the ability to detect that feature anywhere in the image. The innovation of using the convolution operation in a neural network is that the values of the filter are weights that are learned during the training of the network. The network learns which types of features to extract from the input [38, 39].

Mathematically, the convolution operation is denoted with an asterisk and consequently, the feature map values are calculated with the following formula [39]:

$$S(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i-m, j-n) K(m,n)$$

where I denotes the two-dimensional input image, K denotes the kernel, and i and j are respectively the indexes of rows and columns of the resulted matrix.

Figure 15 illustrates an example of a convolution operation, where the input image and kernel only have one channel.

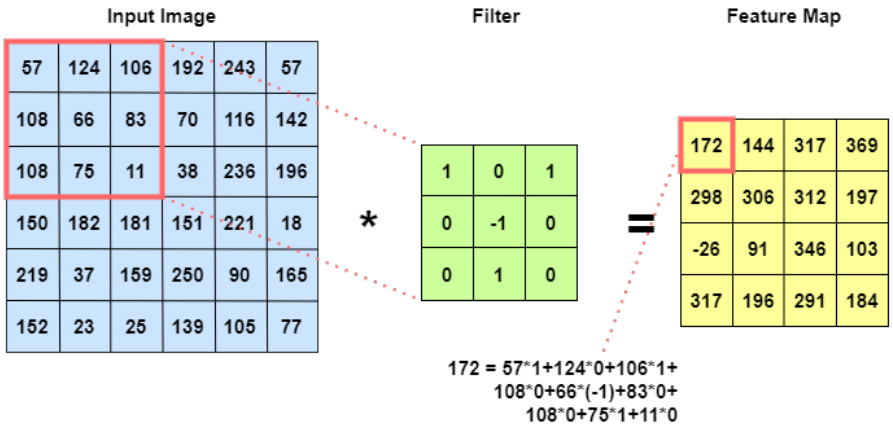


Figure 15: Example of a 2D convolution operation

There have been many advancements in the convolution operation originating variants such as *Tiled convo-*

lution, *Transposed convolution*, *Dilated convolution*, among others which are described in [40].

In the example (Figure 15), the filter moves across the image with a change of one pixel in the column for the horizontal movements and a change of one pixel in the row for the vertical movements. The amount of movement between applications of the filter to the input image is called the *stride* and is almost always symmetrical in height and width dimensions. This value is an integer, usually one, and can be changed, affecting how the filter is applied to the image and the resulting feature map [38]. Figure 16 illustrates this when applying a convolution with a $5 \times 5 \times 1$ input image and a $3 \times 3 \times 1$ kernel, but with different strides. In (a) with a stride of 1 the result is a 3×3 feature map and in (b) with a stride of 2 the result is a 2×2 feature map.

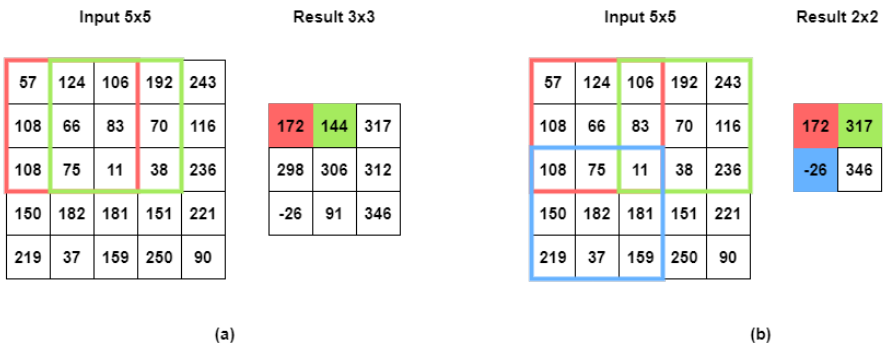


Figure 16: Illustrative examples when using different stride values

One of the disadvantages of the convolution operation is the loss of information that might be present at the edges of the image. Since they are only captured when the filter slides, they have less chance of being seen. A very simple and effective method to solve this problem is the use of *zero-padding*, which is the process of applying a combination of pixels of value zero around the input. It also helps prevent output size from shrinking with depth [36]. For example, in Figure 17 we are using the same input and filter as the example in Figure 15 but with zero-padding. However, the output will be a different feature map with the same size as the original input in Figure 15.

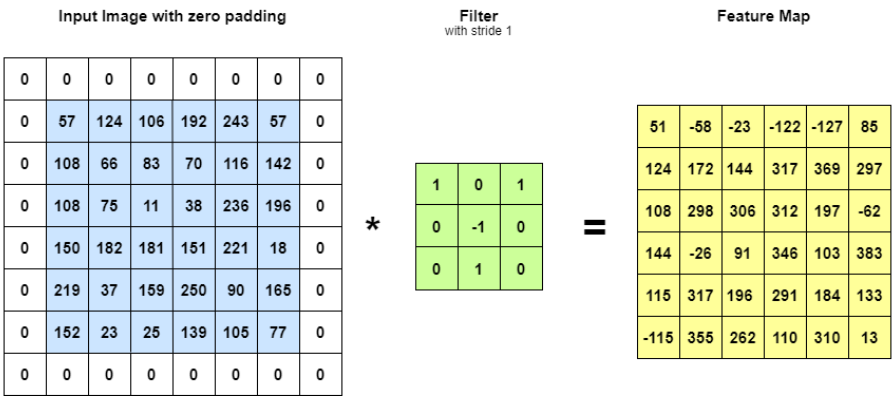


Figure 17: Illustrative example when using zero-padding

The output size of the Convolutional layer can be calculated through the following equation [36]:

$$O = 1 + \frac{I + 2P - K}{S}$$

where O is the output size, I and K are the input and filter size, respectively; S is the stride value and P is the number of zero-padding layers (for example $P = 1$ in Figure 17).

2.4.3.2 Non-linearity

Once the feature map is created, each value in the feature map is passed through a **non-linear activation function**. The non-linearity can be used to adjust or truncate the output generated [36]. The activation function takes a real-valued input and squashes it within a small range. Applying a nonlinear function after the weighting layers is very important because it allows the neural network to learn nonlinear features. A nonlinear function can also be understood as a selection mechanism that decides whether a neuron fires or not given all its inputs. The most common activation functions used in DNNs are *Sigmoid*, *Hyperbolic Tangent* (*Tanh*), and *Rectifier Linear Unit* (*ReLU*) functions [10].

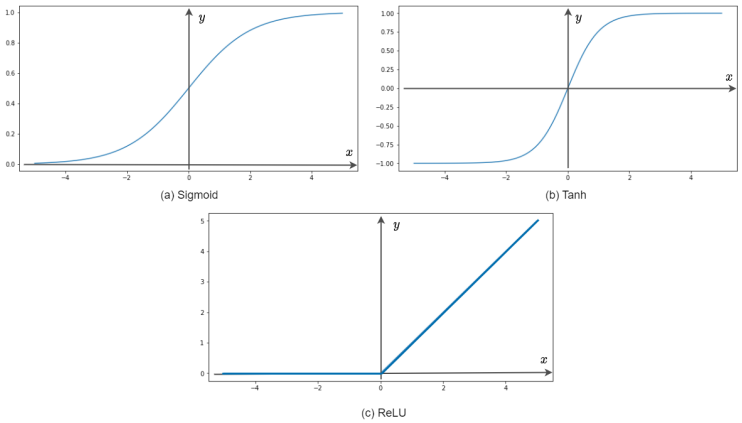


Figure 18: (a) *Sigmoid* (b) *Tanh* (c) *ReLU* activation functions

- **Sigmoid**: The definition of the sigmoid function is as follows:

$$f(x) = \frac{1}{1 + e^{-x}}$$

Here, $Df = \mathbb{R}$ and $D'f =]0, 1[$. Figure 18 (a) shows the graphic of the *Sigmoid* function.

- **Tanh**: The hyperbolic tangent function is defined as follows:

$$Tanh(x) = \frac{sinh(x)}{cosh(x)} = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

Here, $sinh(x)$ is the hyperbolic sine, $cosh(x)$ is the hyperbolic cosine, $D\,Tanh = \mathbb{R}$, and $D'\,Tanh(x) =]-1, 1[$. Figure 18 (b) shows the graphic of the *Tanh* function.

Neural networks using *Tanh* as the activation functions converge faster than those using *Sigmoid*. In addition, the networks using *Tanh* have lower classification errors comparatively to those using *Sigmoid* activation function.[41].

- **ReLU**: The definition of the rectifier linear unit function is:

$$f(x) = max(0, x) = \begin{cases} x, & \text{se} \quad x \geqslant 0, \\ 0, & \text{se} \quad x < 0, \end{cases}$$

Here, $Df = \mathbb{R}$ and $D'f = [0, +\infty[$. Figure 18 (c) shows the graphic of *ReLU* function.

ReLU is computationally cheaper than *Sigmoid* and *Tanh* because it does not need to compute exponential functions, making it much quicker. It converges much faster than the previous activation functions and also allows the network to easily obtain sparse representation, making the network learn more efficiently [41].

The effectiveness of *ReLU* has led to many variants, such as *Noisy ReLU*, *Leaky ReLU*, *Randomized Leaky ReLU* and *Exponential Linear Unit* [10].

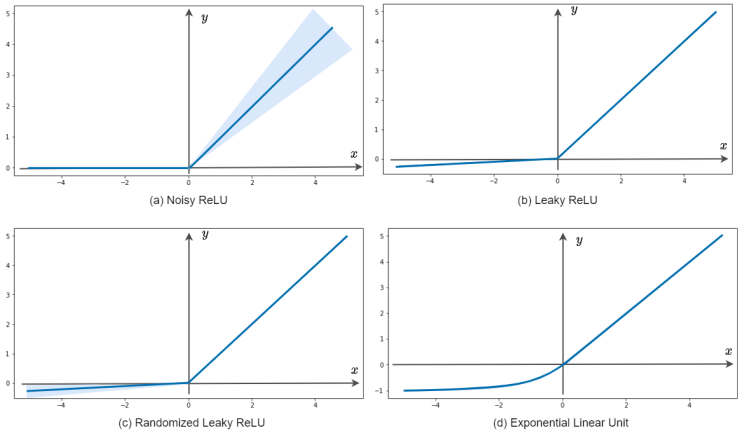


Figure 19: (a) *Noisy ReLU* (b) *Leaky ReLU* (c) *Randomized Leaky ReLU* (d) *Exponential Linear Unit* activation functions

- **Noisy ReLU**: The noisy ReLU adds a sample drawn from a Gaussian distribution with mean zero and a variance that depends on the input value $(\sigma(x))$ in the positive input. The noisy ReLU is defined as follows:

$$f(x) = max(0, x + \epsilon), \quad \text{with } \epsilon \sim \mathcal{N}(0, \sigma(x))$$

  Figure 19 (a) shows the graphic of Noisy ReLU function.

- **Leaky ReLU**: The *LReLU* function is defined by:

$$f(x) = \begin{cases} x, & \text{se} \quad x > 0, \\ cx, & \text{se} \quad x \leqslant 0, \end{cases}$$

  where c is typically a positive small value, such as 0,01. Figure 19 (b) shows the graphic of this function.

  Instead of reducing the output to zero when the input is negative, *LReLU* outputs a down-scaled version of the negative input. Consequently, there are no zero gradients and no neuron units can be "off" always. Experimental results have shown that the learning capabilities of the neural networks become more robust when using *LReLU* [10, 41].

  The **Parametric Linear Units** (*PReLU*) function behaves a lot like the *LReLU*. Its definition is analogous to the previous one with the difference of replacing the constant c with a learnable parameter. Experiments

have shown that *LReLU* can have better results than *PReLU* (and *ReLU*) by choosing the correct c value. However, this process needs repeated training. Oppositely *PReLU* can learn the parameter automatically from the data. In addition, comparatively to *ReLU*, *PReLU* converges faster and has a lower train error [41].

- **Randomized Leaky ReLU**: The definition of *RReLU* is as follows:

$$f(x) = \begin{cases} x, & \text{se} \quad x > 0 \\ ax, & \text{se} \quad x \leqslant 0 \end{cases},$$

Here, in the train phase a is randomly chosen in a given range (l, u), sampled from a uniform distribution $\mathcal{U}$. And in the test phase, it is set to an average value to get the contribution of all samples [10].

$$a \sim \mathcal{U}(l, u) \quad \text{during training;} \quad a = \frac{l+u}{2} \quad \text{during testing.}$$

Figure 19 (c) shows the graphic of *RReLU* function.

- **Exponential Linear Unit**: The *ELU* function can be defined as:

$$f(x) = \begin{cases} x, & \text{se} \quad x > 0, \\ a(e^x - 1), & \text{se} \quad x \leqslant 0, \end{cases}$$

where a is a positive hyperparameter that decides on the saturation level of the function in response to negative inputs.

This function tries to push the mean activations towards zero to decrease the bias shift effect of *ReLU*. This also contributes to faster convergence. Experiments have shown that when using DNNs with more than five layers *ELU* enables faster learning and better generalization, compared to *ReLU* and *LReLU* [10, 41].

Figure 19 (d) shows the graphic of *ELU* function.

There are many other recent activation functions, mostly variants of *ReLU*, such as *SeLU*, *SReLU*, *APLU* and *MeLU* [42].

2.4.3.3 Pooling Layer

After a non-linearity is applied to the feature maps output by a convolutional layer, a **pooling layer** can be employed. The pooling layer passes a filter over each feature map in the input, just like in the convolutional layer. However while in the convolutional operation the kernel has weights, here the filter applies an aggregation

function to the values within the receptive field. Common functions are the *average* and the *maximum* functions, which calculates the average and maximum value for each respective field on the feature map, respectively [37]. The role of the convolutional layer is to detect local conjunctions of features from the previous layer, but the role of the pooling layer is to merge semantically similar features into one [43].

The output of the pooling layer is a down-sampled feature map, this is, a lower-resolution version of the input that contains the more significant features. It is a down-sampling approach that summarises the presence of features in patches of the feature map. This makes the model more robust to changes in the position of the feature in the image, which is technically called local translation invariance [38]. Pooling layers gradually reduce the dimensionality of the representation, further reducing the number of parameters and the computational complexity of the model [37]. It also increases the generalization by reducing over-fitting [34].

The output of the pooling layer applied to a 3 depth input is a 3 depth input of the same depth as the input [13]. Similar to the convolution layer, the size of the pooled region and the stride need to be specified. Figure 20 illustrates an example where a 3×3 filter is used with a stride of 3 on a $5 \times 5 \times 1$ feature map, using max pooling and average pooling.

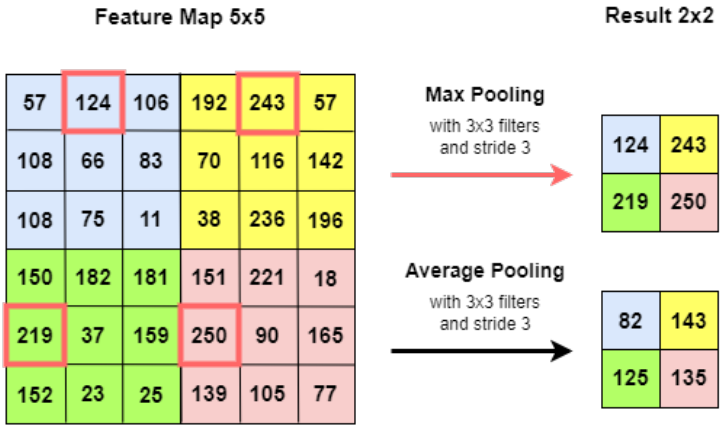


Figure 20: Example of a pooling operation with two different functions

2.4.3.4 Fully Connected Layer

Features are extracted after a combination of convolutional and pooling layers. Depending on the complexity of the images, the number of these layers can be increased to capture even more low-level detail, but at the cost of increased computational power. Next, a *fully connected layer* (FCL) is used. The output of the last layer of the network (convolutional or pooling layer) is of size $n \times n \times m$ and must be flattened to a single one-dimensional vector (size $n^2m \times 1$) to be used as input to the fully connected layer (Figure 21). These layers are identical

to the traditional DNNs (Figure 13), with the addition of an activation function in the output layer. Then the flattened feature map is connected to a series of hidden layers that follow the output layer, which provides the class probability as output for classification [34, 39].

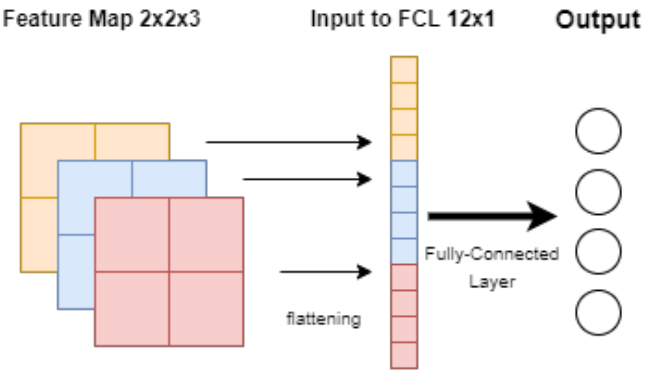


Figure 21: Example of a feature map being flattened to be used as input to an FCL

2.4.3.5 Softmax function

When solving classification tasks we want the output layer to assign a probability to each class. To do so, the **softmax** function is often used as the output of a classifier, to represent the probability distribution over the possible n classes. It is a generalization of the *sigmoid* function which is used to represent a probability distribution over a binary variable. *Softmax* extends it into a multi-class world. *Softmax* assigns probabilities to each class in a multi-class problem in which the total sum is 1 [39]. Mathematically, the *softmax* function is defined as follows:

$$softmax(z)_i = \frac{e(z_i)}{\sum_{j=1}^{n} e(z_i)}$$

where n is the number of classes and z is the vector of raw outputs from the neural network.

2.4.4 Training Neural Networks

This chapter will explain how the learning/training of a neural network works and the hyperparameters involved, which control the learning process.

2.4.4.1 Gradient-Based Learning

For a neural network to be able to classify an image, it needs to be able to recognize and identify patterns in it and understand their meaning. This is achieved through *learning* and happens in the training stage [10].

Initially we feed the network with input data, from the training set, that is grouped into batches. As explained in section 2.4.2, the images are given to the input layer and progress through the hidden layers toward the output layer. This is called **forward propagation**. The output layer contains the estimated numeric values of the forward pass through the model. The learning process adjusts the network weights so that these values are the ones that best represent the model [39].

The main objective of supervised training is to reduce the difference between the predictions (made by the model) and the real output, by discovering the best suitable relationship between the input and the output. To do so we need to have a way to measure how good the output given by the model is to the real value. This measure is computed using a **loss/cost function** described in 2.4.4.2. We want the model to have a small loss value so that the model has minimal losses and maximum precision. In general, for this to happen, an iterative gradient-based optimization method is used which iteratively updates the model weights/parameters in order to minimize the loss function over the training set [10, 44].

Back-propagation (BP), also simply called backprop was proposed by Rumelhart et al. [45]. BP computes the gradient of the loss function by propagating the error value through the network, from the output layer, and then moving towards the input layer. This error is then used to calculate the gradient of the loss function with respect to each weight. The gradient measures the local gradient of the loss function with regard to the vector of weights. And it is used to calculate the steepest descent direction [44, 46].

The term backpropagation is sometimes misunderstood as meaning the whole learning algorithm for multi-layer neural networks. However, back-propagation refers only to the method for computing the gradient, while the optimizer is used to perform learning using the gradient to compute the search direction [39].

Gradient Descent (or **Batch Gradient Descent** (BGD)) is an optimization algorithm that is used to find the weights that minimize the loss function. The optimization of CNNs is a hard task, exacerbated by the fact that these models are composed of a large number of tunable parameters. Therefore, instead of solving for a globally optimal solution, we iteratively search for the locally optimal solution at each step. We use BGD to update the parameters in the direction of the steepest descent. An important parameter in BGD is the size of the steps, given by the *learning rate* (lr) parameter.

We observe that, if the loss function is a continuous convex function, BGD is guaranteed to approach arbitrarily close the global minimum (if we wait long enough and if the learning rate is not too high).

When using BGD method, both loss function and gradient vector are calculated on the entire training set. Each iteration that updates the parameters using the complete training set is called an *epoch* [10, 46].

However, in computer vision problems, the training sets are usually very large and the use of BGD can be

very slow because the gradient on the entire training set must be computed for each parameter. This problem is solved by using *stochastic gradient descent* [10].

Following will be described another two different optimizers: *Stochastic Gradient Descent*, and *Adaptive Moment Estimation*.

Stochastic Gradient Descent (SGD). When using a SGD, both the loss function and its gradient vector are defined, in each iteration, by a single pair chosen randomly from the training set. Thus, SGD performs parameter update for each pair randomly selected from the training set, which makes it converge much faster than BGD. It is also capable of online learning, meaning that parameters can be adjusted in the presence of new training examples. However, the convergence behaviour is usually unstable, especially at relatively high learning rates and when the training data sets contain diverse examples [10].

In practice, instead of using a single pair/element it is used a subset of pairs selected randomly from the training set - called a **mini-batch SGD**. When it is used more stochastic gradients per iteration, it allows the method to obtain better search directions and be easier to fine-tune in terms of learning rate.

Adaptive Moment Estimation (ADAM) is an extension of SGD and has become one of the most popular optimization algorithms for DL. The name derives from adaptive moment estimation, as it is an adaptive learning rate optimization algorithm, meaning it estimates a separate learning rate for each parameter. The method also combines the advantages of two other extensions of SGD, namely Adaptive Gradient Algorithm (AdaGrad) [47] and Root Mean Square Propagation (RMSProp) [48]. AdaGrad maintains one learning rate per parameter, which improves performance on sparse gradients, and RMSProp handles non-stationary problems well. In ADAM, updates are estimated using the first and second moments of the gradient. In practise, ADAM usually scales very well to large problems and has good convergence properties. For this reason, ADAM is often the default choice for many computer vision applications based on Deep Learning [10, 49].

mini-batch SGD and *ADAM* are the two optimizers used in this thesis. There are many others, such as *Momentum*, *Nesterov*, *AdaGrad*, and *RMSProp* [10, 50].

Note that when the networks are deep the learning process can suffer from *vanishing* or *exploding gradient* problems depending on the choice of the activation function. The *vanishing gradient* problem happens in the following way: As the backpropagation algorithm flows from the output layer towards the input layer, the gradient values often get smaller and smaller and approach zero which eventually leaves the weights of the initial layers almost unchanged. Causing the gradient descent to never converge. Contrarily, the gradient values can keep on getting bigger and bigger as the backpropagation algorithm progresses. Causing very large weight updates and consequently diverging the gradient descent. This describes the *exploding gradients problem*. Using *ReLU*

activation functions can avoid both these problems [51, 10].

2.4.4.2 CNN Loss Functions

During the training process, the final layer in a CNN uses a **loss function**, to estimate the quality of the predictions made by the model on the training data, for which the actual labels are known. This function quantifies the difference between the estimated output of the model - *the prediction*, and the correct output - *the ground truth*, and is optimized during the learning process of the model [10].

The type of loss function used in the CNN model depends on the problem at hand. Next, we will describe three loss functions [10, 52]:

- **Categorical Cross-Entropy Loss**, also designated *softmax loss*, decreases as the predicted probability converges to the actual label. It requires that the output layer is set up with the same number of nodes as classes. Using this loss, we will train a CNN to output a probability over the n classes for each image. It is the most common loss function used for multi-class classification and is defined as follows:

$$L = -\sum_{i=1}^{n} y_i log(\hat{y}_i)$$

  where n is the number of classes, y is the desired output and $\hat{y}$ is the softmax probability for each class.

  The *sparse categorical cross-entropy* is sometimes used and it is identical to the categorical cross-entropy loss. The only difference between the two is how the ground truths are defined. Suppose we have a 3 class classification problem. *Categorical cross-entropy* is used when true labels are one-hot encoded, for example, $[1, 0, 0]$, $[0, 1, 0]$, and $[0, 0, 1]$. *Sparse categorical cross-entropy* is used if the ground truths are integer encoded, for example, $[1]$, $[2]$, and $[3]$ [53].

- **Binary Cross-Entropy Loss** is used for binary classification problems with $n = 2$ in the previous formula. Unlike the previous loss, it uses the sigmoid function. It measures the performance of the model whose predicted output is a probability value between 0 and 1, and can be represented as:

$$L = -\sum_{i=1}^{n=2} y_i log(\hat{y}_i) = -\frac{1}{2}\sum_{i=1}^{n=2}(y_i log(\hat{y}_i) + (1 - y_i) log(1 - \hat{y}_i))$$

  where $n = 2$ refers to the two classes, y is the desired output and $\hat{y}$ is the sigmoid probability score.

- **Contrastive Loss** is used to map similar inputs to close points in the output space and to map dissimilar inputs to distant points. This loss function works on the pairs of either similar or dissimilar inputs and can

be defined as:

$$L = \frac{1}{2n} \sum_{n} yd^2 + (1-y)max(0, m-d)^2$$

where m is the margin and $y \in [0, 1]$ indicate whether the pairs are dissimilar or similar respectively. d is a distance measure, for example, the Euclidean distance.

This loss function will be explained in more detail in chapter 3.

The loss functions used for multi-class classification tasks are also applicable to binary classification tasks. However, the case reverse is not true unless a multi-class problem is divided into various *one-vs-rest* binary classification problems where independent classifiers are trained for each case using a binary classification loss [10].

There are many other loss functions such as *SVM Hinge Loss*, l^1 *error* and *Triplet loss*, which are explained in detail in [10] and [40].

2.4.4.3 Weight Initialization

Initializing neural networks plays an important role to stably train deep neural networks. Proper initialization of the weights is critical to its convergence. The weights are updated during the training of the new model, making the pre-trained model act as a weight initialization scheme when training the new model. This helps the new model converge faster since this initialization gives it a starting point at a suitable region that would otherwise be inaccessible by random initialization. The models learned by pre-training are more consistent and provide better generalization performance. [10, 38, 28].

2.4.4.4 Regularizations

As explained previously we want our neural network to perform well on the training data, but also on new unseen data. DNNs have a large number of parameters and tend to over-fit on the training set while learning. There are many strategies used that are explicitly designed to reduce the test error and prevent over fitting. These strategies are designated **regularization** approaches. There are many forms of regularization and developing new effective strategies has been one of the major research efforts in the field [39, 10]. Next, we will describe some regularization techniques.

Dropout

Dropout is a regularization technique to avoid overfitting and thus increase generalization performance. The main goal of dropout is to avoid learning spurious features at hidden nodes. In NNs, multiple connections that learn a nonlinear relation are sometimes co-adapted, so they show good training performance only when used

in highly selective combinations. Dropout randomly drops input and hidden nodes in the network during training to disrupt complex "co-adaptations" in the learned features (Figure 22). Dropout can prevent the network from becoming overly dependent on any one neuron and can force the network to be accurate even in the absence of certain information [28, 40].

Dropout can be added to the model by adding new dropout layers, specifying the amount of nodes removed as a parameter. For example, if the dropout rate is 0.2, 20% of the neurons are randomly selected and ignored at each iteration of forward propagation. We generally use a small dropout value between 20% - 50% of neurons. A probability too low has little effect and a value too high makes the network not learn well enough [32].

Several methods have been proposed to improve dropout. Some are mentioned in [10] and [40].

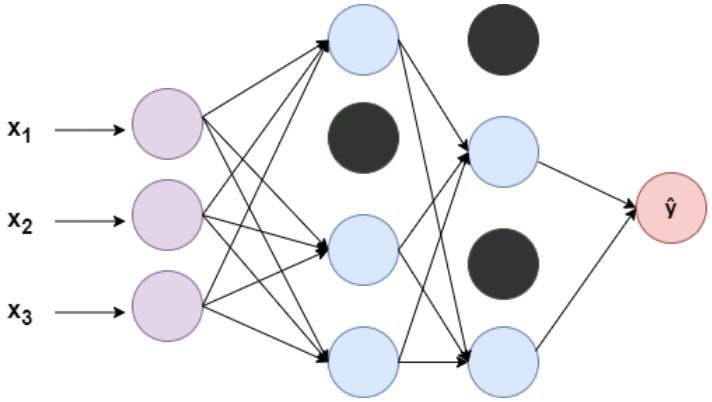


Figure 22: Illustrative example of dropout applied in the hidden layers where the darker nodes represent the "dropped" nodes in an iteration

l^p regularization

This regularization changes the loss function by adding additional terms that penalize the model complexity. Suppose the loss function is L, then the regularized loss will be:

$$E = L + \alpha R(\theta)$$

where $R(\theta)$ is the regularization term and α is the regularization strength [40], being θ the vector of model parameters. Normally α is a very small value.

When $p = 1$, $R(\theta) = \|\theta\|_1$ and the l^1 regularization is equal to the sum of absolute value of the magnitude of the coefficients. When $p = 2$, $R(\theta) = \frac{1}{2}\|\theta\|_2^2$ and the l^2 regularization is commonly referred to as *weight decay* and is the sum of the square of all weights [10, 40].

Data Augmentation

Data Augmentation is an easy and effective way of enhancing the generalization power of CNN models and also enlarging datasets that have a small number of training examples. Data augmentation consists in transforming the available data into new data without altering their natures. Some methods are rotation, cropping, flipping, scaling, and colour distortion. These operations can be performed separately or combined [10, 38].

Figure 24 illustrates some random augmentation techniques applied to Figure 23, such as zoom, rotation, shift, colour distortion and flip.

Figure 23: Original image to apply augmentation

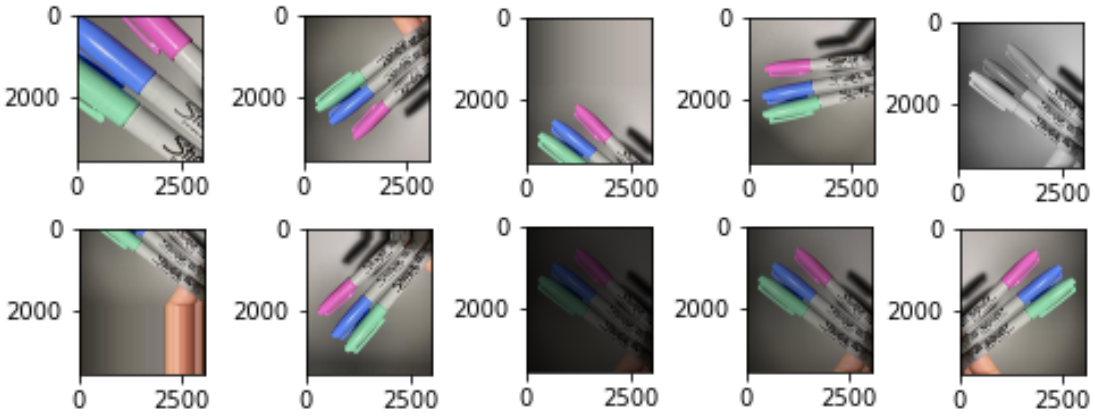


Figure 24: Random augmented images of 23:zoom, rotation, shift, colour jittering, and flip

Batch Normalization

Batch Normalization (BN) is used to solve the problems related to internal covariance shifting within feature maps. Internal covariance shift refers to the change in the distribution of activations of each layer as parameters

are updated during training. BN solves this problem by using a normalization step that fixes the means and variances of the layer inputs, computing the estimates of mean and variance after each mini-batch rather than after the entire training set. This improves convergence and avoids network instability issues such as vanishing/-exploding gradients and activation saturation [34, 10, 40].

During training, in order to zero-center and normalise the inputs, the BN layer calculates the mean μ and variance σ^2 over the current mini-bacth, with m the number of instances in the mini-batch:

$$\mu = \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \sigma^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu)^2$$

Then normalizes the inputs

$$\hat{x}_i = \frac{x_i - \mu}{\sqrt{\sigma^2 + \epsilon}}$$

and scales and shifts the normalized values in order to obtain z_i

$$z_i = \gamma \hat{x}_i + \beta$$

where γ and β are parameters learned during training, ϵ is a constant added to the mini-batch variance for numerical stability [54].

BN is usually applied after the CNNs convolution layers, before applying the nonlinear activation function, and is used in state-of-the-art CNN architectures [10].

BN has many advantages. It stabilizes the training of deep networks and provides robustness to bad weight initializations. In addition, when used the training of the network is less sensitive to the choice of hyperparameters, such as the learning rate, and it greatly improves the convergence rate of the network. Finally, BN regularizes the model, and thus reduces the need for Dropout [10, 40].

Early Stopping

Early stopping is a strategy applied to avoid overfitting. This is achieved by returning to the parameter setting at the point in time where the metric being monitored is the lowest, which is normally the validation set loss (Figure 25). Every time the error on the validation set improves, a copy of the model parameters is stored. When the training is done, we return those parameters, instead of the latest parameters. The algorithm terminates when no parameters have improved over the best-recorded validation loss for a pre-specified number of epochs. This strategy is simple and effective and therefore one of the most used forms of regularization [39].

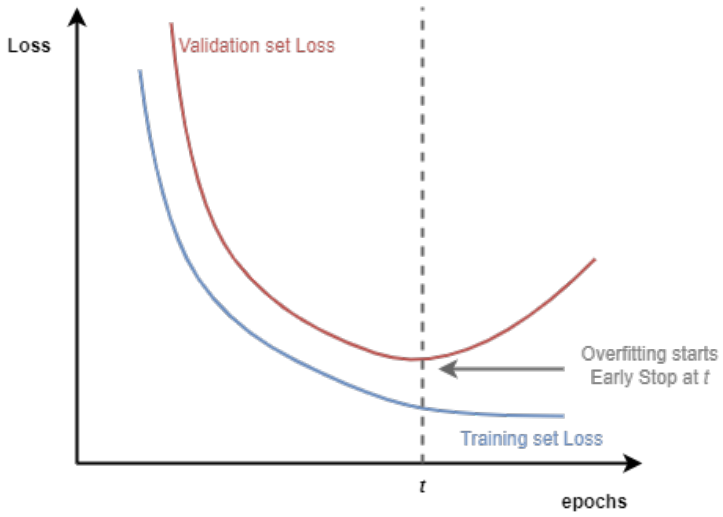


Figure 25: An illustration of a network overfitting during training and applying early stopping

2.4.5 Convolutional Neural Architectures

A CNN is a combination of the layers described previously, arranged in a specific way. The number and the sequence of these layers depend on the way it is dimensioned. Designing a CNN consists on defining the sequence and the structure of the layers: defining the number of filters that each convolutional layer will process, as well as the size of the filters and stride; choosing the activation function to use; defining the pooling operation, along with the number of filters, size, and stride. Figure 14 schematises a CNN architecture.

Following, we will introduce some successful CNN designs which are constructed using the basic building blocks that were explained so far.

2.4.5.1 AlexNet

AlexNet was proposed by Krizhevesky et al. [2] and won the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) in 2012. It achieved a large improvement in image classification performance compared to previous CNN architectures such as LeNet, which were smaller and not tested on large datasets such as the ImageNet dataset [35].

The architecture of the network is summarised in Figure 26. It contains five convolutional layers and three fully-connected layers (FCLs). *ReLU* is applied after each convolutional layer. The output of the last FCL layer is fed into a 1000-way *softmax*, which produces a distribution over the 1000 ImageNet class labels. Data augmentation, dropout, and normalization were also applied to avoid overfitting. *AlexNet* has 62.4 million parameters trained on

ImageNet with 1.2 million images. The deep CNN's learning capability was limited at this time due to hardware limitations. To overcome this, two GPUs were used in parallel to train AlexNet [10, 55, 2].

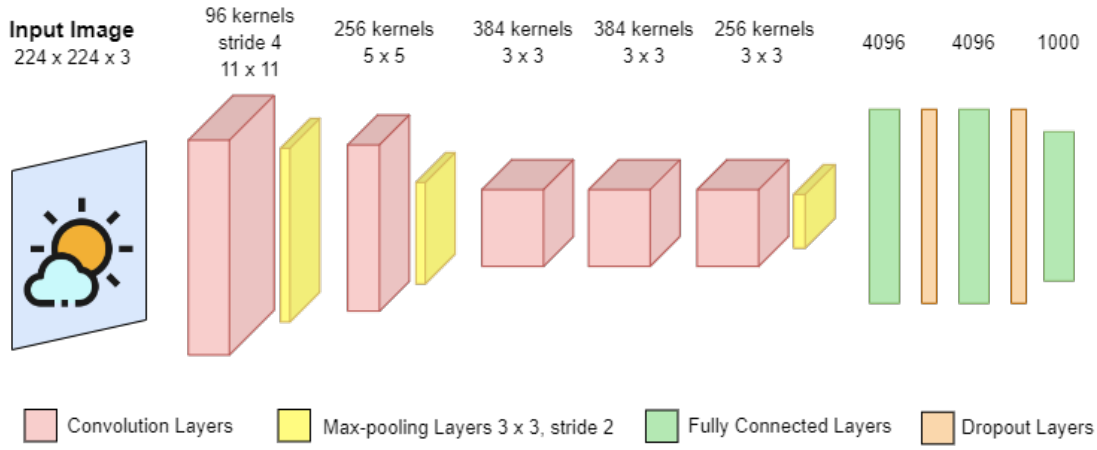


Figure 26: An illustration of the AlexNet Architecture

2.4.5.2 VGGnet

The VGGnet [56] was introduced in 2014 by Simonyan and Zisserman from the Visual Geometry Group (VGG) research lab at Oxford University. Although it did not win the ILSVRC'14, it became one of the most popular CNN models, due to its simplicity and the use of small-sized convolutional filters. There are many different configurations of this network, but the most successful are *VGGnet-16* and *VGGnet-19* [10].

The VGGnet architecture uses only 3x3 convolutional kernels with max-pooling layers and three full-connected layers. Each convolutional layer is followed by a *ReLU* layer. Padding was performed to maintain spatial resolution, and dropouts are used in the first two FCLs to avoid overfitting. The use of smaller filters leads to a relatively small number of parameters and thus efficient training and testing. In addition, smaller filters allow more layers to be stacked, leading to deeper networks and thus better performance in vision tasks. VGGnet set a new trend to use small filters in CNNs [10, 34, 56].

Figure 27 illustrates *VGGnet-19* which has 144 million parameters.

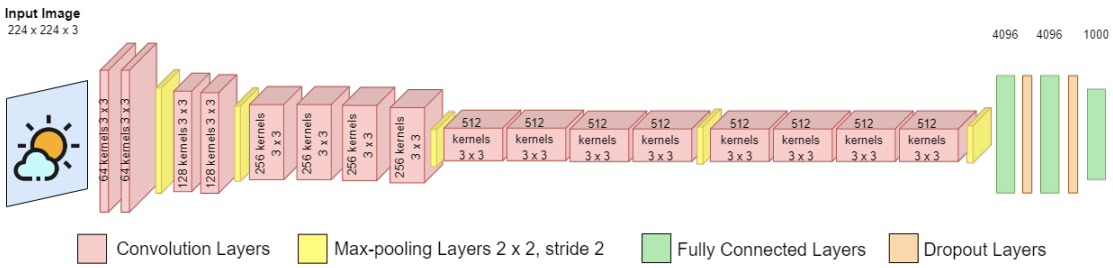


Figure 27: An illustration of the VGGnet-19 Architecture

2.4.5.3 ResNet

He et al. developed ResNet (Residual Network) [57], which won the 2015 ILSVRC competition. Their goal was to design an ultra-deep network that was free from the vanishing gradient problem, as compared to previous networks [55].

The concept behind ResNet was that despite its depth, the network is trained similarly to a shallow network by skipping after every 2 layers [58]. To perform a computation, both the input and the output were copied to the next layer, basically learning the residual of the previous computation. The details of the skip connection are shown in Figure 28. Given an input x, the convolutional layers implement a transformation function on that input, denoted $F(x)$. In a residual block, the original input is added to this transformation, using a direct connection from the input that bypasses the transformation layers. The original mapping thus becomes $x + F(x)$ (Figure 28). This connection is called a *skip connection*. In this way, the transformation function in a residual block is split into an identity term (representing the input) and a residual term, which helps to focus on the transformation of the residual feature maps. ResNet consists of several residual blocks stacked on top of each other. The residual connections are the key to better classification accuracy of deep networks [57, 10, 55].

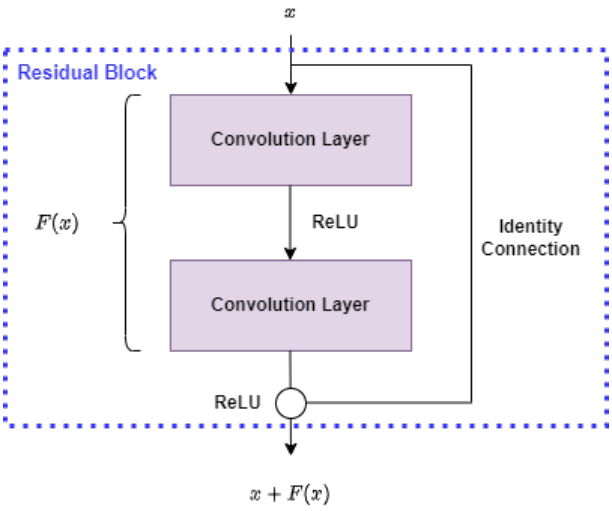


Figure 28: Residual Block in a ResNet

The convolutional layers in the residual block are followed by a BN and a *ReLU* activation layer. Different types of *ResNet* have been developed based on the number of layers (starting from 34 layers up to 1202 layers). The most widely used type was *ResNet50* (Figure 29), which has 49 convolutional layers plus a single FC layer [57].

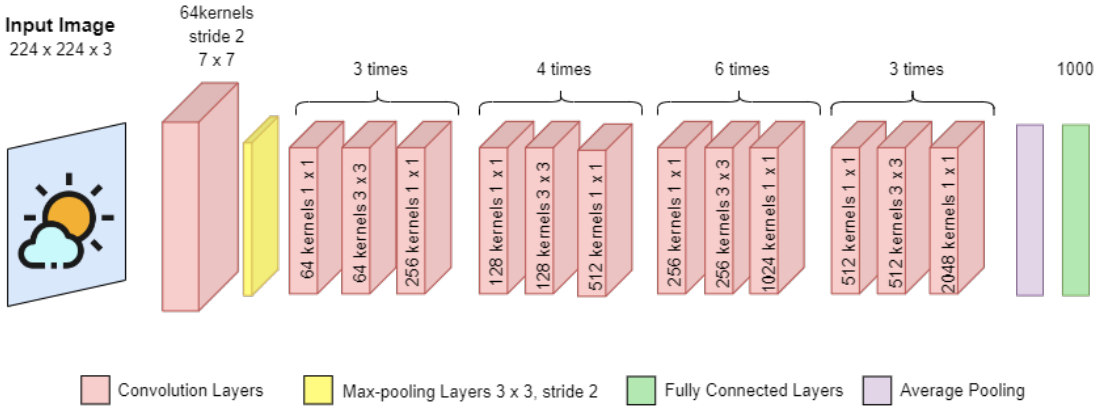


Figure 29: An illustration of the ResNet50

2.5 Computer Vision

Vision is the sense on which humans rely on most, and undoubtedly the one that provides most of the data he receives. The amount of information the higher centers of the brain receive from the eye must be at least two orders of magnitude greater than all the information they receive from the other senses. Of course, humans do

not store all this information, some is forgotten over time and some goes unnoticed. It is impossible to retain all the data received when data rates for continuous viewing are likely to exceed 10 Mbps [59].

In a world where we want to have machines do tasks just like humans, machines need a sense of vision. We are inundated with images. It has never been easier to take and share a photo or video. Youtube is one of the largests search engines and hundreds of hours of videos are uploaded every minute and billions of videos are watched every day.The internet is comprised of text and images. It is relatively easy to index and search text, but to index and search images, algorithms need to know what the images contain. For a long time, the content of images and videos remained opaque and was best described by the meta descriptions of the person who uploaded them. To get the most out of image data, we need computers that can "see" an image and understand the content [38].

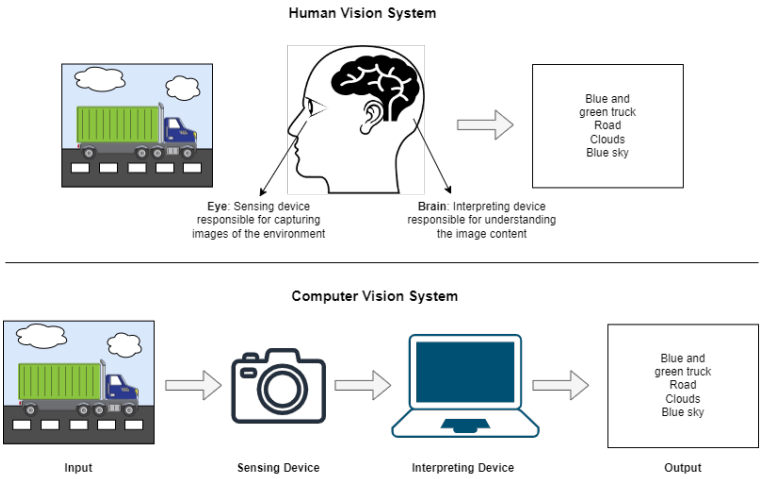


Figure 30: Human vision system and computer vision system

Computer vision (CV) is a field concerned with developing techniques that help computers and systems see and understand the content of digital images, derive meaningful information from them - and take action or make recommendations based on that information. If AI enables computers to think, then computer vision enables them to see, perceive and understand their environment [60]. Typically, this involves developing methods that attempt to replicate the capabilities of human vision.This seems like a simple problem because it is so effortless for humans, even at a very young age. Yet it remains a largely unsolved problem based on the limited perception of human vision and the complexity of the visual world [38].

CV works much like human vision, except that humans have a head start. Human vision has the advantage of a lifetime of training to distinguish objects, how far away they are, whether they are moving, and whether there is something wrong in an image [60]. A true vision system must be able to see in any scene and still extract

something meaningful. Computers work well on tightly constrained problems, not on open-ended, unbounded problems like visual perception. Still, there has been progress in this area, especially in recent years with available optical character recognition and face recognition systems in cameras and smartphones. Computer vision is at an extraordinary point in its evolution. The subject itself has been around since the 1960s, but only recently has it been possible to build useful computer systems using ideas from CV [38].

In 1959 David Hubel and Torsten Wiesel, two neurophysiologists, placed electrodes into the primary visual cortex area of an anesthetized cat's brain and tried to observe the neuronal activity in that region while showing the cat various images (Figure 32). Initially, they couldn't get the nerve cells to respond to anything. However, after a few months, they accidentally saw that a neuron fired as they were slipping a new slide into the projector. They after realised that what got the neuron excited was the movement of the line created by the shadow of the sharp edge of the glass slide. The researchers concluded that there are simple and complex neurons and that visual processing starts with simple structures such as straight edges [61].

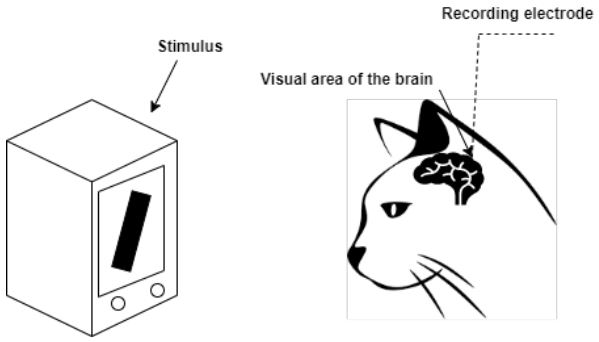


Figure 31: David Hubel and Torsten Wiesel's experiment

Also in 1959 Russell Kirsch and his colleagues developed an apparatus that allowed transforming images into grids of numbers, a binary language machines could understand. It is because of their work that we now can process digital images in diverse ways [62].

In 1963 Lawrence Roberts published his Ph.D. thesis named "Machine perception of three-dimensional solids" [63], where he described the process of deriving 3D information about solid objects from 2D photographs. The objective was to process 2D photographs into a line drawing, transform the line drawing into a 3D representation, and finally, display the 3D structure with all the hidden lines removed, from any point of view. This was considered to be one of the precursors of modern CV.

AI became an academic discipline in the 1960s. In 1966, Seymour Papert a MIT professor launched the Summer Vision Project [64], proposing a group of MIT students to construct a significant part of a visual system

in one summer. The primary goal was to construct a system of programs which would divide a vidisector picture into regions such as objects and background areas - apply segmentation. The project wasn't a success. They realised that the problem was not as simple as they thought. However, the project was considered the official birth of CV as a scientific field.

In 1982 David Marr, a British neuroscientist, published an article [65] where he established that vision is hierarchical. He introduced a vision framework where he used low-level algorithms that detect edges, curves, etc. as an instrument towards a high-level understanding of visual data. This was ground-breaking at the time, but very abstract and high-level. In the same year, Kunihiko Fukushima, a Japonese computer scientist built a self-organising neural network that could recognise patterns unaffected by position shifts - *Neocognitron* [66]. The network contained many convolutional layers with weight vectors. Their function was to slide across 2D arrays of input values - image pixels, and, after doing specific calculations, it would produce activation events that were used as inputs for the next layer of the network. Neocognitron is a grandfather of convolutional networks.

In 1989 Yann LeCun, a French scientist, and his colleagues applied a backpropogation style learning algorithm to Fukushima's Neocognitron [67]. After working on this for some years in 1998 he released the first modern convolutional network - *LeNet-5* [68], that introduced some of the essential ingredients used in today's convolutional neural networks.

Researchers continued to investigate the field, some succeeding past work and others following different paths. In 1999 David Lowe, a Canadian computer scientist, turned towards feature-based object recognition. He created a visual recognition system that uses local features invariant to image scaling, translation, rotation, and partially to illumination changes. He believed that these features were similar to the proprieties of neurons found in the inferior temporal cortex that are involved in the object detection process in primate vision [69].

In 2001 Paul Viola and Michael Jones introduced a face detection framework that worked in real-time [70]. They introduced a method for combining increasingly more complex classifiers in a cascade, which allowed background regions of images to be discarded while spending more computation on object-like regions.

In 2005 Pascal VOC project was launched [71]. It provided a dataset for object detection and object classification with 1578 images, 4 classes and 2209 annotated objects. This dataset has grown over time and currently has 20 classes, 11530 images in the training set and 27450 annotated objects.

In 2010 ImageNet Large Scale Visual Recognition Competition (ILSVRC) was launched [35] with a dataset also for object detection and classification, with 1K of object classes. The error rate in image classification was around 26%. In 2012 a team from the university of Toronto entered the competition with a CNN named *AlexNet* [2], achieving a error rate of 15.3%. This was a breakthrough moment for convolutional networks. The error rate

has been falling since then and the winners have always been CNNs.

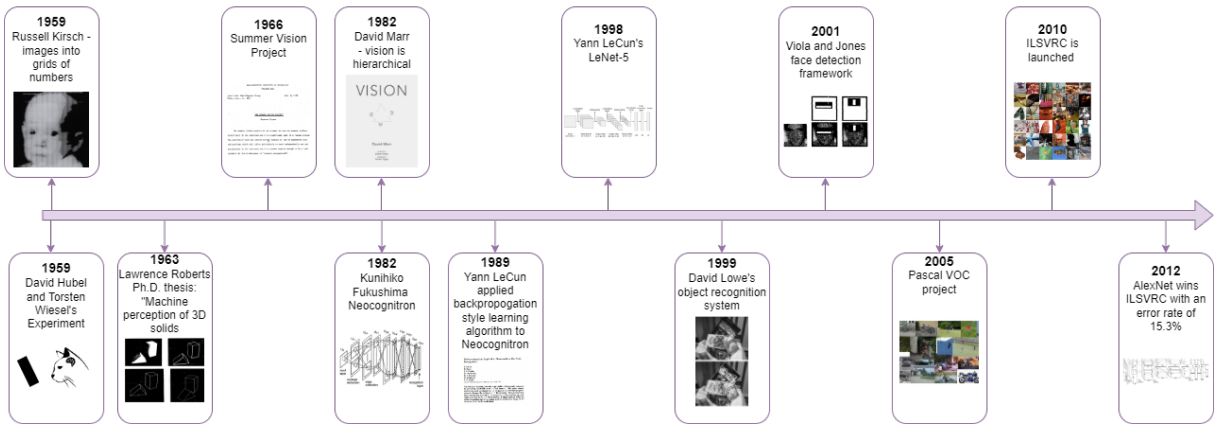


Figure 32: Brief history of computer vision

The most popular CV tasks include: classification, detection and segmentation.

- **Image classification** is the task of processing an entire image as a whole and classifying it into a set of predefined classes.
- **Object detection** is the task to recognise, classify and identify the spatial position of objects in an image.
- **Image segmentation** is the task of dividing an image into sub parts to differentiate objects from the background and/or from other objects in the same image.

CV has had success in a wide variety of real life applications such as: optical character recognition (OCR), machine inspection, retail, 3D model building, medical imaging, self-driving vehicles, motion capture, fingerprint recognition and surveillance [38, 72].

3 Self-Supervised Learning

Advances in Deep Learning have achieved great success in the last decade, especially in supervised learning tasks in computer vision such as image classification [73, 2], semantic segmentation [74, 3, 75], and object detection [1, 76, 77].

SL approaches are heavily dependent on the amount of annotated training data available. The quality of these DNNs is strongly influenced by the number of labeled images. Although there is an abundance of data available, there is a lack of annotation as it is expensive and time consuming. *Imagenet* has over a million images, which allows training networks with impressive performance. However, in many real-world applications, it is not possible to create labeled datasets with millions of images [78]. Moreover, SL suffers from generalization errors and spurious correlations [6, 8]. This has led researchers to look for alternative approaches that can take advantage of the vast amount of freely available data without the need for exhaustive labeling.

A promising alternative is **Self-Supervised Learning** (SSL) , which is attracting a lot of attention due to its tremendous performance in learning representations without the cost of annotating large datasets [6, 7]. In SSL, the model is fed with unlabelled data. During the training phase, the data is automatically labeled by finding and using the relationships between the different input features. This forces the network to learn semantic representations about the data. These learned features can then be applied as pre-trained models to downstream tasks to improve performance and avoid over-fitting [4, 79, 80, 8].

The cognitive motivation behind SSL is the way infants learn, largely through observation. Within 3 to 4 months of birth, infants have meaningful expectations about the world around them, concepts such as gravity and object permanence. Infants' environment becomes a source of supervision that helps them develop a general understanding of how things work. SSL is an attempt to apply this concept to machines, where the data itself contains inherent features that provide supervision for training the model, rather than labels that tell the network what is right and what is not. Some authors believe that SSL is a promising way to build background knowledge and approximate a kind of common sense in AI systems. Also, when humans do new tasks, information from life experience and prior knowledge is used. SSL mimics this when it transfers the learned features to other tasks [8, 81, 82].

SSL, can be considered a branch of unsupervised learning because no manual labeling is required and for that reason, it is sometimes designated *unsupervised representation learning* [83, 84, 4]. However, in a narrower sense, unsupervised learning focuses on detecting specific data patterns, while self-supervised uses information from the input data itself as supervisory signals, which is in line with the supervised settings paradigm [6].

AlexNet, *VGGnet*, *GoogleNet* [85], *DenseNet* [86], and *ResNet* are commonly used architectures in SSL.

SSL methods can be divided in two types: *Pretext tasks* and *Contrastive Learning* [79, 6, 83, 87, 8].

3.1 Pre-text tasks

In the early stages of SSL, pretext tasks were defined as a way to learn representations using labels automatically created based on data attributes. The model learns features as it solves the pretext task, then TL is applied with those features (Figure 33). These pretext tasks must be designed so that a high level of image understanding is useful in solving the task. Therefore, the intermediate layers of CNNs trained to solve these pretext tasks encode high-level semantic visual representations that are useful for solving downstream tasks, such as classification [4, 79, 80, 78]. The choice of the pretext task is important and depends on the type of problem being solved. For example, predicting the rotation of an image as a pretext task may be useful for view-independent aerial image recognition as a downstream task, but probably will not be helpful in detecting which way is up in a photograph for a display application [88].

Examples of pretext tasks include predicting the degree of rotation of an image [89, 90], filling in a missing part of an image [91, 92], coloring a grayscale image [93, 94, 95, 96], improving the resolution of images [97], predicting the relative position of an image [98], solving a jigsaw puzzle [99, 100, 101], ordering a sequence of images [102], and more. Researchers in [103] even investigate methods for combining multiple pretext tasks.

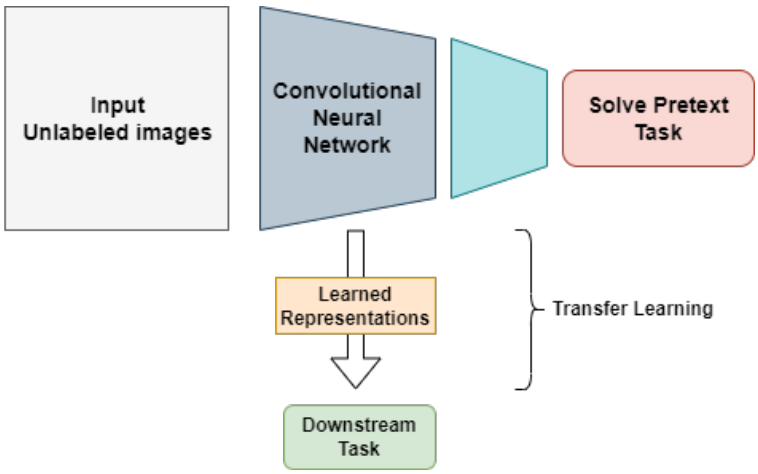


Figure 33: Self-Supervised Learning framework using pretext task

Next, the following pretext tasks will be described: *Colorization*: coloring a gray-scale image, *Jigsaw Puzzle*: solving a jigsaw puzzle, and *Rotation*: predicting the degree rotation of an image.

3.1.1 Colorization

Zhang et al. [93] proposed the coloring of grayscale images as a pretext task. The method uses the CIE Lab color space [104] representation of an input image and trains the model to predict the ab colors from the given input lightness L. The goal is not necessarily to recover the actual ground truth color, but rather to produce a plausible colorization that could potentially deceive a human viewer. An illustration of this task is given in Figure 34.

The authors propose a fully connected network consisting of an encoder for feature extraction and a decoder for color hallucination for colorization. The network is optimized with an $L2$ loss between the predicted and original colors. It is trained with over a million color images by simply using the L channel of the image as input and the ab channels as the supervisory signal.

Color prediction is multimodal. Many objects can take on multiple plausible colorings. For example, an apple is typically red, green, or yellow, but probably not blue or orange. To model this appropriately, the problem is treated as a multinomial classification. The ab output space is quantized into 313 bins and the model predicts a distribution of possible colors for each pixel. The final colorization is generated by taking the annealed mean of the distribution. The final output is a vivid and realistic colorization.

Trained with large collections of images, the method shows great results and fools humans on 32% of the trials during the colorization test.

Once the network is trained on the pretext task, the decoder part of the network is removed, and the rest is used to perform downstream tasks. Experimental results have shown that learning via colorization as a pretext task is effective for solving object recognition, object classification, and segmentation problems. On the *PASCAL VOC 2007* [105] classification and detection tasks this method achieves 65.9% and 46,1% of mean average precision (mAP) , respectively. On the *PASCAL VOC 2012* [106] segmentation task it reaches a 35.0% of mean intersection over union (mIU) .

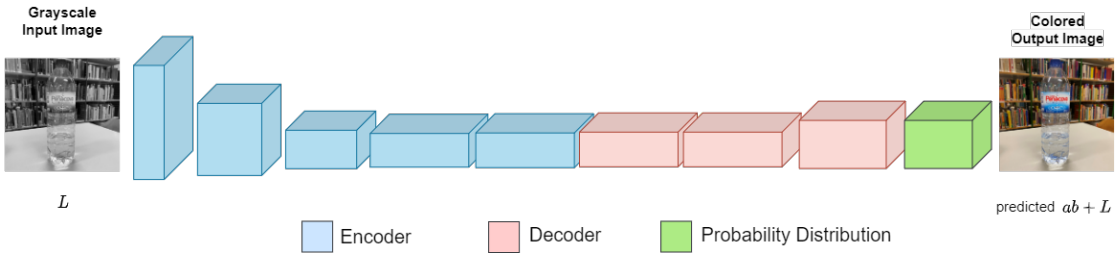


Figure 34: Illustration of colorization as a pretext task

3.1.2 Jigsaw Puzzle

Norozzi and Favaro [99] propose a CNN that can be trained to solve jigsaw puzzles as a pretext task.

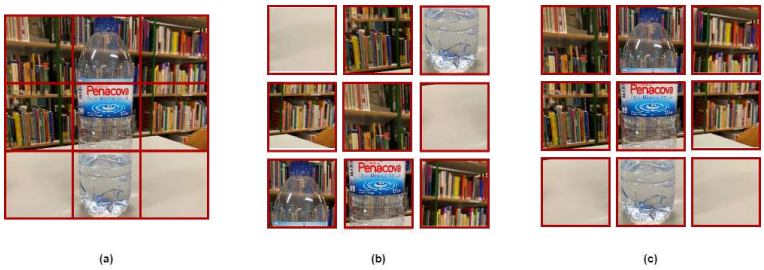


Figure 35: (a)The image split in a 3×3 grid. (b)The puzzle obtained by shuffling the tiles. (c)The puzzle solved after being shuffled.

They crop a random region from the picture, divide this region into a 3×3 grid (Figure 35 (a)) and shuffle the tiles (Figure 35 (b)). The goal is to solve the jigsaw puzzle (Figure 35 (c)). With 9 image patches, there are $9! = 362880$ possible permutations and each of them has an index. The authors randomly choose a permutation (in the example in Figure 36, the permutation is $(9, 8, 4, 2, 1, 5, 6, 3, 7)$) and rearrange the 9 input patches according to this permutation. To limit the number of permutations, the Hamming distance is used to select only a subset of the permutations with the largest Hamming distance. Only the selected permutations are used to train the CNN to recognise the permutation of the mixed image patches.

The image patches are fed into the CNN, which is trained to recognise the correct spatial positions of the input patches by learning spatial context structures of images such as object colour, structure and high-level semantic information. Each row up to the first fully connected layer (FCL6) uses the AlexNet architecture with shared weights. The output of all FCL6 layers, i.e. the patch representations, are concatenated and passed to the FCL7 layer, followed by the FCL8 layer. The output of this last layer is passed to a softmax function that provides a vector with a probability value for each index. The goal is for the CNN to predict the correct index of the corresponding permutation (in the example in Figure 36 the index of the permutation is 27). Once the permutation is known, the puzzle can be solved.

To avoid shortcuts due to edge continuity and pixel intensity distribution, the authors leave a random gap between tiles.

To solve the jigsaw puzzle, the model must learn to recognise how the parts in an object are assembled, their shapes and the relative position of the different parts of objects. Thus, the representations are useful for classification and recognition of difficult tasks. On the *PASCAL VOC 2007* classification and detection tasks this method obtains 67,6% and 53,2% mAP, respectively. On the *PASCAL VOC 2012* segmentation task it achieves a 37,6% mIU, outperforming the colorization pretext task.

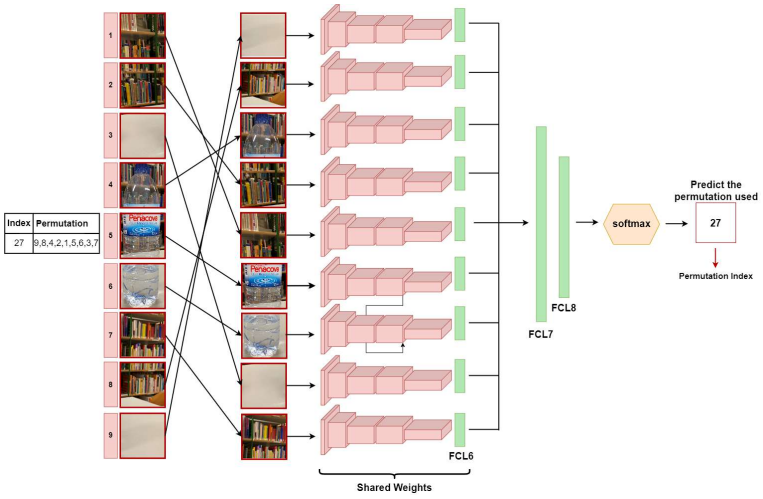


Figure 36: Illustration of how the jigsaw puzzle pretext task is generated and solved with the permutation in Figure 35 (b) $(9, 8, 4, 2, 1, 5, 6, 3, 7)$ and index 27

3.1.3 Rotation

Gidaris et al. [89] proposed a pretext task to learn image representations by training CNNs to recognize the geometric transformation that is applied to an input image. Each input image is rotated with 4 different 2D rotations, which are all fed to a CNN model. This model is trained with cross-entropy on a 4-class classification task to predict the transformation applied.

Formally, X is an input image, and $G = \{g(X|y)\}_{y=1}^{4}$ the set of geometric transformations as all the image rotations by multiples of $90°$, this is, 2D image rotations by $0°$, $90°$, $180°$ and $270°$. $g(X|y)$ is the operator that rotates X by y degrees, which yields the transformed image $X^y = g(X|y)$. The model $F(.)$ receives as input the image X^{y*} (where $y*$ is unknown to the model) and outputs a probability distribution over all possible geometric transformations: $F(X^{y*}|\theta) = \{F^y(X^{y*}|\theta)\}_{y=1}^{4}$. Where $F^y(X^{y*}|\theta)$ is the predicted

probability for the geometric transformation with label y and θ are the learnable parameters of the model (Figure 37).

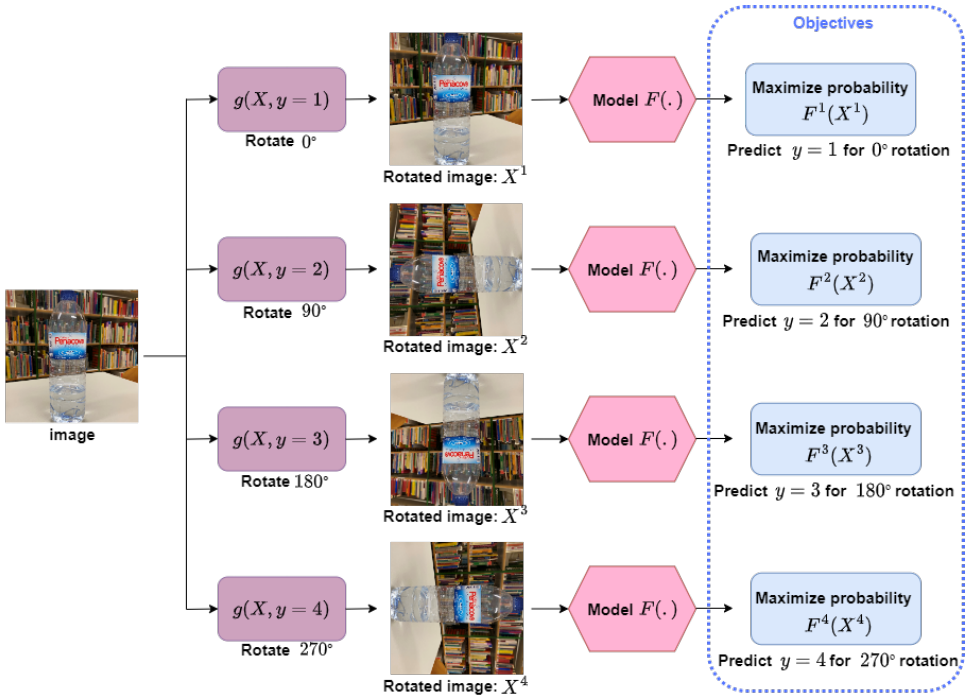


Figure 37: Illustration of the rotation pretext task where g is the geometric transformation (0°, 90°, 180° or 270° rotations). $F^y(X^{y*})$ is the probability of rotation transformation y predicted by $F(.)$ when it receives as input an image that has been transformed $y*$

The authors believe that it is essentially impossible for a CNN model to effectively perform the above rotation recognition task unless it has first learned to recognize and detect objects as well as their semantic parts in images. Despite the simplicity of the task, it provides a very powerful supervisory signal for semantic feature learning.

Fine-tuning the learned features on the *PASCAL VOC 2007* classification and detection tasks this method reaches 72,97% and 54,4% mAP, respectively. On the *PASCAL VOC 2012* segmentation task it reaches a 39.1% mIU, outperforming the previous pretext tasks.

3.2 Contrastive Learning

As mentioned previously, in the early stages of SSL, representation learning focused on exploiting pretext tasks. Though these approaches succeed in computer vision tasks, there is still a large gap between these methods and

supervised learning [93, 99, 89, 79, 8]. Recently, there has been a significant advancement in using **Contrastive Learning** (CL) , which significantly closes the gap between SSL methods and supervised learning [107].

CL is a discriminative model that uses positive and negative pairs to learn representations by distinguishing between views of the same image [79]. It aims at pushing different views of the same instance close together and different instances further apart in the representation space [87, 5, 7, 8].

Figure 38 (a) shows a batch of two images, where each image forms its own class. To create a positive pair, the image is augmented in two different ways. One of the augmented views is called the *anchor* of the image and the other is called a *positive*. Any image that differs from the anchor image is *negative* to the anchor [8]. In Figure 38 (b) the image denoted as x^a is the anchor image of x, x^+ is a positive pair to the anchor image, and x^- a negative, as it is from a different class. The CL objective is to learn a representation space that pulls representations that come from the same images (x^a and x^+) and repel representations that come from different images (x^a and x^-).

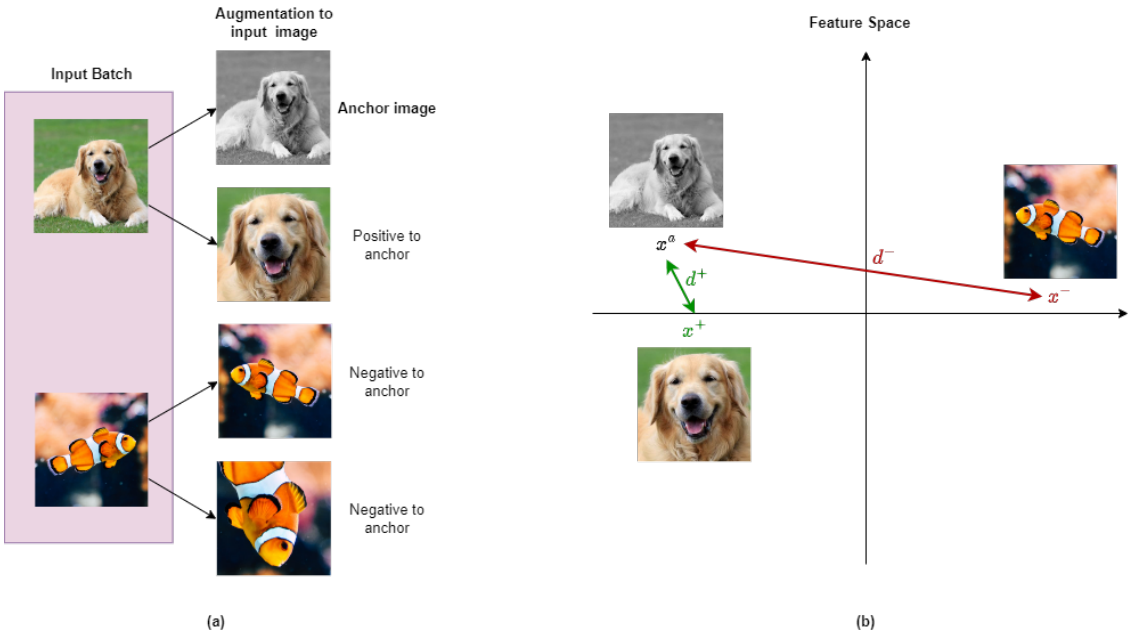


Figure 38: (a) For each image in the batch random augmentation is applied to get a pair of two images that represent different instances of the same image. (b) CL pulls the anchor and positive images close together and the negative image away.

The CL framework can be divided into 5 parts [8, 108, 79]:

1. **Data Augmentation Pipeline:** The purpose of data augmentation in contrastive learning is to generate anchor, positive and negative images. Augmentation is applied to the input images to generate new

samples which preserve the same underlying semantics as the original input images [108, 109]. A good augmentation strategy is an important factor for CL, as it forces the network to learn rich and generalizable features in an SSL environment [8]. Research has shown that combining multiple data augmentation techniques boosts representations [110, 83]. Figure 24, in section 2.4.4.4, illustrates examples of augmentation.

2. **Encoder:** The encoder part of the network extracts the feature representations of the images. Given two augmented images x_i and x_j it extracts embedding vectors h_i and h_j (Figure 39), which are feature representations of x_1^+ and x_2^+ respectively [8, 108]. These representations affect how well a classification model learns to distinguish between different classes. It has been shown that features extracted from the later stages of the encoder are a better representation of the input than features extracted from the earlier stages [7]. ResNet and its variants are the most commonly used CNNs in CL [8, 7, 79].

3. **Projection Head:** After extraction, the embedding vectors h_i and h_j pass through an MLP to produce embeddings z_i and z_j on which the contrastive loss is computed (Figure 39). It has been proven that adding the MLP achieves better results [8, 79].

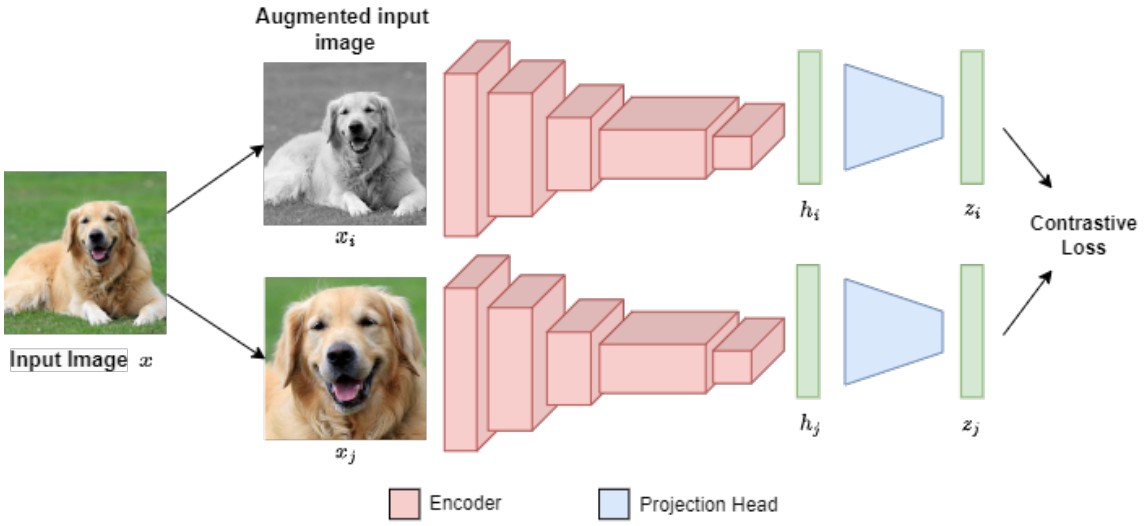


Figure 39: Encoder extracting embeddings h_i and h_j that pass trough a projection head producing z_i and z_j on which the contrastive loss is computed

4. **Similarity Measure:** The central idea in contrastive learning is to bring similar instances closer together and dissimilar instances far apart. A metric is needed to measure the closeness between representations [7]. Cosine similarity is one of the most commonly used similarity metrics, which measures the cosine of

the angle between two non-zero vectors. The cosine similarity is calculated for the z_i and z_j embeddings and is defined as [7, 79]:

$$sim(z_i, z_j) = \frac{z_i \cdot z_j}{\|z_i\|\|z_j\|}$$

where $\|\cdot\|$ is the Euclidean norm of the vector and "." is the dot product. This similarity ranges from 1 to -1 [79].

5. **Contrastive Loss:**

   A contrastive loss function is defined to penalize the network for getting different representations of different versions of the same image. The original image and the transformed image should provide similar predictions and produce similar features in the intermediate representations. Therefore, the loss function is minimized when the similarity between the query image and the positive embedding is greater and maximized when the dissimilarity between the two images is greater. Based on the loss, the representations of the encoder and the projection head improve over time and the obtained representations place similar image instances closer in the space and negatives far away [8].

   Widely used loss functions include the Noise-Contrastive Estimation Loss (NCE) [111], the Triplet Loss [112], and InfoNCE [113]. Current contrastive learning methods compare embeddings with a contrastive loss called *NT-Xent* loss (Normalized Temperature-Scaled Cross-Entropy Loss) [114, 110]. This loss function for a positive pair of examples (i, j) is defined as:

$$\ell_{i,j} = -log\frac{\exp(sim(z_i, z_j)/\tau)}{\sum_{k=1}^{2N} \mathbb{1}_{[k\neq i]} \exp(sim(z_i, z_k)/\tau)}$$

   where N is the number of samples, $2N$ are the transformed (augmented) pairs and $2(N-1)$ negative pairs from other examples in the dataset. $sim(z_i, z_j)$ represents the cosine similarity defined previously. The term in the numerator is the positive pairs and the terms in the denominator are the negative pairs. $\mathbb{1}_{[k\neq i]} \in 0, 1$ is an indicator function evaluating to 1 if and only if $k \neq i$ and τ denotes a temperature parameter. The final loss is computed across all positive pairs, both (i, j) and (j, i). The goal is to identify positive pairs of each z_i and repel others [8, 110].

Similar to other deep learning methods, CL uses a variety of optimization algorithms for effective training. The training involves learning the parameters of the network by minimizing the contrastive loss function. SGD and its variants are the most popular optimization algorithms used with CL methods [7].

Recently, CL methods for CV tasks have increased and some have begun to outperform supervised learning

methods. DeepCluster [84], ClusterFit [115], SimSiam [116], PIRL [117], MoCo [118], BYOL [119], SimCLR [110], and SwAV [83] are some CL methods. SwAV is described in the following section.

3.2.1 Swapping Assignments between multiple Views (SwAV)

Caron et al. [83] proposed an online clustering-based self-supervised method for learning visual features named **SwAV**.

The authors purposed a *multi-crop* augmentation strategy that generates multiple views of the same image instead of just one pair without quadratically increasing memory and computational requirements. They use two standard resolution crops and take V additional low resolution crops that cover only small portions of the image (Figure 40). Using low resolution images provides only a small increase in computational cost and the model becomes scale invariant. This strategy showed an improvement in performance not only for SwAV but also for other contrastive learning methods [83]. Next, random horizontal flips, color distortions, and Gaussian blur are applied to each resulting crop.

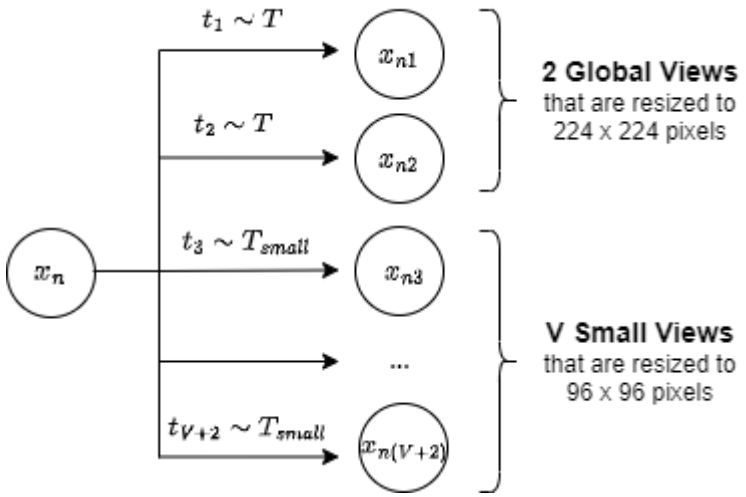


Figure 40: Multi-crop: image x_n is transformed into $V + 2$ views: two global views and V small resolution zoomed views

Each image x is transformed into augmented views x_1 and x_2 by applying a transform t selected randomly from a set of image transformations T. For simplicity, only 2 augmented views are listed here, but there can be many more by using multi-crop. The augmented views are passed through a CNN f_θ (ResNet50) that follows a projection head and outputs vectors z_1 and z_2 (Figure 41).

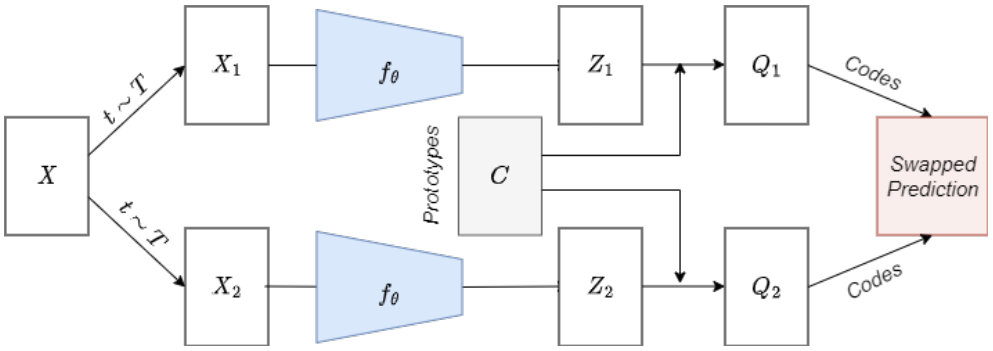


Figure 41: SwAV Architecture

The feature vectors z_1 and z_2 are then mapped to a set of K trainable prototype vectors $c_1, c_2, ..., c_K$. C is the matrix whose columns are $c_1, c_2, ..., c_k$. This mapping/code is denoted by $Q = [q_1, ...q_B]$ (Figure 42).

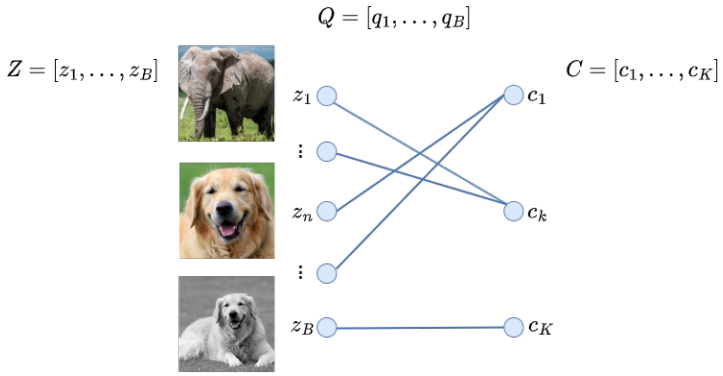


Figure 42: Assigning B samples to K trainable prototype vectors

Q is optimized to maximize the similarity between the features and the prototypes, i.e,

$$max_Q = Tr(Q^T C^T Z) + \epsilon H(Q)$$

where $H(Q) = -\sum_{ij} Q_{ij} log Q_{ij}$ is the entropy function and ϵ is a parameter that controls the smoothness of the mapping. In practice, ϵ is kept low because using a high value generally leads to a trivial solution where all samples fall into a unique representation and all are assigned to all prototypes uniformly.An equal partition is enforced by constraining the matrix Q so that each prototype is selected the same amount of times. Once a continuous solution Q^* is found it takes the form of a normalized exponential matrix:

$$Q^* = Diag(u) exp\left(\frac{C^T Z}{\epsilon}\right) Diag(v)$$

where u and v are renormalization vectors in $\mathbb{R}^K$ and $\mathbb{R}^B$ respectively. These vectors are computed using the iterative *Sinkhorn-Knopp* algorithm [120].

A "swapped" prediction problem is set up consisting of predicting the code q_1 from the feature z_2 and q_2 from z_1 with the following loss function:

$$L(z_1, z_2) = \ell(z_1, q_2) + \ell(z_2, q_1)$$

where $\ell(z_1, q_2)$ is the cross-entropy loss between the code and the probability obtained by taking a *softmax* of the dot products of z_i and all prototypes in C:

$$\ell(z_t, q_s) = -\sum_k q_s^{(k)} log p_t^{(k)}, \text{ where } p_t^{(k)} = \frac{exp((z_t^T c_k)/\tau)}{\sum_{k'} exp((z_t^T c_{k'})/\tau)}$$

where τ denotes a temperature parameter.

The loss function is minimized with respect to the prototypes C and the parameters θ of the encoder f_θ. This method compares features z_1 and z_2 using intermediate codes q_1 and q_2. If these two features capture similar information, it should be possible to predict the code from the other feature. The features are learned by Swapping Assignments between multiple Views (SwAV) of the same image. Figure 43 illustrates this idea.

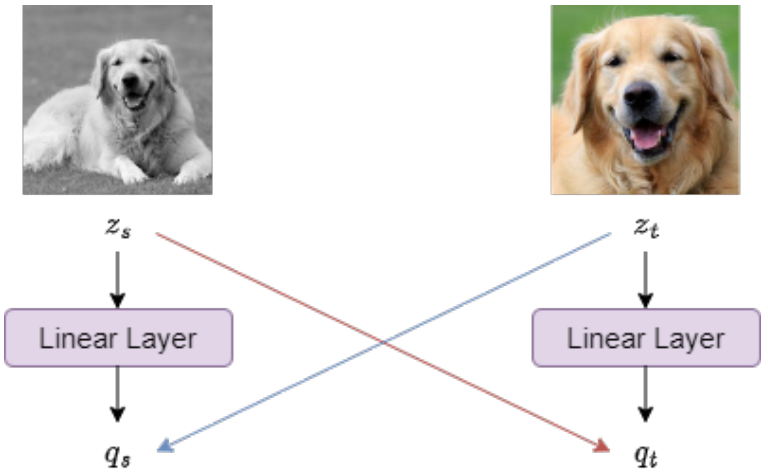


Figure 43: Swapped prediction problem between two views of the same image

3.3 Evaluation on Self-Supervised Learning

After learning the representations, either by pretext task or by contrastive learning, these representations must be evaluated to ensure quality. There are three ways to do so:

- **Linear Classification:** a linear classifier is trained on top of the CNN trained on the unlabeled dataset. The last fully connected layer is removed and the rest of the CNN is frozen, on which the classifier is trained. Evaluation is often performed on the same dataset that was used to train the network [8].

- **Fine-tuning with a % of labels:** the CNN trained with an SSL method/model is fine-tuned with a certain percentage of the labeled images (usually 1% or 10% from the same dataset). This is considered semi-supervised learning since the model is still trained with a few labels [83].
- **Transfer learning to downstream tasks:** Transfer learned representations to downstream tasks such as image classification, semantic segmentation, object recognition, and action recognition, etc., on a different data set. The performance of transfer learning on these high-level tasks shows the generalization ability of the learned features. If the CNN can learn general features, then the pre-trained models can be used as a good starting point for other vision tasks that require the acquisition of similar features from images or even small datasets that require this additional information.[7]

4 Development Tools and Datasets

The model was implemented on *Kaggle* [121], a cloud computational environment for data scientists and machine learning engineers. Kaggle allows users to find and publish datasets, build AI models, work with other enthusiasts, and even enter competitions to solve data science challenges. It provides a 4-core CPU with 30GB of RAM and an Nvidia Telsa P100 GPU with 13GB of RAM. GPUs are very helpful when using code that takes advantage of GPU-accelerated libraries. Each notebook editing session is provided with 12 hours of execution time for CPU and GPU and 20GB of auto-saved disk space.

As for the development environment, *Python* was the programming language used and *Tensorflow* the main library. Tensorflow [122] is an end-to-end open-source ML library developed by Google for the implementation and deployment of models. It contains many built-in functions that allow the creation of deep neural networks. In addition, the following libraries were also used: *Keras*, *Numpy*, *Scikit-Learn*, and *Matplotlip*.

Weights & Biases (W&B) [123] was used for experiment tracking and visualizations to develop insights for this work. It is a machine learning platform for developers to build better models faster. W&B has interoperable tools to track experiments, evaluate performance, reproduce models and visualize results.

4.1 Flowers Dataset

One of the datasets used in this work was *Tensorflow's flowers dataset* [124], which contains 3670 images and respective labels for flower classification. This dataset contains 5 different types of flowers: Dandelion (denoted by 0), Daisy (denoted by 1), Tulip (denoted by 2), Sunflower (denoted by 3), and Rose (denoted by 4). In Figure 44 are represented some image examples from the dataset.

Figure 44: Examples of images from the flowers dataset

4.2 BookCase Dataset

The *bookcase dataset* was a laboratory scenario created to resemble a real problem in an industrial space context. The goal is to classify whether or not a shelf is empty so that an alert can be triggered if it is empty.

The dataset was created at Neadvance company. Firstly, photographs were taken of a whole bookcase, with different proximity, as shown in Figure 45. The bookcase shelves contained books and other objects. In total 166 images were taken.

Figure 45: Examples of photographs taken of the bookcase

The images were then cropped to capture only the bookcase, removing wall and ceiling areas, as displayed in Figure 46.

Figure 46: Example of a bookcase image being cropped to the desired format

Finally, the images were sliced, using the *split-image* Python package, returning each square shelf as an image (Figure 47).

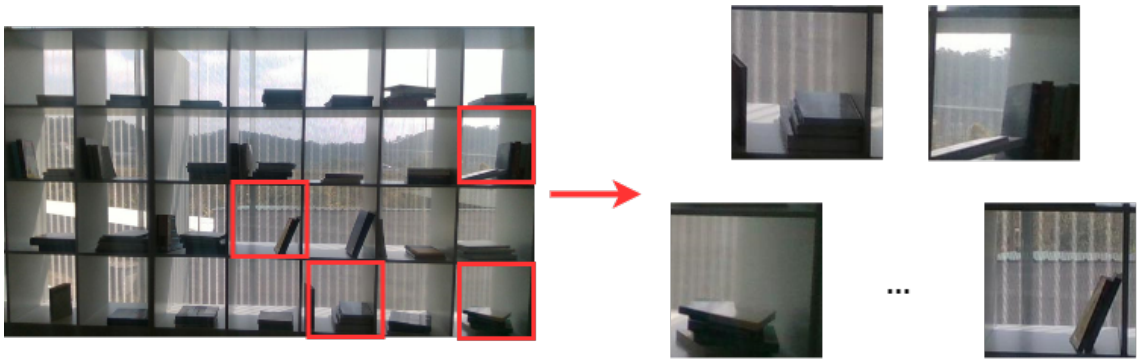


Figure 47: Example of a cropped bookcase image being sliced into tiles

The final result was a dataset containing 4597 images, of which 1094 are empty (denoted by 0) and 3505 are not empty (denoted by 1). Figure 48 illustrates examples of the two classes.

Figure 48: Examples of images from the final created bookcase dataset

4.3 Boxes Dataset

The *boxes dataset* was another laboratory scenario created at Neadvance. In this case, the goal is to classify whether the boxes are correctly positioned or if there is an anomaly in their positioning. This dataset contains a total of 116 images, 37 correct images (denoted 1) and 79 incorrect images (denoted 0). In Figure 49 are displayed some image examples from the dataset.

Figure 49: Examples of images from the created boxes dataset

5 Experiments and Results

The self-supervised learning method chosen for the classification task was SwAV. This choice was made due to it being the state-of-the-art method at the time of the research on SSL, as can be seen in the following articles[83, 8, 7, 79, 87] and Figure 50. The official implementation can be found on [125] free to use and modify. The implementation used in this thesis is a TensorFlow re-implementation and can be found on [126].

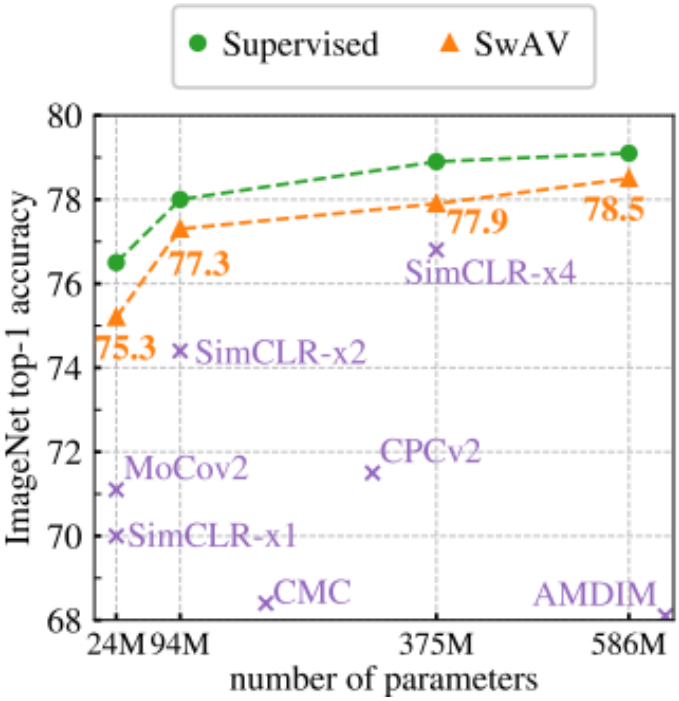


Figure 50: ImageNet Top-1 accuracy for linear models trained on frozen features from different self-supervised methods

5.1 Flowers Dataset

SwAV was initially trained on the flowers dataset to see its performance, to then train it to the custom dataset.

First of all *multi-crop* is applied to every batch of images fed to the network. From each image, **two 224x224** high resolution views and **three 96x96** low resolution views are generated. Data augmentation is applied to the resulting crop as described previously. The images are then fed to the backbone **ResNet50** model and through 2 Dense layers with ReLU (projection head). Throughout all the experiments the backbone model and projection head were the same.

For the SwAV training 5 experiments were made on this dataset. In all these experiments SwAV was trained

with 85% of the data, corresponding to 3120 images, and the evaluation of the model was with the remaining 15%, corresponding to 550 images (validation set). A linear classifier is trained on the frozen features with a single dense layer with 5 neurons, with softmax activation function and a l^2 regularizer.

The first experiment **A** was based on the implementation in [126]. The backbone model was initialized with random weights. SwAV was trained with mini-batch SGD using batches of 32 different instances. The mini-batch SGD used *polynomial decay schedule* decaying the learning rate from 0.1 to 0.01 using square root (power=0.5). The temperature parameter τ was set to 0.1 and the Sinkhorn regularization parameter ϵ was set to 0.05 for all runs, as was the number of prototypes that was set to 15. Early stopping was defined to stop training if the loss did not improve in 15 epochs. The experiment was set to train in 50 epochs and stopped at 17 with a loss of 2.30305. A linear classifier is trained on top of the frozen final representations of a ResNet50 trained with SwAV. This linear layer is trained during 100 epochs, with categorical cross-entropy loss, ADAM optimizer, and a learning rate of 0.001. Early stopping was defined to restore the best weights and stop training if the validation loss did not improve in 5 epochs. Experiment A obtained an accuracy of 44,7%, a precision of 48%, a recall of 46%, and a F_1 Score of 44%. Building the model from scratch initialized with random weights results in an inefficient solution as the network starts from a point where it does not know anything.

In experiment **B** the backbone model was initialized with *imagenet* weight and the end learning rate was decreased from 0.01 to 0.001. Initializing the model with the weights of the *imagenet*, a huge dataset with various images, allows the model to start from a point with a network capable of extracting meaningful features from images, as opposed to a random start, leading to better training. High learning rates result in rapid changes and require fewer training epochs (Figure 51 - Experiment A) but often result in a sub-optimal final set of weights. For that reason, the learning rate was decreased in this experiment. The other parameters remained equal to experiment A. When training SwAV the model finished all 50 epochs and returned a loss of 2.30368. Meaning that the model could still improve if given more epochs to train. These features obtained an accuracy of 85,6%, which was a great improvement from experiment A. The precision, recall and F_1 Score were all equal to 86%, meaning that the number of false positives is equal to the number of false negatives. This experiment showed that a better start point (by adding *imagenet* weights) and a lower learning rate allows the model to better train, resulting in a boost in its performance.

In experiment **C** the number of epochs was increased to 500, since the previous experiment would still improve without the limit of 50 epochs, and the end learning rate further decreased to 0.0001. SwAV stopped at epoch 225 with a loss of 2.30538. As expected, the model kept training after the first 50 epochs, and achieved a even better result than in experiment B. The linear classifier achieved an accuracy of 88%, equal to all the other

metrics, and a loss of 0.327 in epoch 39.

Figure 51 illustrates the loss function during training in experiments A, B, and C.

Figure 51: Loss during SwAV training in experiments A, B, and C

In experiments **D** and **E** a different optimizer schedule was used, the *inverse time decay*. This schedule applies the inverse decay function to an optimizer step, given a provided initial learning rate, which was 0.1 and 0.001 in experiments D and E, respectively. Features from experiment D achieved 86% accuracy and from experiment E achieved 87%. Both optimizer schedules show better results with lower learning rates.

Tables 1 and 2 show the results of all 5 experiments of SwAV training and of the linear classifier, respectively. Figure 52 illustrates the loss during SwAV training and figures 53 and 54 illustrate the validation loss and accuracy during the linear classifier training, respectively, for all 5 experiments.

The lowest loss value in SwAV training was in experiment A, which does not indicate it was the best model. The best parameter to choose are metrics with data not used for training. A low loss of A does not represent the performance of the model with new data and its generalization progress.

Exp	Init. W	O. Schedules	I. lr	DS	E. lr	P	DR	Epochs	Stop	Loss
A	None	Poly. Decay	0.1	5	0.01	0.5	-	50	17	2.30205
B	ImageNet	Poly. Decay	0.1	5	0.001	0.5	-	50	50	2.30368
C	ImageNet	Poly. Decay	0.1	5	0.0001	0.5	-	500	225	2.30538
D	ImageNet	I. Time Decay	0.1	50	-	-	5	500	88	2.30346
E	ImageNet	I. Time Decay	0.001	100	-	-	5	500	125	2.37668

Table 1: Results of SwAV training with different hyperparameters on flowers dataset. **Exp**: Experiment, **Init. W**: Initial Weights, **O. Schedules**: Optimizer Schedules, **I.lr**: Initial learning rate, **DS**: Decay Steps (how often to apply decay), **E. lr**: End learning rate, **P**: Power, **DR**: Decay Rate, and **Stop**: Stopped at.

Experiment	Stop	Best Epoch	Loss	Accuracy	Precision	Recall	F1-Score
A	100	100	1.298	0.447	0.48	0.46	0.44
B	26	21	0.412	0.856	0.86	0.86	0.86
C	44	39	**0.327**	**0.882**	**0.88**	**0.88**	**0.88**
D	29	24	0.397	0.860	0.86	0.86	0.86
E	27	22	0.348	0.871	0.87	0.87	0.87

Table 2: Results of a linear classifier on SwAVs frozen features with flowers dataset

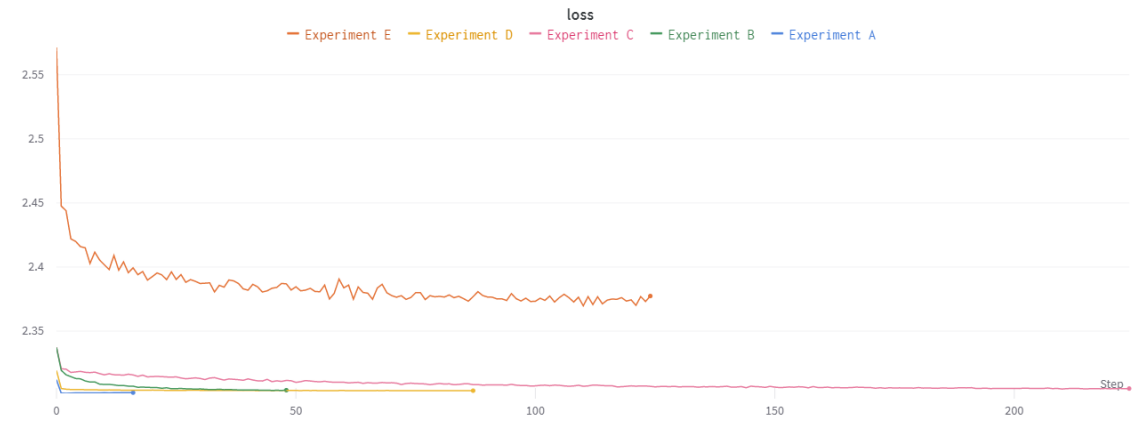


Figure 52: Loss during SwAV training in all experiments

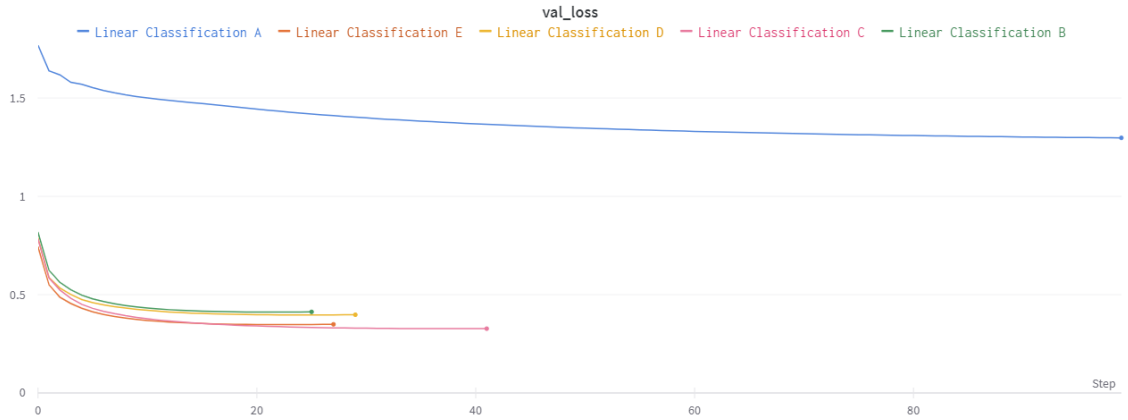


Figure 53: Validation loss during linear classifier training in all experiments

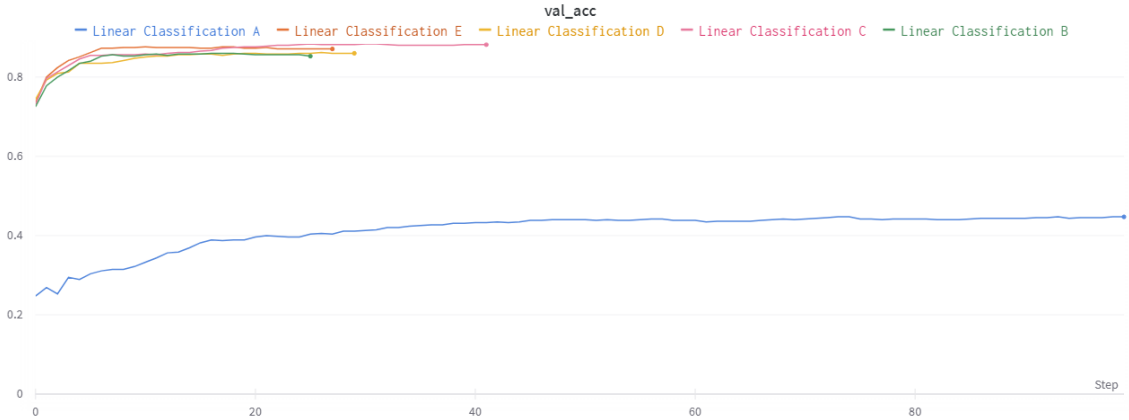
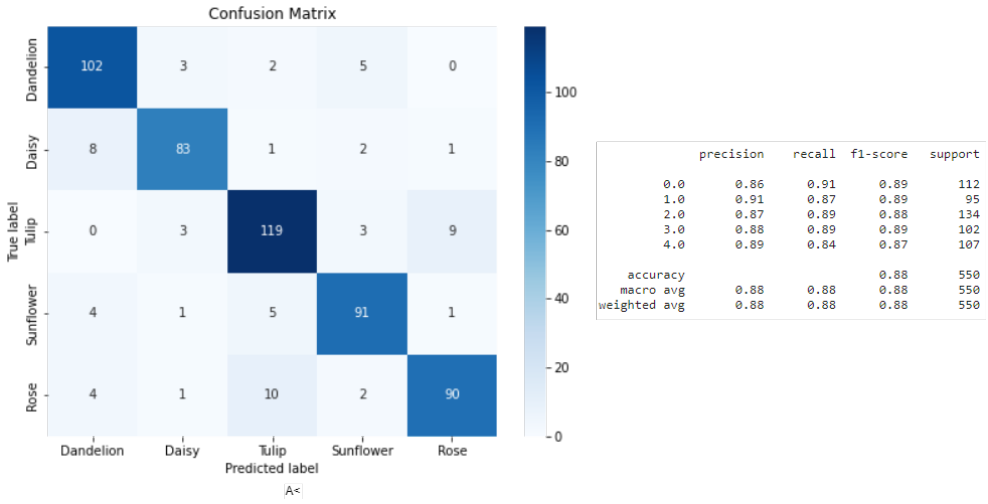


Figure 54: Validation accuracy during linear classifier training in all experiments

Given the experiments made, experiment **C** was the one that achieved better performance. In more detail Figure 55 shows the confusion matrix and metric values obtained. The model predicted incorrectly 71 images out of a total of 550 images. Dandelion is the class with the highest recall as it is the class that fewer images were misclassified, only 8.9%. Rose has the highest percentage of misclassified images - 15.9%. Daisy is the class with less false positives, having the highest precision. We can also observe that the average metrics precision, recall, and F_1 Score are all equal to 88% due to the fact that the $FN = 65 = FP$.

These experiments could have been optimized to achieve better results but because this dataset was not the focus of the work no more experiments were made.



	precision	recall	f1-score	support
0.0	0.86	0.91	0.89	112
1.0	0.91	0.87	0.89	95
2.0	0.87	0.89	0.88	134
3.0	0.88	0.89	0.89	102
4.0	0.89	0.84	0.87	107
accuracy			0.88	550
macro avg	0.88	0.88	0.88	550
weighted avg	0.88	0.88	0.88	550

Figure 55: Confusion matrix and metric values from experiment C

5.2 Bookcase Dataset

As described previously the bookcase dataset was created to assimilate to a real problem in an industrial space context. As experiment **C** achieved the best performance in the flowers dataset, the SwAV training and linear classifier on the bookcase dataset were done in the exact same settings. The only difference made was an increase in the early stopping of SwAV training. The training was defined to stop if the loss did not improve in 30 epochs, instead of 15. The dataset was also divided into 85% training and 15% validation, corresponding to 3907 and 690 images, respectively. This experiment was defined as **F** and the results are shown in Table 3 as are the plots in Figures 56 and 57.

Exp	**Init. W**	**O.Schedules**	**I. lr**	**DS**	**E. lr**	**P**	**Stop**	**Loss**
F	ImageNet	Poly. Decay	0.1	5	0.0001	0.5	308*/500	2.301
	Stop	**Best Epoch**	**Loss**	**Accuracy**	**Precision**	**Recall**	**F1-Score**	**MCC**
	100	100	0.032	0.988	0.99	0.99	0.99	0.98

Table 3: Results of SwAV training on bookcase dataset and results of the linear classifier on SwAVs frozen features with the same dataset

Figure 56: Loss during SwAV training on bookcase dataset

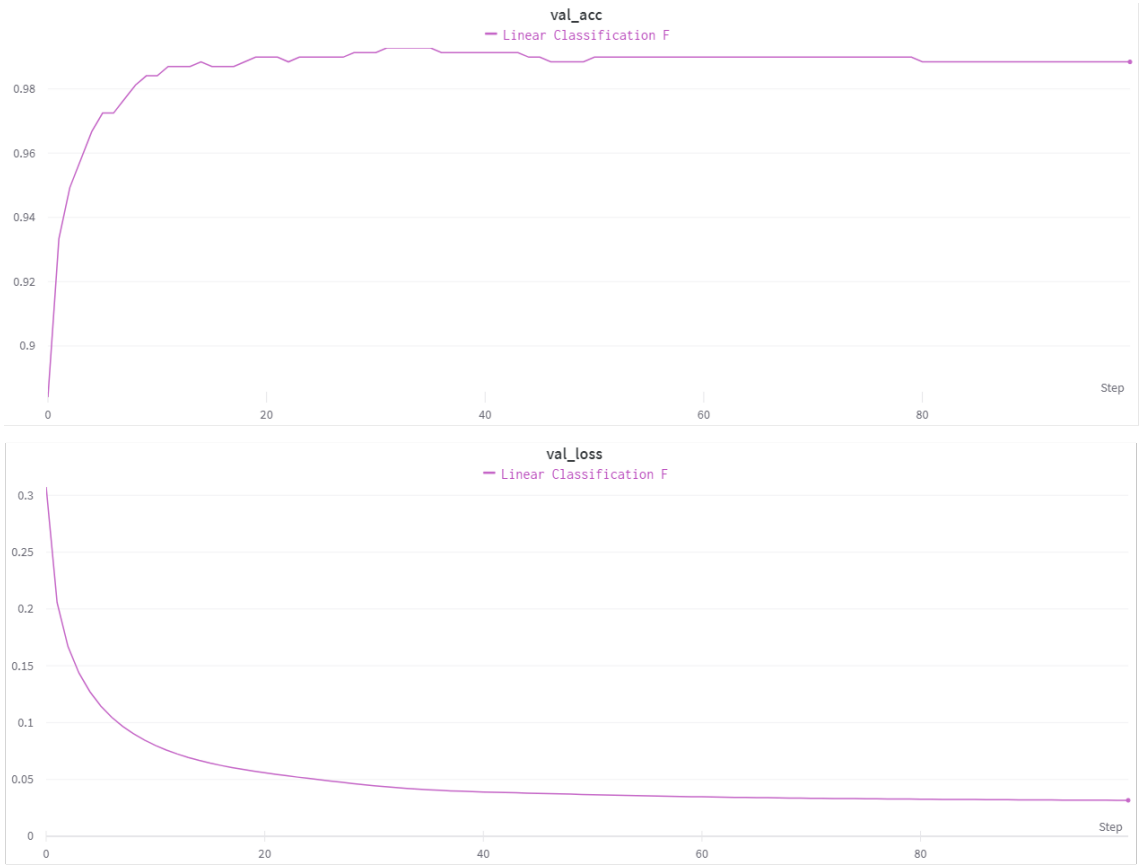


Figure 57: Validation accuracy (Top) and validation loss (Bottom) during linear classifier training on bookcase dataset

During SwAV training F achieved a loss of 2.301 in 308 epochs. The model stopped training due to *Kaggle's* 12 hours execution time limitation. However, the linear classifier on these frozen features still achieved 98.8%

accuracy, 98% MCC, and 99% on the remaining metrics. The score obtained shows that the binary classifier was able to predict the majority of positive and negative instances.

Figure 58 displays the confusion matrix and the metric values from experiment F. The model only misclassified 6 images from a total of 690.

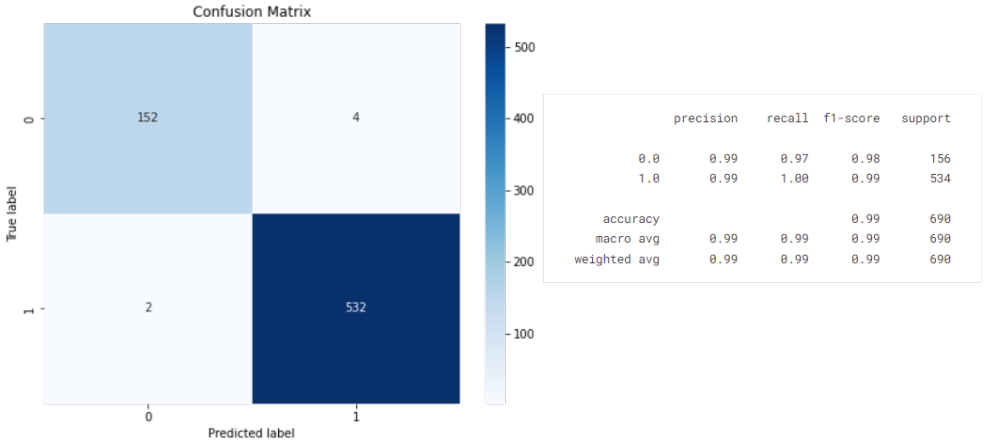


Figure 58: Confusion matrix and metric values from experiment F

The bookcase dataset was also trained in a fully supervised manner with hyperparameter optimization to indicate the best model possible. Training took several days and achieved a 99.6% accuracy. Despite the self-supervised method obtaining a lower accuracy, it took around 12 hours with is much less compared to the supervised setting.

There was an attempt to do more experiments. One was to increase the batch size from 32 to 64 as SwAV works well on small and large batches [83]. This was not possible to test due to an *out of GPU memory* problem. Another experiment made was the use of a different backbone model. The settings were equal to experiment F except for the backbone model, where ResNet50 was replaced by VGGnet-16 and VGGnet-19. The linear classifier on the frozen features achieved 81,3% and 77,4% accuracy, respectively.

5.3 Transferring to Downstream tasks

Another experiment made was transfer learning to a downstream classification task. The learned representations from SwAV training on the bookcase dataset were used to solve the classification problem with the boxes dataset. A linear classifier is trained on the frozen features (learned with the bookcase dataset) with a single dense layer with 2 neurons, with softmax function and a l^2 regularizer. The dataset was divided into 70% for training and 30% for validation. The dataset was divided this way because it only contains 116 images. Dividing it for example 70%

for training, 20% for validation, and 10% for testing makes the test set contain only 12 images which would not be enough to evaluate the model. To increase the test set, the training set would have to be decreased which would give fewer images for the model to learn from. The model was trained during 500 epochs and completed the total number of epochs. Despite the 30 epochs defined for early stopping the model kept on improving (Figure 59).

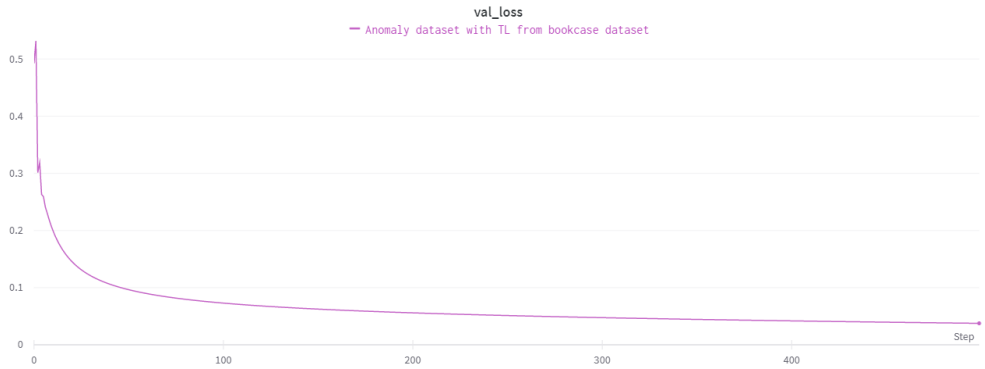


Figure 59: Validation loss of the linear classifier on the boxes dataset

Figure 60 displays the results achieved. The model correctly predicted all of the 35 images, achieving 100% in all metrics.

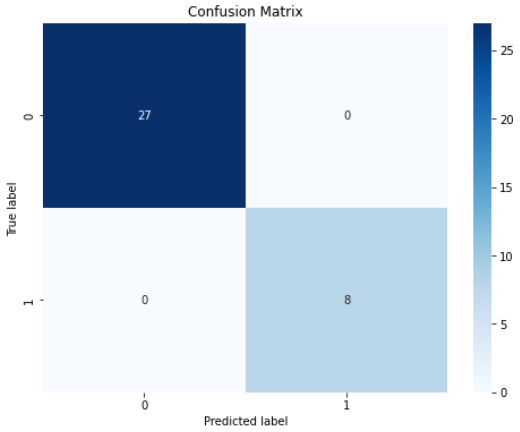


Figure 60: Confusion matrix of the linear classifier on the boxes dataset

This dataset was also trained in a fully supervised manner where 10 models were trained with different hyperparameters to obtain the best possible model. For a fair comparison, the dataset was also divided into 70% for training and 30% for validation. Figure 61 shows the results obtained. The supervised model misclassified 8 images, achieving an accuracy of 81.2% and an MCC of 54.5%. The reason for these results is that the dataset

contains very few images for the model to learn. In this specific case and with these settings, the information gained on the bookcase dataset was useful to improve performance compared to the supervised model.

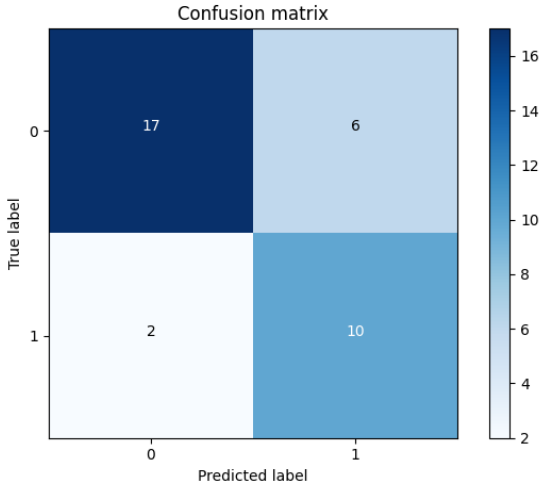


Figure 61: Confusion matrix of the model in a supervised seeting

However, when given new images the model did not classify them all correctly. Since the new images are a bit different from the images the model learned from (Example in Figure 62), this shows that the model did not generalize well. This also happens due to the dataset being small and very specific. During SwAV training on the bookcase dataset, certain features were learned. The linear classifier grabbed those features and tried to fit them into the two classes of the boxes dataset. However, the meaningful features of the bookcase problem can bring the model to associate the new images to incorrect classes when regarding the new problem. This happens because the learned features are not adequate for the new problem. One way to achieve better results would be to acquire more images with different noise and also train SwAV with this dataset. SwAV would extract specific features from this dataset which would improve performance.

Figure 62: New image with different angle and background noise. The model predicted it incorrectly as 0

6 Conclusions and future work

6.1 Conclusion

The main goal of this work was to explore and research this new type of learning: self-supervised learning and to develop a classification model in a self-supervised manner that would classify if a shelf is empty or not in an industrial space context. After an investigation of the available methods, SwAV was chosen.

In five experiments, SwAV was trained with different parameters using the flower dataset from Tensorflow. After training, the models were trained with a linear classifier based on the frozen final representations of the ResNet-50 trained with SwAV. The results were analyzed using various metrics. Based on these results, SwAV was then trained on the custom bookcase dataset with the same settings and achieved an accuracy of 98.8% and an MCC of 98% for the linear classifier. The SSL setting achieved 98.8% in much less time compared to the SL setting which achieved 99,6%. However, SwAV and other SSL methods require a lot of computational power which can be a limitation. Depending on the problem and resources, SSL can be an option for being faster and getting close to fully supervised learning performance.

Regarding transfer learning to downstream tasks the results were not as expected. Despite the self-supervised method improving performance compared to the supervised model, the model did not generalize well. Given new images the model failed to predict them correctly, showing that the features learned from the bookcase model were not adequate to solve the new problem with the boxes dataset. This is not surprising because SwAV was trained on a very specific dataset with only two classes. It does not extract the same features as if it was trained on, for example, *ImageNet* that has over a million images and 1000 classes.

In general, SwAV extracts useful features that can add information to solve other tasks. However, this depends on the datasets, the task, and the features learned during SwAV training.

6.2 Future Work

In terms of classifying whether a shelf is empty or not in an industrial setting, the aim of future work is to capture real images in an industrial environment in order to create a new dataset and test the developed model. Further, train SwAV with the boxes dataset and even with both bookcase and boxes datasets and evaluate. As for SwAV in particular, the experiments conducted with the other backbones should be optimized and the results compared. Also, with more resources, experiment F should be trained with a higher number of epochs and evaluated. Another goal would be to implement other recent methods of self-supervised learning methods such as *BYOL*, *ReLICv2*, and *MoCov3*, which should also be tested in an industrial context.

References

[1] T. Darrell R. Girshick, J. Donahue and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *arXiv*, 2014.

[2] I. Sutskever A. Krizhevsky and G.E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, volume 25, page 1097–1105, 2012.

[3] I. Kokkinos K. Murphy L.C. Chen, G. Papandreou and A.L. Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *arXiv*, 2017.

[4] X. Zhai A. Kolesnikov and L. Beyer. Revisiting self-supervised visual representation learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 1920–1929, 2019.

[5] A. Gupta P. Goyal, D. Mahajan and I. Misra. Scaling and benchmarking self-supervised visual representation learning. In *Proceedings of the IEEE/CVF International Conference on computer vision*, pages 6391–6400, 2019.

[6] Z. Hou L. Mian Z. Wang J. Zhang X. Liu, F. Zhang and J. Tang. Self-supervised learning: Generative or contrastive. *IEEE Transactions on Knowledge and Data Engineering*, 2021.

[7] M.Z. Zadeh D. Banerjee A. Jaiswal, A.R. Babu and F. Makedon. A survey on contrastive self-supervised learning. *Technologies*, 9(1):2, 2020.

[8] K. Ohri and M. Kumar. Review on self-supervised image recognition using deep neural networks. *Knowledge-Based Systems*, 224:107090, 2021.

[9] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*, chapter 1. Pearson, fourth edition, 2021.

[10] S. Shah S. Khan, H. Rahmani and M. Bennamoun. *A Guide to Convolutional Neural Networks for Computer Vision*, chapter 1,3,4,5,6. Morgan and Claypool publishers, 2018.

[11] Data definition. `https://dictionary.cambridge.org/pt/dicionario/ingles/data`. Last accessed: 05-09-22.

[12] S. Sah. Machine learning: A review of learning types. *preprints*, 2020.

[13] A. Burkov. *The Hundred Page Machine Learning Book*, chapter 1,5. 2019.

[14] Y. Zhang. *New Advances in Machine Learning*, chapter 3. In-Tech, 2010.

[15] I. Sarker. Machine learning: Algorithms, real world applications and research directions. *Springer Nature*, 2021.

[16] R. C. Tryon. Genetic differences in maze-learning ability in rats. *Yearbook of the National Society for Studies in Education*, pages 111–119, 1940.

[17] X. Zhu and A. Goldberg. *Introduction to Semi-Supervised Learning*, page Abstract. Morgan and Claypool publishers, 2009.

[18] M. Sober J. Benedito E. Olivas, J. Guerrero and A. López. *Handbook of Research on Machine Learning Applications and Trends: Algorithms, Methods, and Techniques*, chapter 11. Information Science Reference, 2009.

[19] M.Kamber J.Han and J.Pei. *Data Mining: Concepts and Techniques*, chapter 3, 8.5. Elsevier, third edition, 2012.

[20] D. Bourke A. Neagoie and Zero to Mastery. Tensorflow developer certificate in 2022: Zero to mastery. `https://www.udemy.com/course/tensorflow-developer-certificate-machine-learning-zero-to-mastery/`, 2022. Appendix: Machine Learning and Data Science Framework (`https://www.mrdbourke.com/a-6-step-field-guide-for-building-machine-learning-projects/`) Last accessed 05-09-22.

[21] J. Brownlee. *Master Machine Learning Algorithms - Discover how they work and Implement Them From Scratch*, chapter 7. Machine Learning Mastery, 2016.

[22] J. Brownlee. *Machine Learning Mastery with Python: Understand Your Data, Create Accurate Models and Work Projects End-To-End*, chapter 10.3. Machine Learning Mastery, 2016.

[23] Rok Blagus and Lara Lusa. Class prediction for high-dimensional class-imbalanced data. *BMC Bioinformatics*, 11(1):523, 2010.

[24] Davide Chicco and Giuseppe Jurman. The advantages of the Matthews correlation coefficient (MCC) over F1 score and accuracy in binary classification evaluation. *BMC Genomics*, 21(1):6, 2020.

[25] L. Deng and D. Yu. *Deep Learning: Methods and Applications*, chapter 1. Now Publishers Inc, 2014.

[26] D. Graupe. *Principles Of Artificial Neural Networks: Basic Designs To Deep Learning*, chapter 2,4. World Scientific Publishing Company, 3 edition, 2013.

[27] M.B. Khan M. Mohammed and E.B.M Bashier. *Machine Learning: Algorithms and Applications*, chapter 6. CRC Press, 2017.

[28] A. Karpatne P.N. Tan, M. Steinbach and V. Kumar. *Introduction to Data Mining*, chapter 6. Pearson Education, 2 edition, 2019.

[29] J.D. Kelleher. *Deep Learning*, chapter 3, 5. MIT Press, 2019.

[30] S. Shalev-Shwartz and S. Ben-David. *Understanding Machine Learning: From Theory to Algorithms*, chapter 20. Cambridge University Press, 2014.

[31] F. Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[32] O. Campesato. *Artificial Intelligence, Machine Learning and Deep Learning*, chapter 4. Mercury Learning & Information, 2020.

[33] L. Fausett. *Fundamentals of neural networks: architectures, algorithms, and applications*, chapter 1. Prentice-Hall, 1994.

[34] U. Zahoora A. Khan, A. Sohail and A.S. Qureshi. A survey of the recent architectures of deep convolutional neural networks. volume 53, page 5455–5516. Kluwer Academic Publishers, 2020.

[35] ImageNet Large Scale Visual Recognition Competition (ILSVRC). `https://www.image-net.org/challenges/LSVRC/index.php`. Last accessed: 31-10-22.

[36] S. Albawi. T.A. Mohammed and S. Al-Zawi. Understanding of a convolutional neural network. In *2017 International Conference on Engineering and Technology (ICET)*, pages 1–6, 2017.

[37] K. O'Shea and R. Nash. An introduction to convolutional neural networks. 2015.

[38] J. Brownlee. *Deep Learning for Computer Vision - Image Classification, Object Detection and Face Recognition in Python*, chapter 1, 11, 12, 18. Machine Learning Mastery, 2019.

[39] Y. Bengio I. Goodfellow and A. Courville. *Deep Learning*, chapter 6,7,8,9. MIT Press, 2016.

[40] J. Kuen L. Ma A. Shahroudy B. Shuai T. Liu X. Wang G. Wang J. Cai J. Gu, Z. Wang and T. Chen. Recent advances in convolutional neural networks. volume 77, pages 354–377, 2017.

[41] H. Qian B. Ding and J. Zhou. Activation functions and their characteristics in deep neural networks. In *2018 Chinese control and decision conference (CCDC)*, pages 1836–1841. IEEE, 2018.

[42] L. Nanni G. Maguolo and S. Ghidoni. Ensemble of convolutional neural networks trained with different activation functions. volume 166, page 114048. Elsevier, 2021.

[43] Y. Bengio Y. LeCun and G.Hinton. Deep learning. volume 521, pages 436–444, 2015.

[44] A. Moawad. Neural networks and back-propagation explained in a simple way. `https://medium.com/datathings/neural-networks-and-backpropagation-explained-in-a-simple-way-f540a3611f5e`, 2018. Last accessed: 04-01-23.

[45] R. Golden D.E. Rumelhart, R. Durbin and Y. Chauvin. Backpropagation: The basic theory. *Backpropagation: Theory, architectures, and applications*, pages 1–34, 1995.

[46] M. Kalirane. Gradient descent vs. backpropagation: What's the difference? `https://www.analyticsvidhya.com/blog/2023/01/gradient-descent-vs-backpropagation-whats-the-difference/`, 2023. Last accessed: 04-01-23.

[47] E. Hazan J. Duchi and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. volume 12, pages 2121–2159, 2011.

[48] G.Hinton T. Tieleman et al. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. volume 4, pages 26–31, 2012.

[49] D.P. Kingma and J.L. Ba. Adam: A method for stochastic optimization. *arXiv*, 2014.

[50] Z. Nado J. Lee C.J. Maddison D. Choi, C.J. Shallue and G.E. Dahl. On empirical comparisons of optimizers for deep learning. 2019.

[51] Y. Bohra. The challenge of vanishing/exploding gradients in deep neural networks. `https://www.analyticsvidhya.com/blog/2021/06/`

the-challenge-of-vanishing-exploding-gradients-in-deep-neural-networks/, 2021. Last accessed: 04-01-23.

[52] S. Gupta. The 7 most common machine learning loss functions. https://builtin.com/machine-learning/common-loss-functions, 2022. Last accessed: 02-01-23.

[53] F. Moreno. Sparse categorical cross-entropy vs categorical cross-entropy. https://towardsdatascience.com/cross-entropy-loss-function-f38c4ec8643e, 2021. Last accessed: 02-01-23.

[54] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015.

[55] A.J. Humaidi A. Al-Dujaili Y. Duan O. Al-Shamma J. Santamaría M.A. Fadhel M. Al-Amidie L. Alzubaidi, J. Zhang and L. Farhan. Review of deep learning: Concepts, cnn architectures, challenges, applications, future directions. volume 8, pages 1–74. Springer, 2021.

[56] A. Z. Simonyan and Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[57] S. Ren K. He, X. Zhang and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer Vision and Pattern Recognition*, page 770–778, 2015.

[58] K. Acharya A. Ajit and A. Samanta. A review of convolutional neural networks. In *2020 international conference on emerging trends in information technology and engineering (ic-ETITE)*, pages 1–5. IEEE, 2020.

[59] E.R. Davies. *Computer Vision: Principles, Algorithms, Applications, Learning*, chapter 1. Academic Press, fifth edition, 2018.

[60] IBM. What is computer vision? https://www.ibm.com/topics/computer-vision. Last accessed: 28-10-22.

[61] D. H. Hubel and T. N. Wiesel. Receptive fields of single neurones in the cat's striate cortex. volume 148, pages 574–591, 1959.

[62] C. Ray R. A. Kirsch, L. Cahn and G. H. Urban. Experiments in processing pictorial information with a digital computer. In *Papers and Discussions Presented at the December 9-13, 1957, Eastern Joint Computer*

Conference: Computers with Deadlines to Meet, IRE-ACM-AIEE '57 (Eastern), page 221–229. Association for Computing Machinery, 1957.

[63] L. Roberts. *Machine perception of three-dimensional solids*. PhD thesis, Massachusetts Institute of Technology, 1963.

[64] S. Papert. The summer vision project. *Massachusetts Institute of Technology, Artificial Intelligence Group*, 1966.

[65] D. Marr. Vision: A computational investigation into the human representation and processing of visual information. *WH San Francisco: Freeman and Company, San Francisco*, 1982.

[66] K. Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition. In *Competition and cooperation in neural nets*, pages 267–285. Springer, 1982.

[67] J.S. Denker D. Henderson-R.E. Howard W. Hubbard Y. LeCun, B. Boser and L.D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

[68] Y. Bengio Y. LeCun, L. Bottou and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[69] D. Lowe. Object recognition from local scale-invariant features. In *Proceedings of the seventh IEEE international conference on computer vision*, volume 2, pages 1150–1157. Ieee, 1999.

[70] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *Proceedings of the 2001 IEEE computer society conference on computer vision and pattern recognition. CVPR 2001*, volume 1, pages I–511 – I–518, 2001.

[71] The PASCAL Visual Object Classes Homepage. `http://host.robots.ox.ac.uk/pascal/VOC/`. Last accessed: 31-10-22.

[72] R. Szeliski. *Computer Vision: Algorithms and Applications*, chapter 1. Springer, second edition, 2022.

[73] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv*, 2015.

[74] E. Shelhamer J. Long and T. Darrell. Fully convolutional networks for semantic segmentation. *arXiv*, 2015.

[75] P. Dollar K. He, G. Gkioxari and R. Girshick. Mask r-cnn. *arXiv*, 2018.

[76] R. Girshick. Fast r-cnn. *arXiv*, 2015.

[77] R. Girshick S. Ren, K. He and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *arXiv*, 2016.

[78] S.-M. Schröder L. Schmarje, M. Santarossa and R. Koch. A survey on semi-, self-and unsupervised learning for image classification. *IEEE Access*, 9:82146–82168, 2021.

[79] S. Albelwi. Survey on self-supervised learning: Auxiliary pretext tasks and contrastive learning methods in imaging. *Entropy*, 24(4):551, 2022.

[80] L. Jing and Y. Tian. Self-supervised visual feature learning with deep neural networks: A survey. *IEEE transactions on pattern analysis and machine intelligence*, 43(11):4037–4058, 2020.

[81] B. Dickson. Meta's yann lecun is betting on self-supervised learning to unlock human-compatible ai. `https://thenextweb.com/news/metas-yann-lecun-is-betting-on-self-supervised-learning-to-unlock-human-compatible-a` `//thenextweb.com/news/metas-yann-lecun-is-betting-on-self-supervised-learning-to-unl` 2022. Last accessed: 16-01-23.

[82] Y. LeCun and I. Misra. Self-supervised learning: The dark matter of intelligence. `https://ai.facebook.com/blog/self-supervised-learning-the-dark-matter-of-intelligence/`, 2021. Last accessed: 16-01-23.

[83] J. Mairal P. Goyal-P. Bojanowski M. Caron, I. Misra and A. Joulin. Unsupervised learning of visual features by contrasting cluster assignments. *Advances in Neural Information Processing Systems*, 33:9912–9924, 2020.

[84] A. Joulin M. Caron, P. Bojanowski and M. Douze. Deep clustering for unsupervised learning of visual features. In *Proceedings of the European conference on computer vision (ECCV)*, pages 132–149, 2018.

[85] Y. Jia P. Sermanet-S. Reed D. Anguelov D. Erhan V. Vanhoucke C. Szegedy, W. Liu and A. Rabinovich. Going deeper with convolution. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[86] Zhuang L. Van Der Maaten G. Huang, Z. Liu and K.Q. Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

[87] Y.-C. Hsu T.-L. Liu Y. Chen C.-H. Yeh, C.-Y. Hong and Y. LeCun. Decoupled contrastive learning. In *European Conference on Computer Vision*, pages 668–684. Springer, 2022.

[88] A.A. Efros T. Xiao, X. Wang and T. Darrell. What should not be contrastive in contrastive learning. *arXiv preprint arXiv:2008.05659*, 2020.

[89] N. Komodakis S. Gidaris, P. Singh. Unsupervised representation learning by predicting image rotations. pages 1–16, 2018.

[90] J. Liu L. Jing, X. Yang and Y. Tian. Self-supervised spatiotemporal feature learning via video rotation prediction. *arXiv preprint arXiv:1811.11387*, 2018.

[91] J. Donahue T. Darrell D. Pathak, P. Krahenbuhl and A.A Efros. Context encoders: Feature learning by inpainting. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2536–2544, 2016.

[92] E. Simo-Serra S. Iizuka and H. Ishikawa. Globally and locally consistent image completion. *ACM Transactions on Graphics (ToG)*, 36(4):1–14, 2017.

[93] P. Isola R. Zhang and A.A. Efros. Colorful image colorization. In *European conference on computer vision*, pages 649–666. Springer, 2016.

[94] P. Isola R. Zhang and A.A. Efros. Split-brain autoencoders: Unsupervised learning by cross-channel prediction. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1058–1067, 2017.

[95] M. Maire G. Larsson and G. Shakhnarovich. Learning representations for automatic colorization. In *European conference on computer vision*, pages 577–593. Springer, 2016.

[96] M. Maire G. Larsson and G. Shakhnarovich. Colorization as a proxy task for visual understanding. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6874–6883, 2017.

[97] F. Huszár J. Caballero A. Cunningham-A. Acosta A. Aitken A. Tejani J. Totz Z. Wang et al. C. Ledig, L. Theis. Photo-realistic single image super-resolution using a generative adversarial network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4681–4690, 2017.

[98] A. Gupta C. Doersch and A.A. Efros. Unsupervised visual representation learning by context prediction. In *Proceedings of the IEEE international conference on computer vision*, pages 1422–1430, 2015.

[99] M. Noroozi and P. Favaro. Unsupervised learning of visual representations by solving jigsaw puzzles. In *European conference on computer vision*, pages 69–84. Springer, 2016.

[100] D. Yoo D. Kim, D. Cho and I.S. Kweon. Learning image representations by completing damaged jigsaw puzzles. In *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 793–802. IEEE, 2018.

[101] X. Ren Y. Xia C. Su-J. Liu Q. Tian C. Wei, L. Xie and A.L. Yuille. Iterative reorganization with weak spatial constraints: Solving arbitrary jigsaw puzzles for unsupervised representation learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1910–1919, 2019.

[102] J.-B. Huang M. Singh H.-Y., Lee and M.-H. Yang. Unsupervised representation learning by sorting sequences. In *Proceedings of the IEEE international conference on computer vision*, pages 667–676, 2017.

[103] C. Doersch and A. Zisserman. Multi-task self-supervised visual learning. In *Proceedings of the IEEE international conference on computer vision*, pages 2051–2060, 2017.

[104] Wikipedia contributors. Cielab color space. `https://en.wikipedia.org/wiki/CIELAB_color_space`, 2023. Last accessed: 21-01-23.

[105] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2007 (VOC2007) Results. `http://host.robots.ox.ac.uk/pascal/VOC/voc2007/`. Last accessed: 21-01-23.

[106] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results. `http://host.robots.ox.ac.uk/pascal/VOC/voc2012/`. Last accessed: 21-01-23.

[107] H. Gouk L. Ericsson and T.M. Hospedales. How well do self-supervised models transfer? In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5414–5423, 2021.

[108] W. Falcon and K. Cho. A framework for contrastive self-supervised learning and designing a new approach. *arXiv preprint arXiv:2009.00104*, 2020.

[109] X. Wang and G.-J. Qi. Contrastive learning with stronger augmentations. *arXiv preprint arXiv:2104.07713*, 2021.

[110] M. Norouzi T. Chen, S. Kornblith and G. Hinton. A simple framework for contrastive learning of visual representations. In *International conference on machine learning*, pages 1597–1607. PMLR, 2020.

[111] M. Gutmann and A. Hyvärinen. Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 297–304. JMLR Workshop and Conference Proceedings, 2010.

[112] M. Schultz and T. Joachims. Learning a distance metric from relative comparisons. *Advances in neural information processing systems*, 16, 2003.

[113] Li Y. Oord, A. v.d and O. Vinyals. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018.

[114] K. Sohn. Improved deep metric learning with multi-class n-pair loss objective. *Advances in neural information processing systems*, 29, 2016.

[115] A. Gupta D. Ghadiyaram X. Yan, I. Misra and D. Mahajan. Clusterfit: Improving generalization of visual representations, 2019.

[116] X. Chen and K. He. Exploring simple siamese representation learning, 2020.

[117] I. Misra and L. van der Maaten. Self-supervised learning of pretext-invariant representations, 2019.

[118] Y. Wu S. Xie K. He, H. Fan and R. Girshick. Momentum contrast for unsupervised visual representation learning, 2019.

[119] F. Altché C. Tallec P.H. Richemond E. Buchatskaya C. Doersch B.A. Pires Z.D. Guo M.G. Azar B. Piot K. Kavukcuoglu R. Munos J-B. Grill, F. Strub and M. Valko. Bootstrap your own latent: A new approach to self-supervised learning, 2020.

[120] M. Cuturi. Sinkhorn distances: Lightspeed computation of optimal transport. *Advances in Neural Information Processing Systems*, 26, 2013.

[121] How to use kaggle. https://www.kaggle.com/docs/notebooks, 2022.

[122] Tensorflow. https://www.tensorflow.org/. Last accessed: 22-01-23.

[123] L. Biewald. Experiment tracking with weights and biases. https://www.wandb.com/, 2020. Software available from wandb.com.

[124] The TensorFlow Team. Flowers. http://download.tensorflow.org/example_images/flower_photos.tgz, 2019.

[125] J. Mairal P. Goyal P. Bojanowski M. Caron, I. Misra and A. Joulin. Unsupervised learning of visual features by contrasting cluster assignments. https://github.com/facebookresearch/swav, 2021.

[126] SwAV-TF. A. thakur and s. paul. https://github.com/ayulockin/SwAV-TF, 2022.