National **T**echnical **U**niversity of **A**thens
School of **N**aval **A**rchitecture and **M**arine **E**ngineering
Division of **S**hip **D**esign and **M**aritime **T**ransport

DIPLOMA THESIS

An Application of Object Detection in ship navigation

Author: Ioannis Dimitrelos

Supervisors: Nikolaos P. Ventikos

ATHENS, JULY 2022

Acknowledgments

First and foremost, I would like to express my gratitude to my supervisor, Professor Nikolaos P. Ventikos, for his overall guidance and the trust he has shown in me, but also to the Ph.D. Candidate Panagiotis Sotiralis, as his precious help and support, was crucial in the development of this thesis.

Additionally, I would like to express my sincere gratitude to all the people who contributed to the completion of the present thesis with their experience and judgment, each from his field of expertise.

Also, I would like to thank the entire community of the School of Naval Architecture and Marine Engineering of the National Technical University of Athens, and of the Division of Ship Design and Maritime Transport, which provided me with all the knowledge and experience during my studies, for making the current work possible.

Last but not least, I would like to thank my parents and my brother for their continuous trust, for not losing faith, and for supporting me with all their strength. Special thanks to my precious friends for their understanding and support throughout the completion of my studies.

Contents

List of figures

Abbreviation List

ANN – Artificial Neural Network
ECDIS – Electronic Chart Display and Information System
BCE– Binary Cross-Entropy
BCEWithLogits – Binary Cross Entropy With Logits
BN – Batch Normalization
CNN – Convolutional Neural Network
CSP – Cross Stage Partial
Fast R-CNN – Fast Regional Convolutional Neural Networks
Faster R-CNN – Faster Regional Convolutional Neural Networks
FPN – Feature Pyramid Network
GPU – Graphics Processing Unit
LeakyReLU – Leaky Rectified Linear Unit Activation Function
mAP – mean Average Precision
MB – Megabyte
PAN – Path Aggregation Network
RCNN – Regional Convolutional Neural Networks
ReLU – Rectified Linear Unit Activation Function
RFCN – Region-based Fully Convolutional Network
ROI – Region of Interest
SPP – Spatial Pyramid Pooling
SPP-net – Spatial Pyramid Pooling Network
SSD – Single Shot MultiBox Detector
YOLO – You Only Look Once

Abstract

The present thesis is an attempt to light a spark in the field of object detection in ship navigation. This introduction is followed by an application based on an object detection model, named You Only Look Once (YOLO). Even though it is not the first time an object detection algorithm was applied to ship navigation problems, it is very fascinating to further research and experiment in this field by firstly testing an already existing algorithm and commenting on the results.

Many object detection algorithms have been proposed with approaches ranging from traditional to deep learning. However, the majority of them have limited applications in real-time applications as they are computationally intensive and have accuracy problems. Another challenge when dealing with ship navigation is the wide range of background sizes of the objects. To overcome these problems the most recent object detection algorithm was selected. In this thesis, the You Only Look Once version 5 (YOLOv5) model was used, which is created in 2020 and is fine-tuned with the more recent and best practices in object detection and also is constantly modified and readjusted to achieve better results. From the different models of YOLOv5, the smaller one was picked, because of its size and the comparatively good accuracy it performs.

After the model was chosen, a sufficient database for the model's training was created using images that contained the most common obstacles that a ship can face. These are other ships, buoys, humans on the surface, containers, and rocks. The images were categorized into 3 teams, ship, floating object, and rock, which are the classes of the problem. The images were then labeled in a way that the YOLO accepts as input while some of them were kept and created the validation dataset. With these data some scenarios were created, namely, images that contained only one class in them, images of ship at night, and images of at least 2 classes coexisting in the same image and their combination. The model was trained with these different datasets and the results were collected, compared, and analyzed. The model achieved a Precision of 84%, Recall of 74%, and mAP of 79% on average when trained with 400 images of all the classes combined for 300 epochs and batch size 8. The model was also tested in a real-time application using a video and it detected all of the ships in most cases with 33 frames per second reload time.

Περίληψη

Η παρούσα διπλωματική εργασία είναι μια προσπάθεια γίνει ένα πρώτο βήμα στον τομέα της ανίχνευσης αντικειμένων στη ναυσιπλοΐα πλοίων. Αυτή η εισαγωγή ακολουθείται από μια εφαρμογή που βασίζεται σε ένα μοντέλο ανίχνευσης αντικειμένων, που ονομάζεται YOLO. Παρόλο που δεν είναι η πρώτη φορά που εφαρμόζεται αλγόριθμος ανίχνευσης αντικειμένων σε προβλήματα πλοήγησης πλοίων, είναι πολύ συναρπαστική η περαιτέρω έρευνα και πειράματισμός σε αυτό το πεδίο, δοκιμάζοντας πρώτα έναν ήδη υπάρχοντα αλγόριθμο και σχολιάζοντας τα αποτελέσματα.

Πολλοί αλγόριθμοι ανίχνευσης αντικειμένων έχουν προταθεί με προσεγγίσεις που κυμαίνονται από την παραδοσιακή έως την deep learning. Ωστόσο, η πλειοψηφία τους έχει περιορισμένες εφαρμογές σε πραγματικό χρόνο, καθώς απαιτούν πολλούς υπολογισμούς και έχουν προβλήματα ακρίβειας. Μια άλλη πρόκληση όταν ασχολούμαστε με τη ναυσιπλοΐα πλοίων είναι το ευρύ φάσμα background και μεγεθών των πλοίων. Για να ξεπεραστούν αυτά τα προβλήματα επιλέχθηκε ο πιο πρόσφατος αλγόριθμος ανίχνευσης αντικειμένων. Σε αυτή τη διατριβή χρησιμοποιήθηκε το μοντέλο You Only Look Once έκδοση 5 (YOLOv5), το οποίο δημιουργήθηκε το 2020 και είναι τελειοποιημένο με τις πιο πρόσφατες και βέλτιστες πρακτικές στον εντοπισμό αντικειμένων και επίσης τροποποιείται και αναπροσαρμόζεται συνεχώς για να επιτυγχάνονται καλύτερα αποτελέσματα. Από τα διαφορετικά μοντέλα του YOLOv5 επιλέχθηκε το μικρότερο λόγω του μεγέθους του και της συγκριτικά καλής ακρίβειας που αποδίδει.

Μετά την επιλογή του μοντέλου, δημιουργήθηκε μια επαρκής βάση δεδομένων για την εκπαίδευση του μοντέλου χρησιμοποιώντας εικόνες που περιείχαν τα πιο κοινά εμπόδια που μπορεί να αντιμετωπίσει ένα πλοίο. Αυτά είναι άλλα πλοία, σημαδούρες, άνθρωποι στην επιφάνεια της θάλασσας, container και βράχοι. Οι εικόνες κατηγοριοποιήθηκαν σε 3 ομάδες, πλοίο, πλωτό αντικείμενο και βράχος, που είναι οι κατηγορίες του προβλήματος. Στη συνέχεια, οι εικόνες επισημάνθηκαν με τρόπο που το YOLO δέχεται ως είσοδο, ενώ ορισμένες από αυτές διατηρήθηκαν και δημιουργήθηκε με αυτές ένα σύνολο δεδομένων επικύρωσης. Με αυτά τα δεδομένα δημιουργήθηκαν κάποιες περιπτώσεις, δηλαδή εικόνες που περιείχαν μόνο μία κατηγορία, εικόνες πλοίου τη νύχτα και εικόνες τουλάχιστον 2 κατηγοριών που συνυπάρχουν στην ίδια εικόνα καθώς και ο συνδυασμός τους. Το μοντέλο εκπαιδεύτηκε με αυτά τα διαφορετικά σύνολα δεδομένων και τα αποτελέσματα συλλέχθηκαν, συγκρίθηκαν και αναλύθηκαν. Το μοντέλο YOLOv5s πέτυχε precision 84%, recall 74% και mAP 79% κατά μέσο όρο. Επιπλέον, τα δεδομένα χρησιμοποιήθηκαν ως σύνολο δεδομένων εκπαίδευσης για το YOLOv5m, που είναι το μεσαίου μεγέθους μοντέλο, και συγκρίθηκαν οι προβλέψεις καθώς και ο χρόνος εκπαίδευσης των 2 μοντέλων. Το μοντέλο δοκιμάστηκε τέλος σε μια εφαρμογή σε πραγματικό χρόνο χρησιμοποιώντας ένα βίντεο και ανίχνευσε όλα τα πλοία στις περισσότερες περιπτώσεις με χρόνο ανανέωσης 33 καρέ ανά δευτερόλεπτο.

ΕΦΑΡΜΟΓΕΣ OBJECT DETECTION

Η ανίχνευση αντικειμένων, object detection, έχει αρχίσει και εφαρμόζεται σε πολλούς τομείς, όπως την ανίχνευση και αναγνώριση προσώπων που υπάρχει στα σημερινά smartphone για να ξεκλειδώνεις το τηλέφωνο αλλά και στα μέσα μαζικής επικοινωνίας ενώ σε τράπεζες, και άλλα υψηλής προστασίας μέρη χρησιμοποιείται για να μπεις σε κάποια περιοχή. Ακόμη μία χρήση του είναι για την ασφάλεια και την παρακολούθηση, καθώς λόγω της αύξησης της εγκληματικότητας η επίβλεψη κάποιου απομακρυσμένου οικήματος ή την αναγνώριση των εισβολέων. Επιπλέον στην ρομποτική η γρήγορη απόκριση στα ερεθίσματα του περιβάλλοντος είναι πολύ σημαντική και μπορεί να επιτευχθεί με την γρήγορη και ακριβείς επεξεργασία των δεδομένων από μία κάμερα. Ένας ακόμη τομέας που μπορεί να χρησιμοποιηθεί η αναγνώριση

αντικειμένων είναι στην καταμέτρηση και την ιχνηλάτηση αντικειμένων σε παιχνίδια ποδοσφαίρου ή κινήσεις ανθρώπων σε κάμερες ή ακόμη και σε επίβλεψη της κίνησης στους δρόμους και άλλα. Τέλος μια πιο πρόσφατη αλλά και πολύ σημαντική εφαρμογή του είναι τα αυτόνομα αυτοκίνητα, όπου όπως γίνεται εμφανές το να γνωρίζει το αυτοκίνητο ή το πλοίο τα εμπόδια στον δρόμο του είναι απαραίτητο για να επιλεγεί η επόμενη κίνηση του.

YOLO ΕΦΑΡΜΟΓΗ ΣΤΗΝ ΝΑΥΤΙΛΙΑ

Στα πλαίσια της διπλωματικής έγινε προσπάθεια να εφαρμοστεί η λογική το object detection ώστε να αναγνωριστούν διάφορα εμπόδια στην πορεία του πλοίου. Για να επιτευχθεί αυτό χρησιμοποιήθηκε ο πιο πρόσφατος αλγόριθμος αναγνώρισης εμποδίων YOLO λόγω όχι μόνο της ταχύτητας του και της ακρίβειας του αλλά και της δυνατότητας για πραγματικού χρόνου εφαρμογές. Το YOLO διαφέρει από τις πιο παλιές μεθόδους της αναγνώρισης εμποδίων καθώς δεν επιλέγει μόνο συγκεκριμένες περιοχές της φωτογραφία, που έχουν μεγαλύτερη πιθανότητα να περιέχουν ένα αντικείμενο, για να κάνει την αναγνώριση αλλά κάνει την αναγνώριση σε όλη την εικόνα βάζοντας ορθογώνια με κέντρο κάποιο κομμάτι του πλέγματος της εικόνας, με διαστάσεις που να περιέχουν το αντικείμενο καθώς και σε ποια ομάδα ανήκει το αντικείμενο με κάποια πιθανότητα. Η ομάδα με την μεγαλύτερη πιθανότητα επιλέγεται σαν την ομάδα του αντικειμένου στο τέλος. Με αυτόν τον τρόπο μειώνεται το λάθος λόγο του background της εικόνας καθώς αυξάνεται η ταχύτητα και η ακρίβεια, όμως υπάρχει πιθανότητα να αναγνωρίσει περισσότερες φορές ένα αντικείμενο ενώ δεν μπορεί εύκολα να βρει πολλαπλά αντικείμενα στο ίδιο κομμάτι του πλέγματος.

Αυτά τα αρνητικά αφορούν κυρίως την αρχική έκδοση του YOLO καθώς στις επόμενες εκδόσεις γίνεται προσπάθεια να περιοριστούν αυτά τα αρνητικά και να γίνει καλύτερη η απόδοση του μοντέλου. Γι’ αυτό το λόγο στην διπλωματική γίνεται επιλογή της τελευταίας έκδοσης του YOLO που είναι η έκδοση 5, που έχει εφαρμόσει τις πιο σύγχρονες πρακτικές στον τομέα. Οι πρακτικές που έχουν χρησιμοποιηθεί σε κάθε έκδοση αναφέρονται στο κεφάλαιο YOLO Versions.

Για να γίνει η αναγνώριση εμποδίων πέραν του μοντέλου χρειάζεται και μια βάση δεδομένων. Αυτή η βάση δεδομένων περιέχει εικόνες που έχουν αντικείμενα που ενδιαφέρουν στην μελέτη. Τέτοια αντικείμενα είναι άλλα πλοία, βράχοι, σημαδούρες, άνθρωποι στην επιφάνεια της θάλασσας και container που είναι τα πιο συχνά εμπόδια στην πορεία ενός πλοίου. Οι φωτογραφίες αυτές για να μπορέσει να τις διαχειριστεί το πρόγραμμα πρέπει να επεξεργαστούν ανάλογα. Δηλαδή πρέπει με ένα πρόγραμμα να δημιουργηθούν αρχεία .txt που περιέχουν τις διαστάσεις των ορθογωνίων των αντικειμένων και την ομάδα στην οποία ανήκει το αντικείμενο. Αυτές οι φωτογραφίες μαζί με τα αρχεία τους χωρίζονται σε training και validation και τοποθετούνται σε φακέλους που δηλώνονται στο πρόγραμμα με ένα αρχείο YAML. Έχοντας αυτά τα δεδομένα το μοντέλο μπορεί να εκπαιδευτεί ώστε να μάθει τα χαρακτηριστικά του κάθε αντικειμένου στις φωτογραφίες και να ελεγχθεί το πόσο καλά ανταποκρίνεται σε φωτογραφίες που δεν τις έχει ξανά επεξεργαστεί.

ΣΕΝΑΡΙΑ ΚΑΙ ΑΠΟΤΕΛΕΣΜΑΤΑ

Στην παρούσα διπλωματική έγιναν διάφορα σενάρια για καταστάσεις και αντικείμενα που περιέχονται στις εικόνες και εξαχθήκαν κάποια αποτελέσματα. Τα σενάρια αυτά αφορούν τις ομάδες των αντικειμένων στις εικόνες, το πλήθος των φωτογραφιών και τις συνθήκες κάτω από τις οποίες έγινε η φωτογράφηση. Αρχικά υποτίθεται ότι μία μόνο ομάδα αντικειμένων, όπως πλοίο, βράχος, και πλεόμενα αντικείμενα, υπήρχε στην κάθε φωτογραφία. Γι’ αυτές τις περιπτώσεις έγινε χρήση 20, 50 και 100 φωτογραφιών και έγινε σύγκριση αποτελεσμάτων. Μετά έγινε εκπαίδευση του δικτύου με 50 φωτογραφίες από πλοία την νύχτα για να μπορεί το δίκτυο να λειτουργεί υπό όλες τις συνθήκες. Επόμενο σενάριο ήταν οι 100 φωτογραφίες από

όλες τις περιπτώσεις όπου υπήρχε μόνο μία ομάδα αντικειμένων σε κάθε μία καθώς και ο συνδυασμός όλων των περιπτώσεων μαζί με εικόνες που περιείχαν τουλάχιστον 2 ομάδες από αντικείμενα σε κάθε εικόνα, με συνολικό αριθμό 400 φωτογραφιών. Οι φωτογραφίες αυτές αντλήθηκαν από το ίντερνετ λόγω έλλειψης βάσης δεδομένων με διάφορες αναλύσεις καθώς το πρόγραμμα μπορεί να δεχτεί και διαφορετικές αναλύσεις. Τέλος η περίπτωση των 400 φωτογραφιών εκπαιδεύτηκε και στον μεγαλύτερο νευρωνικό, το medium, και έγινε μία εφαρμογή του μοντέλου σε ένα βίντεο σε πραγματικό χρόνο.

Το μοντέλο YOLOv5s για τις 400 φωτογραφίες πέτυχε precision 84%, recall 74% και mAP 79% κατά μέσο όρο, ενώ μπόρεσε στις περισσότερες περιπτώσεις να προβλέψει όλα τα αντικείμενα με καλή σιγουριά και με χρόνο ανανέωσης 33 καρέ το δευτερόλεπτο.

ΣΥΜΠΕΡΑΣΜΑΤΑ

Ο αριθμός των εικόνων παίζει σημαντικό ρόλο για το μοντέλο. Καθώς το σύνολο δεδομένων εκπαίδευσης μεγαλώνει, οι μετρήσεις, η ακρίβεια, η ανάκληση και το mAP του μοντέλου, φτάνουν σε υψηλότερες τιμές για τις ίδιες εποχές και η διακύμανση των τιμών με κάθε εποχή είναι μικρότερη. Επιπλέον, οι υψηλότερες τιμές για τις μετρήσεις επιτυγχάνονται νωρίτερα καθώς επεκτείνεται το σετ εκπαίδευσης. Παρατηρείται επίσης ότι όταν το μοντέλο εκπαιδεύεται με διαφορετικές εικόνες, αλλά ίδιες σε αριθμό, τότε τα αποτελέσματα για τις μετρήσεις είναι διαφορετικά. Αυτό σημαίνει ότι το μοντέλο εξαρτάται σε μεγάλο βαθμό από την ποιότητα, το φόντο των εικόνων και τις συνθήκες κάτω από τις οποίες τραβήχτηκαν οι εικόνες.

Ο χρόνος που δαπανάται για την εκπαίδευση αυξάνεται επίσης με τον αριθμό των εικόνων στο σύνολο δεδομένων εκπαίδευσης κατά σημαντικό ποσό. Ένα μοντέλο εκπαιδευμένο για 300 εποχές με ένα σύνολο δεδομένων 100 εικόνων χρειάζεται 1 ώρα για την εκπαίδευσή του, ενώ για τον ίδιο αριθμό εποχών 400 εικόνες χρειάζονται 4 ώρες. Επίσης, ο χρόνος που δαπανάται για την εκπαίδευση του μοντέλου αυξάνεται εκθετικά καθώς το μοντέλο γίνεται πιο μεγάλο, με 400 εικόνες να λαμβάνουν 4 ώρες στο μικρό δίκτυο και 11 ώρες στο μεσαίο δίκτυο εκπαιδευμένες για τις ίδιες 300 εποχές.

Η εμπιστοσύνη του δικτύου στην ανίχνευση αντικειμένων σε εικόνες που δεν έχει δει ποτέ ενισχύεται καθώς το σετ εκπαίδευσης παρέχεται με μεγαλύτερη ποικιλία εικόνων και εικόνων που είναι σχετικές με αυτές που πρέπει να ανιχνεύει το μοντέλο. Τέλος, το μέγεθος του δικτύου είναι επίσης πολύ σημαντικό για τα αποτελέσματα, καθώς όσο περισσότερο μεγαλώνει το δίκτυο τόσο πιο εύκολο είναι να βρείτε τα σωστά αντικείμενα και να έχετε μεγαλύτερη ακρίβεια.

ΠΡΟΤΑΣΕΙΣ ΓΙΑ ΜΕΛΛΟΝΤΙΚΗ ΕΡΕΥΝΑ

Παρατηρήθηκε ότι ο αριθμός των εικόνων στις οποίες εκπαιδεύτηκε το μοντέλο ήταν πολύ ανεπαρκής για να μπορέσει το μοντέλο να λειτουργήσει σωστά, καθώς απαιτούνται τουλάχιστον 1500 εικόνες ανά κατηγορία. Μελλοντικά, το μοντέλο θα πρέπει να ενισχυθεί με περισσότερες εικόνες για κάθε τάξη για να ληφθούν πιο ακριβή αποτελέσματα.

Επιπλέον, οι εικόνες που χρησιμοποιούνται θα πρέπει να είναι πιο αντιπροσωπευτικές της περίπτωσης που θεωρείται, επικεντρώνοντας κυρίως στην προσπάθεια εύρεσης εικόνων που έχουν την ίδια γωνία και ύψος μιας κάμερας τοποθετημένης στην γέφυρα του πλοίου και επίσης της ίδιας ανάλυσης που απαιτεί το μοντέλο . Το σύνολο δεδομένων εκπαίδευσης θα πρέπει επίσης να δοκιμαστεί στις μεγαλύτερες εκδόσεις του YOLOv5 για να έχουμε καλύτερα

αποτελέσματα και οι παράμετροι του μοντέλου θα πρέπει επίσης να αλλάξουν και να γίνει έρευνα για το πού επιτυγχάνονται τα καλύτερα αποτελέσματα.

Τέλος, μετά την εκπαίδευση του μοντέλου και την επίτευξη καλύτερων αποτελεσμάτων, το επόμενο βήμα θα πρέπει να είναι η δημιουργία μιας τεχνητής νοημοσύνης που συνδυάζοντας τα αποτελέσματα με ένα άλλο σύστημα του πλοίου, που ονομάζεται Electronic Chart Display and Information System (ECDIS) να μπορεί να βρίσκει την ταχύτητα και την πορεία των άλλων αντικειμένων. Σύμφωνα με αυτά τα αποτελέσματα να προτείνει μία πορεία για το πλοίο, ώστε να αποφύγει τα αντικείμενα στο δρόμο του. Στο τελικό στάδιο θα πρέπει αυτή η τεχνητή νοημοσύνη να μπορεί να αλλάζει την πορεία και την ταχύτητα του πλοίου για να αποφευχθεί η σύγκρουση, ενώ το σύστημα θα πρέπει να δοκιμαστεί πάνω σε μοντέλο σε κάποια δεξαμενή.

1 Literature Review

Neural networks and especially Convolutional Neural Networks have made significant development over the last decades and are finding more and more applications in various industries. Specifically, object detection is a field that the tools for it to develop were found only recently. However its evolution was massive and so in only 20 years dozens of networks were created with each one being better than its predecessor solving some of their problems by applying new ways to manage the images or the feature maps, or even totally different ways to make the detection. YOLO is a good example, as in only 5 years there have been 5 versions of it that each applied the most recent techniques known until then and so having better results. YOLO applications extend to every industry that has an interest in computer vision, detection, tracking, counting, and so on. That's why it is widely applied in ship navigation mainly in autonomous ships, where the detection of obstacles in the ship's path is a very important feature, and in sea surveillance.

Yang-Lang Chang and his team developed a ship detection algorithm based on YOLOv2 for SAR imagery in 2019 (Chang et al., 2019). By reducing the number of layers, and so creating a new model called YOLOv2-reduced, they managed to have the same accuracy, near 90%, of YOLOv2 but with reduced computational time. In the same period an article referring to ship detection under different weather conditions based on a deep neural network by Xin Nie (Nie et al., 2019). He and his team proposed to enlarge a dataset that contains only clear images in normal weather conditions, with synthetically generated images that match different weather conditions. They then trained the YOLOv3 network with this dataset and resulted in a decrease in accuracy and recall of 6% and 10% respectively, compared to the clear weather dataset, especially in smaller boats. In another article, Tianwen Zhang and his team proposed a high-speed SAR ship detection approach by improved YOLOv3 in 2019 (Zhang et al., 2019). They experimented on a public SAR ship detection dataset and the results indicated an increase in the detection speed compared to the YOLOv3, Faster R-CNN, and SSD methods while maintaining the same accuracy. In 2019 again Ruidong Zheng combined the YOLO algorithm and Automatic Identification System (AIS) to assist vessels in obstacle avoidance (Zheng et al., 2019). This system processes the image and video information that are acquired by cameras mounted on the ship and in combination with the AIS system of the ship provides a more accurate global view. The experiments showed that the system can identify the ships and visualize the AIS information, but didn't report numbers for the metrics.

Another application of YOLOv3 in ship navigation was done by Xinqiang Chen that used YOLOv3 to detect small, medium, and large ships in 2019 (Chen et al., 2019). The detection was done in different port navigation scenes, namely low traffic, foggy environment, high traffic, and small image scale. The results returned an average value of 84% and 92% for Recall and Precision respectively. In the same year, a region of interest extraction algorithm based on YOLOv3 was proposed by Li Tianwei (Li et al., 2019). In this method, different quality naval images were generated using image degradation algorithm, the network was retrained utilizing migration learning that achieves better accuracy and detection rate and lastly, the dimensions of the output tensors were optimized. These innovations resulted in a 4.25% increase in the detection rate, improving the effectiveness of the algorithm. An improved version of YOLOv3 for ship detection was proposed by Haiying Cui in 2019 (Cui et al., 2019). The main improvements were in the dimensions of Clusters, some network improvement, and applying the Squeeze-and-Excitation module. The experiments had as a result an mAP of 91%, increased by 4% from the YOLOv3 algorithm. Except for the last article an improved YOLOv3 algorithm was also developed by Yuchao Wang and Xiangyun Ning that borrowed the CFE module from the CFE network and changed the 1x1 convolutional of YOLOv3, while improving the loss function and augmenting data for small ships (Wang et al., 2019). The

module achieved an accuracy of 74.8%, increased by 3.6% from the YOLOv3, but with fps 29.8, decreased by 3.6 compared with YOLOv3. The last application of YOLO in shipping for 2019 was Jie Yang and his team which developed a tracking algorithm for unmanned surface vehicles using YOLOv3 to extract high-performance object detection (Yang et al., 2019). Then a data association method is implemented combining the estimation of motion state through Kalman filter and the appearance feature. The results of the detectors that used YOLOv3 returned an mAP of 80% while the tracking system seemed to better complete the detection compared with the SORT tracking algorithm.

In 2020 Zhenfeng Shao and his team worked on making YOLOv2 more efficient in near coastline problems in the sea (Shao et al., 2020). They developed a novel saliency-aware CNN based on the YOLOv2 model that extracts coastline feature maps and inputs them into a CNN in increase to reduce the program's ability to differentiate nearshore buildings and small ships. They compared their model with the most used ones, Fast R-CNN, Faster R-CNN, SSD, and YOLOv2 for different types of ships, and resulted in an mAP of 87%, 14% higher than the YOLOv2. In the same year, an article regarding the discrimination between icebergs and ships using YOLOv3 was published by Fredrik Seerup Hass (Hass & Jokar Arsanjani, 2020). In this article, the author trained the YOLOv3 model with iceberg and ship images from a Synthetic aperture radar (SAR) for different epochs and resulted in precision, recall, and mAP scores with maximum values of 65%, 60%, and 55.7% respectively. Zhelin Li published an article about a Lightweight Ship Detection Method based on the YOLOv3 and DenseNet, in 2020 called LSDM (Li et al., 2020). The backbone of this method is improved by using dense connections inspired by the DenseNet, while the feature pyramid networks are improved using spatial separation convolution instead of regular convolution. These innovations affected positively the overall performance with the recall being 95%, the precision 85%, and the mAP 94%, almost the same as the YOLOv3 but with 66% fewer parameters.

In 2021 the article of Yang Jie was published and involved ship detections and tracking in inland waterways with the use of YOLOv3 but with some improvements (Jie et al., 2021). They added the Kmeans clustering algorithm to initialize the anchor boxes, modified the output classifier to a single softmax classifier, and changed the Non-Max Suppression to a Soft Non-Max Suppression. The mAP of the model was 95.5% with an increase of 5% in comparison to the YOLOv3 algorithm. Dehai Chen and his team worked, in 2021, on a ship detection algorithm based on the improved YOLOv3 algorithm, containing the attention mechanism that was embedded in Darknet-53 and a new feature enhancement algorithm for having more semantic information on low-level features (Chen et al., 2021). The results of this research were 97% for both the precision and the recall and a mAP of 99%, increased by 3% compared to YOLOv3. Meanwhile, in the same period of time, a similar article was published researching the recognition and tracking of water surface targets using YOLOv3 improved by the Inception module and the KCF algorithm to reduce the loss of blocked targets (Ma et al., 2021). This research was conducted by Zhongli Ma and his team and resulted in a mAP of 88%.

The one to apply the YOLOv4 algorithm in ship detection was Xu Han in 2021 who replaced the backbone with a network called RCSPDarknet to improve precision, designed an amplified receptive field module named DSPP to reduce the loss of small ships, and used the attention mechanism and Resnet's shortcut idea to create a new feature pyramid structure (Han et al., 2021). All these improved $mAP_{0.5:0.95}$ by 1%, reaching 57.7% and increasing FPS to 69.4 from 56.1 with 23% reduced parameters compared to YOLOv4. In the same year, a sea surface object detection algorithm based on YOLOv4 was created by Tao Liu (Liu et al., 2021). They added a Reverse Depthwise Separable Convolution in the backbone of the algorithm so as to detect unmanned surface vehicles resulting in 40% decreased weights, 20% increased detection speed and 2% increased mAP compared to the YOLOv4. Junchi Zhou in 2021 worked on an

improved YOLOv5, in which the initial frame at the target is re-clustered by K-means, the receptive field area is expanded and the loss function is optimized (Chen et al., 2021). The results from these changes show a precision of 98% and a recall of 96.2% while the mAP increased by 4.4% with a value of 98.6% compared to the YOLOv5.

One year after the articles above, in 2022, an improved version of YOLOv3 was created by Lena Chang that appropriate input image size, fewer convolution filters, detection scales, and modifications of the spatial pyramid pooling to reduce complexity and improve performance (Chang et al., 2022). The model was trained with visible and infrared images and resulted in a 48% reduction in the billion floating point operations while increased the mAP and FPS by 2% and 8%, respectively, reaching values of 93% and 104.7. A ship detection and classification algorithm based on YOLOv4 was also designed by Weina Zhou and Lu Liu in 2022 (Zhou & Lu, 2022). The model integrated a Multi-layer Feature Fusion and a Multi-layer Receptive Field Block module into the neck of YOLOv4 to reduce feature information loss. The impact on the main algorithm of YOLO was noticeable with an increase of 12% in mAP, with a value of 76.4%.

In 2022 also, with their article "An Improved YOLO v4 Algorithm-based Object Detection Method for Maritime Vessels" Guowen He and his team used a k-means algorithm to increase clustering at the input side of image data (He et al., 2022). This change led to mAP, precision, and recall values of 86%, 86.4%, and 84% respectively, which is a 3% increase compared to YOLOv4. In the same year, Zakria published his modified version of YOLOv4 (Zakria et al., 2022). A classification setting of the nonmax suppression threshold and two allocation schemes for the problem of frame anchor allocation in the base algorithm were proposed in order to increase the accuracy without affecting the speed. The mAP for the 3 innovations either decreased or stayed at the same level as the YOLOv4 with the highest reaching 74.22%, namely decreasing the mAP by 1%. A new model that modified the YOLOv5 model by replacing the CSP-DarkNet with CSP-DenseNet to increase the accuracy of target detection and classification was proposed by Xuan Zhang in 2022 (Zhang et al., 2022). Experiments showed that it reached 71.6% on mAP in comparison to 62.2% of the YOLOv5.

Jia-Chun Zheng published a model based on YOLOv5 network in 2022 for fast ship detection (Zheng et al., 2022). The team proposed an optimization of the anchor boxes according to the ship target characteristics, mapped the k-means clustering algorithm to select more appropriate anchor boxes, and also used the scaling factor γ for the batch normalization, reaching an 86.5% mAP, increased by 2% of the base model YOLOv5. That year an article by Emmanuel Vasilopoulos and his team regarding autonomous object detection algorithms in the maritime environment using a UAV platform was published (Vasilopoulos et al., 2022). In the context of the research an embedded system that employed machine learning algorithms, specifically the YOLOv5 algorithm, was created allowing a UAV to detect objects in the water. The results for the precision were 87%, the recall 62%, and the mAP 67%. In 2022 also, an article about a Complete YOLO-based ship detection method for Thermal Infrared Remote Sensing Images under Complex Backgrounds was developed by Liyuan Li and his team (Li et al., 2022). The dataset was developed using a thermal imaging system, they were preprocessed and input in the YOLOv5s, which was improved by adding Dilated Convolutional, depthwise convolution, and SELayer modules. The results showed an mAP of value 98.7%, which is 9% higher than the YOLOv5s model but also increased the number of layers, from 283 in YOLOv5s to 390 in the proposed method. Lastly, Jun-Hwa Kim and his team in 2022 used a ship detection and classification algorithm based on YOLOv5 (Kim et al., 2022). They implemented the mix-up technique in addition to the basic augmentation of YOLOv5 and corrected the Singapore Maritime Dataset. They then used the dataset to train the YOLOv5 which resulted in 89.8% mAP, in comparison to the 77.2% of the SMD before the correction.

2 Neural Networks

Since the late 1980s research activity in neural networks has seen a noticeable increase in interest. Even though most of the research has focused on developing new algorithms, there has been an increasing urge in the direction of applying neural networks to real-world problems. Through their use, it became clear that they are extremely good in pattern recognition and data processing and so very helpful to areas that need these features, like scientific instrumentation.

Many types of neural networks have been developed, but in this thesis the main focus will be on Convolutional Neural Network (CNN) which is the one that the dataset was trained on. But before going in depth on those networks specifically there will be a brief introduction to the general idea behind Artificial Neural Networks (ANN) and their history.

2.1 Overview of neural networks

Neural Networks represent a computational example in which a set of examples is used in order to find a solution to a problem. The idea of neural networks comes mainly from studies of the information processing mechanism in biological neural networks, especially in the human brain. In fact, in the early years, the main focus of the research was on understanding the function of the biological nervous system of the human brain.

A neural network in general is regarded as a non-linear mathematical function that is used to transform a set of input into a set of output. This transformation is controlled by a set of parameters that are called weights whose values are defined by a set of examples. The act of determining these weights is called learning or training and it generally requires much computational power. When the weights have been fixed then the neural can process new data, out of the training data quickly (Bishop, 2006).

Neural networks can process data with high speed, find the solution to a problem with only a number of input values and also give the ability to work with incomplete data and produce an output. Although the loss of performance depends on the importance of the missing information.

The disadvantages of neural networks originate from the need to produce a suitable set of examples to train on and the potential problems that may arise if the network is needed to locate the relation between data that are significantly different from the ones that it is trained on. Moreover, neural networks are very hardware depended as they require processors with parallel processing power and they also produce a solution without the user knowing why or how. Generally speaking, neural networks are suitable for solving problems that have the following characteristics (Bishop, 1994):

i. There are abundant data to train on,
ii. It is difficult to create an adequate simple solution based on models for the problem,
iii. The data must be processed at high speed, as there is a great number of data to be processed and there is a need for real time application,
iv. The method must be steady even if the input data have a moderate noise.

2.1.1 Biological neural networks

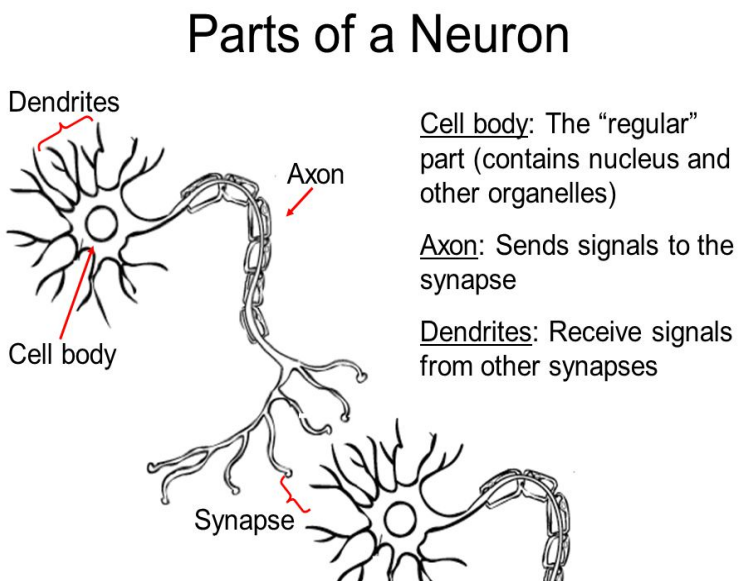

Figure 2.1: Schematic of two biological neuron.

The human brain is one of the most complicated structures of the human body but it is also the most exciting task that scientists faced. Biological neural networks are of high interest for humans, which derives from the desire to build better pattern recognition and information processing systems. For completeness and better understanding of the way neural networks work in this thesis, there will be a given a simplified review of biological neural networks.

The human brain consists of 10^{11} active cells called neurons. Their most common features are shown in Figure 2.1. Every neuron consists of a cell body or soma that contains the cell nucleus. The branching tree of dendrites is associated with the cell body and acts as an input to the neuron as they receive signals from other neurons. From the soma, a single long fiber extends called axon, which branches out to threads connecting to many other neurons at the synapses. For the transition to take place a complex chemical process must be done, where substances are released from one neuron to the receiving one. This way the electrical potential is either lowered or increased in the receiving neuron. If that potential is higher than a threshold then the neuron is said to be fired and an electric impulse is triggered (Yegnanarayana, 1994). When this happens the signal of the fixed strength is sent down the axon and through the synapses passed to the dendrites of the next neuron. From there the neuron computes a weighted sum of the inputs from other neurons and if the sum exceeds a threshold then the neuron fires.

The activity of a given synapse is dependent on the rate of the signals arriving at it. A synapse that continuously triggers the activation of its presynaptic neuron will grow in strength while others will gradually weaken. Thus the strength of the synaptic connection will repeatedly get modified. This mechanism of neural connectivity plays a significant role in the process of learning.

In the human brain, each neuron is receiving signals from approximately 10^4 synapses. Also, every neuron passes its signal to hundreds of other neurons so the total number of connections between neurons reaches 10^{15}, the majority of which are developed during the first few months after birth.

Each neuron alone is generally slow in processing information but when they are connected as the information can be processed parallel in many neurons the total computational power can be equal to or even more than the newest computers. It also leads to a high loss tolerance as the fact that every day many neurons die doesn't affect the performance of the overall system. The overall simplified picture of the biological neural networks can be a good starting point in understanding the way neural networks are formed and the way they work.

2.1.2 History of neural computing

The history of neural networks goes back to 1943 when Warren McCulloch and Walter Pitts (McCulloch & Pitts, 1943) created a computational model for neural networks based on algorithms called threshold logic. They showed that networks of neural networks are capable of universal computation, paving the way for research that focused on the application of neural networks to artificial intelligence.

Next up D.O.Hebb in 1949 in his book (Hebb, 1949) proposed a learning hypothesis based on the mechanism of neural plasticity, namely the ability of neural networks in the human brain to change through growth and reorganization. He suggested that learning happens through modification of the strengths of the synaptic connection between 2 neurons. He also said that when 2 neurons fire together then the synapse between them should be strengthened and accordingly if they don't fire the synapse should be weakened. This learning hypothesis became known as Hebbian learning. Different researchers then started to apply these ideas and created neural network computational machines. Farley and Clark (Farley & Clark, 1954) in 1954 were the first to use computational machines. Other neural network computational machines were created by Rochester, Holland, and Duda (1956) (Rochester et al., 1956).

In 1958 Rosenblatt was the one to develop the first hardware neural network system (Rosenblatt, 1958). They named it the perceptron and was based on McCulloch-Pitts neuron models. Its external input was an array of photoreceptors and the synaptic connections were provided by potentiometers. Adjustments in the potentiometers were made using the perceptron learning algorithm (Rosenblatt, 1961). The perceptron could learn to see the difference between characters and shapes which were provided as images in the input. Rosenblatt also resented the result that if a problem is soluble from the perceptron then the perceptron algorithm will solve it in a finite number of steps.

Research, although, decreased in the 1960s after the discovery of 2 issues with computational machines by Minsky and Papert in 1969 (Minsky & Papert, 1969). The first was that the perceptron was incapable of computing the exclusive-or circuit and the second was that current computers couldn't handle the work required by the larger neural networks as they lacked computational power. Therefore the research reduced in speed until greater processing power in computers was achieved. In 1970 the field of neural networks was abandoned with only a few researchers still active in the field.

A "revival" in this field of science started in the early 1980s and was led by J. J. Hopfield with his work (Hopfield, 1982), (Hopfield, 1984), who pointed out the relation between neural network models and some systems known as spin glasses. The next development was the creation of new algorithms based on error backpropagation in 1986 by Rummelhart (Rumelhart et al., 1986). Backpropagation describes a method in which errors are processed at the output and then go through the system's layers for learning and training. The fact that there was a big availability of cheap and high computational power computers, combined with the work of Rumelhart and the fact that Artificial Intelligent was not as good as anticipated led to a burst of interest in neural networks.

In 1992 the max-pooling layer was introduced to help in 3D object recognition through the building of the cresceptron by John (Juyang) Weng, Narendra Ahuja, and Thomas S. Huang (Weng et al., 1992) (Weng et al., 1993). In the same year, Schmidhuber used a multi-level hierarchy of networks (Schmidhuber, 1992). He pre-trained one level at a time by unsupervised learning and then used backpropagation to evaluate. This way he solved the vanishing gradient problem, in which the error in a neural network shrinks as it goes from one layer to another so it reaches a vanishingly small value which prevents the weight from changing in the next layer.

In the next years, the topic of neural networks has attracted significant attention and thus there has been a great deal of research and progress. Especially the topic of pattern recognition, with the development of CNNs (LeCunn et al., 1998) that started to attract more and more interest over the years. From then on the main focus moved to CNN and especially object detection algorithms and their applications in real-time problems, with the development of different methods that will be analyzed in the next sections.

Nowadays neural networks have been widely applied in many aspects of life problems. Some of their applications are in healthcare, as CNN are used for analyzing X-ray photos ultrasounds, face recognition in mobile smartphones, handwriting analysis, signature verification, weather forecasting, autonomous cars or ships that don't need human assistance and the list goes on.

2.2 A single neuron model

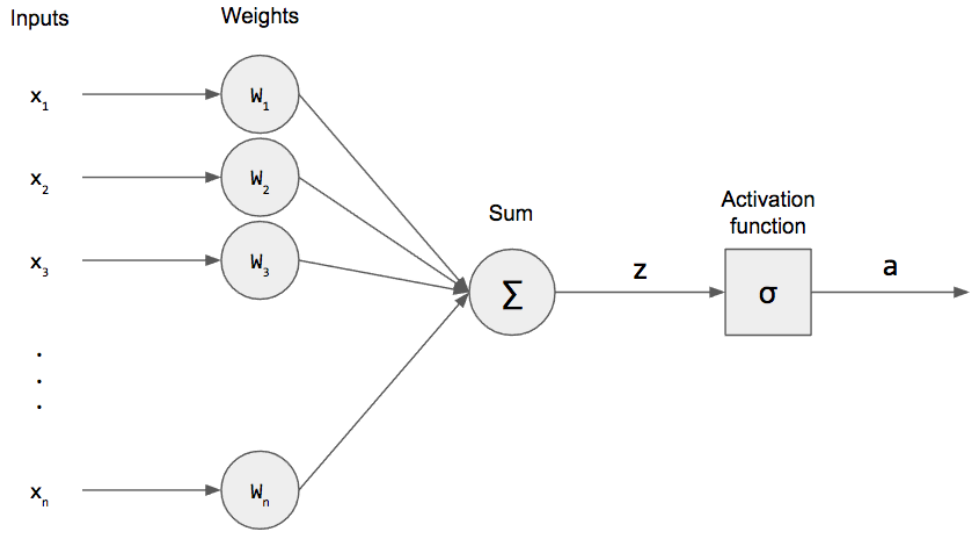

Figure 2.2: McCulloch-Pitts model.

In 1943 McCulloch and Pitts (McCulloch & Pitts, 1943) introduced the first neural network that consisted of one neuron. A schematic of the network is shown in Figure 2.2. This model transforms a number of input values $x_1, \dots, x_n$ into an output variable z. The input x_i is firstly multiplied with the corresponding parameter w_i, which is called weight (the equivalent to the synaptic strength in the biological neural networks that were covered above). All the weighted inputs are then appended and so the output is produced (Equation (2.1)) :

$$z = \sum_{i=1}^{n} w_i x_i + b \tag{2.1}$$

, where b is the bias and is equivalent to the threshold in the biological neural networks.

The output a of the network, which can be compared slightly with the average firing rate of the neuron in the biological counterpart, is calculated by passing it to the non-linear function σ( ), which is called activation function, as in the following Equation (2.2):

$$a = \sigma(z) \tag{2.2}$$

This simple model forms the base that every network works on. Just by linking many of these simple processing units together, more complicated networks can be formed. Now that both the biological and the artificial neural networks have been mentioned, a comparison between the elements of the two can be done with the help of the following Figure 2.3:

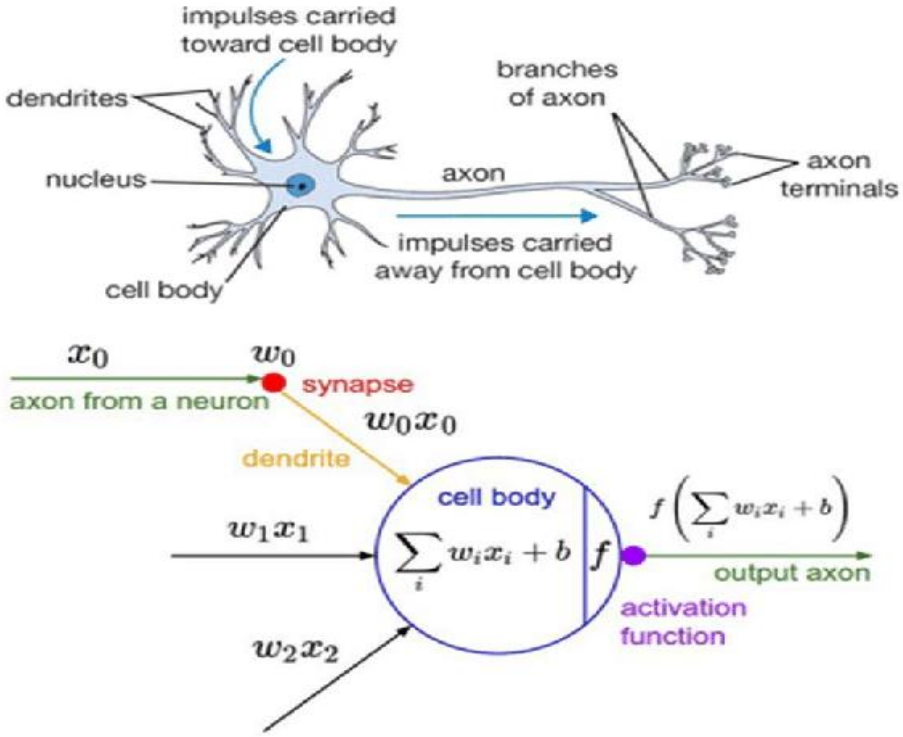

Figure 2.3: Comparison between biological and artificial neurons

The two neurons, both biological and artificial, have many similarities. The axon of the previous neuron and the dendrites are the arrows that connect the different artificial neurons. The synapses is where the multiplication of the input and the corresponding weight is done. The cell body is the artificial neuron where all the calculations are done and where the output of the summarization is altered with the use of the activation function. Finally, the output axon is the arrow that connects the artificial neuron with the next one.

2.3 The Multilayered Perceptron

Even though neurons on their own don't have much computational power, when combined they can create a network of neurons which is called a neural network. Neurons that are in the same column create a layer. A neural network has many layers with each consisting of many neurons. The Figure 2.4 depicts a network with 3 separate layers and 2 layers of weights. Neurons that are in the first layer create the input layer where the input values are inserted. The ones that are in the middle layer create the hidden layer, which is called this way because they are not exposed directly to the input of the network and their activation values can't be accessed from outside of the network. The simplest network consists of only one neuron in the hidden layer that outputs the final value. Although with the use of the continuously developing technology and computational power of computers it is feasible to create very deep

neural networks, namely to have many hidden layers, things unimaginable for researchers in the previous decades. Lastly, the output layer is the one that is responsible for outputting the final value or the vector of values, according to the format required for the problem.

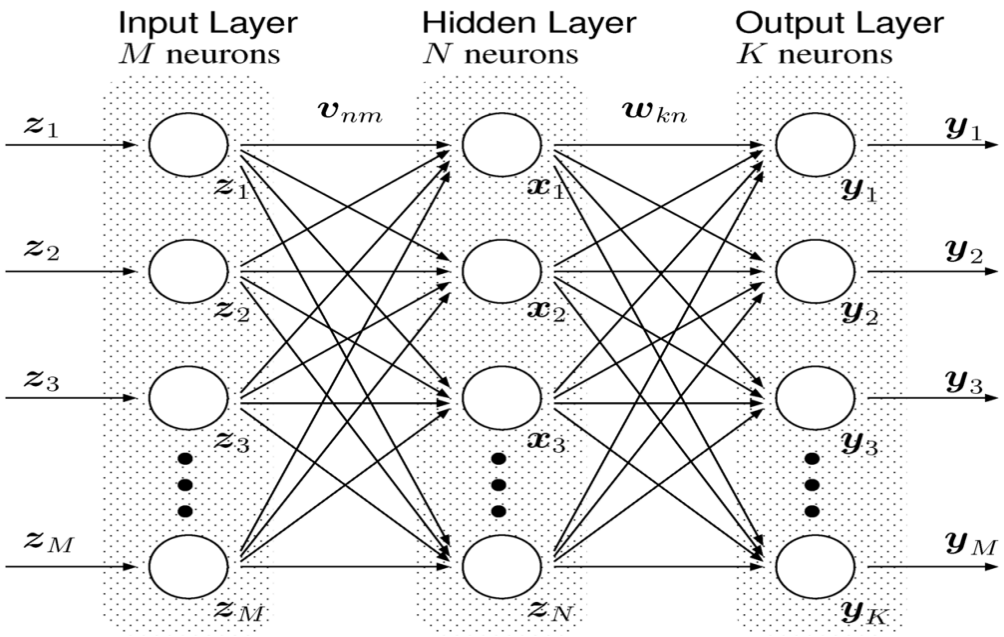

Figure 2.4: A multilayer perceptron with 3 layers of neurons, 2 layers of weights.

The hidden layer's values are denoted as x_n and are given by the following Equation (2.3) (Bishop, 2006)

$$x_n = g\left(\sum_{m=0}^{M} v_{nm} z_m\right) \tag{2.3}$$

, where v_{nm} are the weights connecting the m neuron in the input layer with the n neuron in the hidden layer and g is the activation function between the input and hidden layer. Note that instead of putting the bias we included it in the summarization as a special weight from an input of $z_0=1$. Although that is not the output of our network. To reach the output the z neurons must first be transformed by the activation function between the hidden layer and the output layer with a similar Equation (2.4)

$$y_k = g'\left(\sum_{n=0}^{N} w_{kn} x_n\right) \tag{2.4}$$

, where w_{kn} are the weights that connect the n neuron of the hidden layer with the k neuron in the output layer. Again the bias is included as a special weight from an input of $x_0=1$. Lastly combining these previous 2 equations the final output of our neural network is created, which is represented in the Equation (2.5) below

$$y_k = g'\left(\sum_{n=0}^{N} w_{kn} g\left(\sum_{m=0}^{M} v_{nm} z_m\right)\right) \tag{2.5}$$

Each of the components in Equation (2.5) corresponds to an element in the Figure 2.4. Something to note here is that activation functions g and g' don't need to be the same.

In the literature, there are 2 ways to count the number of layers in a neural network, and they are both commonly used. The first way is by counting all the layers that the neural network has, except for the input layer. So in the above example, in Figure 2.4, the total layers are 2, the hidden and the output layer, which is equal to the layers of the weights. In the other way, the layers of the network are equivalent to the total number of layers in the network with the input layer included. So in the paradigm, there are 3 layers in the network and 2 layers of weights.

2.3.1 Activation Functions

Activation functions, as discussed, are the functions that alter the output of the neuron. They are just a simple mapping of the weighted sum of the inputs in the neurons. It is called this way because it governs the threshold at which the neuron will activate and the strength of the output (Sharma et al., 2020). Activation functions are divided into 2 types:

1. Linear Activation Functions
2. Non-Linear Activation Functions

The Linear Activation Function is the simple and well-known function of a line. As this function doesn't help with the complexity or the various parameters of usual data inserted into the neural networks there won't be an emphasis on that function. A typical figure for that function is illustrated in Figure 2.5.

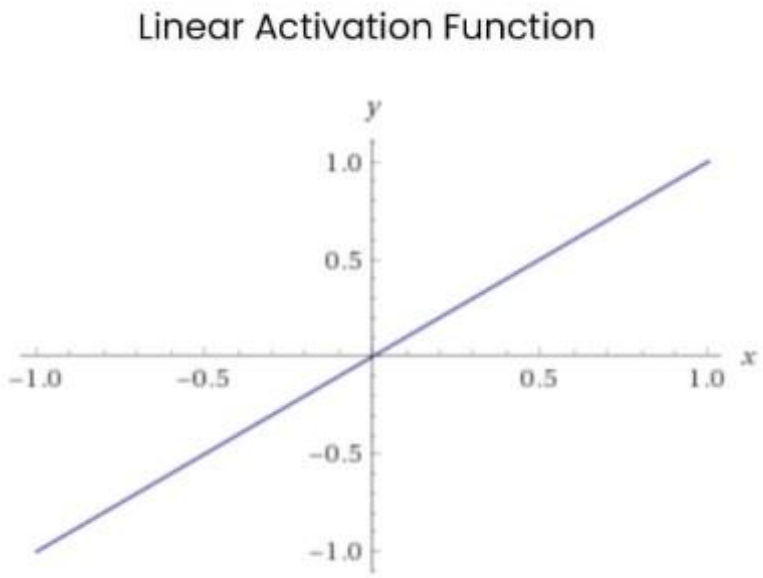

Figure 2.5: Linear Activation Function

Even though Linear Activation Function is useful in some problems, when generalization and adaptation of the model through different input values are needed then a Non-Linear Activation Function is chosen. A Non-Linear Activation Function is used to give non-linearity to the network. There are many functions of this type, although for simplicity the most common will only be mentioned.

The Sigmoid Activation Function is depicted in the Figure 2.6(a) and has an S-shape. It is useful because it restricts values between 0 and 1. Therefore, it is used especially in models that have to predict the probability of the output, as the probability is also limited between 0 and 1.

The Tanh or hyperbolic tangent Activation Function is illustrated in the Figure 2.6(b) and has also an S-shape as the Sigmoid. Although the difference is that the Tanh function's values range from -1 to 1, which means that the negative values will be mapped strongly negative and respectively the positive will be mapped strongly positive. For this reason, the Tanh function is mainly used in classification between 2 classes.

The Rectified Linear Unit Activation Function (ReLU) is shown in the Figure 2.6(c). It is the most commonly used function right now as it is used widely in convolutional neural networks and deep learning. All the values that are less than 0 are mapped 0 and the positive are mapped, as in the linear function, without some difference. Although it is not the best go-to function as it doesn't map the negative input values, which decreases the ability of the network to train from the data properly.

The Leaky Rectified Linear Unit Activation Function (LeakyReLU) function, (Figure 2.6(d)) is an attempt to solve the problem of the ReLU not mapping negative values. For this reason, a "leak" is put and the function doesn't nullify the negative values but maps them with a linear function ax with a not being 1, as then it turns into the already discussed linear function.

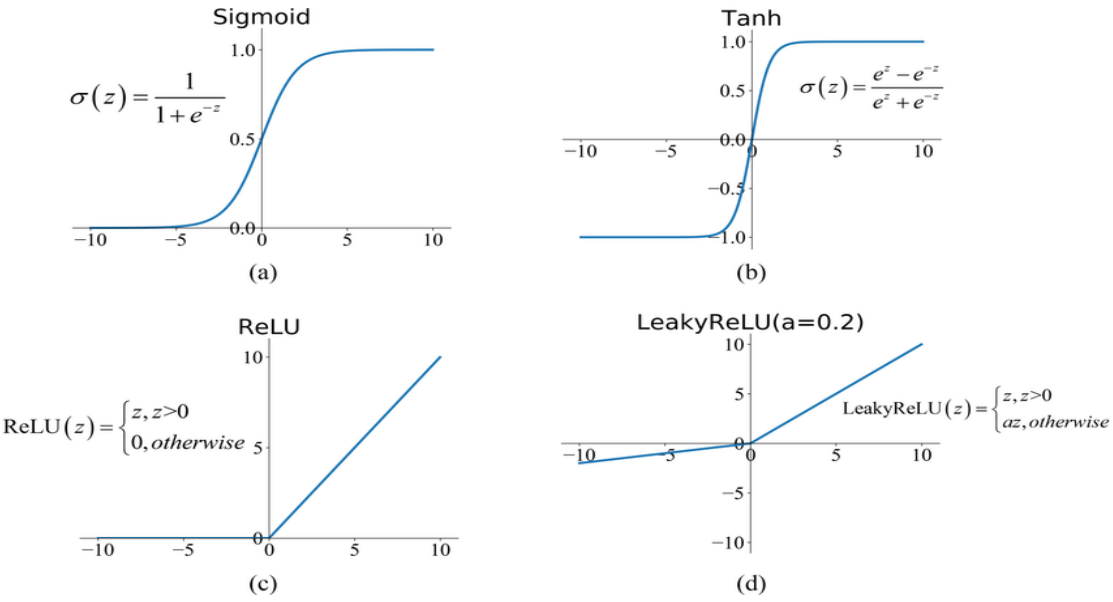

Figure 2.6: Typical Activation Functions (a) Sigmoid, (b) Tanh or hyperbolic tangent, (c) Rectified Linear Unit(ReLU), (d) Leaky Rectified Linear Unit(LeakyReLU)

In the above example (Figure 2.4) there were 2 activation functions g() and g'(). If these functions were linear then the network transformation would be reduced to a product of 2 matrices, which is again a matrix. Although if the g() was considered to be non-linear then the network would represent easier some general-purpose problems.

Another thing to note here is that if a sigmoid activation function is used in the hidden layers then its properties will hold even if in the output layer a linear function is applied, as when a linear function is used as an activation function is like not applying any change in the values. In general for interpolation problems, where a smooth change in the values is needed, usually, a linear activation function is used in the output layer (Sharma et al., 2020). For classification problems though a sigmoid function is used as it limits the values between 0 and 1 values. Sigmoid however is inappropriate for most interpolation problem as the restriction of the output between 0 and 1 is not needed.

2.4 Network training and error implementation

In order to better understand the training and especially the loss which will be mentioned in this chapter, an analogy is drawn between the training of neural networks and the fitting of a polynomial curve. Below the m-th order polynomial equation is illustrated in Equation 2.6:

$$y = w_m x^m + \cdots + w_1 x + w_0 = \sum_{j=0}^{m} w_j x^j \tag{2.6}$$

This is a non-linear mapping that takes x as input and y as output. The value of y is determined by the values of $w_m \ldots w_0$ that are equivalent to the weights in a neural network. The equivalent of the w_0 is the bias. Polynomials are in general similar to neural networks, although neural networks have a higher number of inputs compared to the one input of polynomials and they can also define a large class of functions easily. In fact, a large enough network can define any continuous function with sufficient accuracy (Bishop, 1994).

Once the neural network has been configured it must be trained on the dataset. Training is called the process of deciding the values of the weights in the neural network. To better introduce this process, the analogy of the polynomial curve above will be used for a set of data. Each point in the dataset, which is used as the input, has an index number q=1,…,n, symbolized as x^q, and a value that is desired for the output to be, symbolized with t^q and called target value. To find the best coefficients for the polynomial, the error between the output value predicted by the function of the polynomial $y(x^q;w)$, for a specific data point x^q, and the corresponding desired output value t^q for the same data point is calculated. Usually, some error functions are used to be minimized. The most common one is the sum-of-squares error function illustrated below (Equation (2.7)).

$$E(w) = \frac{1}{2}\sum_{q=1}^{n}\{y(x^q, w) - t^q\}^2 \tag{2.7}$$

It is obvious that E is a function of w so that the curve will be fitted to the data just by selecting the best values for w that minimizes the function E. In the Figure 2.7 a cubic polynomial, in Equation (2.6) setting m=3, is fitted over a set of data by minimizing the sum-over-squares.

In neural networks, the training follows an analogous pattern. Firstly a suitable error function is found, with respect to the input data. The weights are then chosen in order to minimize the error. Although, minimizing the error function in a complex neural network is more difficult than in the polynomials as the network's functions depend in a non-linear manner on the weights and so require the use of non-linear algorithms for optimization. In neural networks, there are several input vectors $x^q(x_1^q,\ldots, x_d^q)$, each one of them having a target vector t^q. In that respect for every output k, the error, taking into consideration the Equation (2.7) if we instead sum over all q and output vectors k, is shown in the below Equation (2.8).

$$E(w) = \frac{1}{2}\sum_{q=1}^{n}\sum_{k=1}^{c}\{y_k(x^q, w) - t_k^q\}^2 \tag{2.8}$$

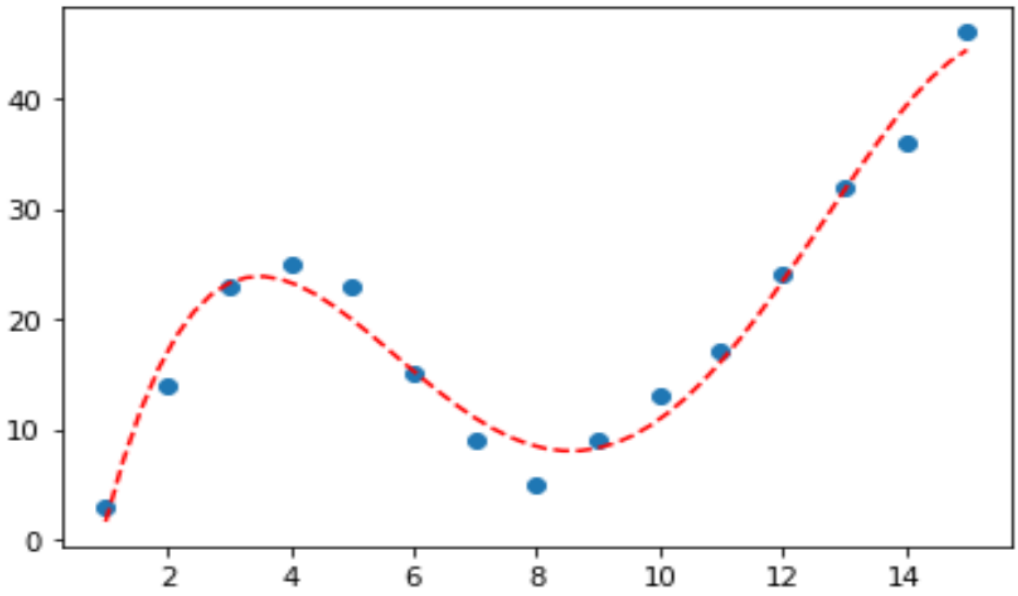

Figure 2.7: An example of curve fitting using a polynomial function.

The error function can be considered as a surface above the weights, as in the Figure 2.8. This way the problem of training a neural network can be regarded as finding the minimum value of the error function, which is the w^A in the Figure 2.8. This is called global minimum. Although there might be other values higher than the global minimum, like w^B, which is called local minimum. The ∇E is the function's gradient at a point on the surface and is a key point to the gradient descent algorithm, as it firstly selects a value for the weight and then it alters it in order to move it in the direction of the negative of the error function gradient. In the case of single-layered neural networks with a linear activation function, the sum-of-squares error function has no local minima so by solving the linear equation the global minima are easily found just. Although in multilayered neural networks error function is non-linear and the minimum is found by firstly considering a random weight and then making some changes in order to find the minimum. Some algorithms will find the local minimum, which in some applications where the surface is very complicated can be sufficient, while others have techniques to escape the local minima and have a higher possibility to find the global minima.

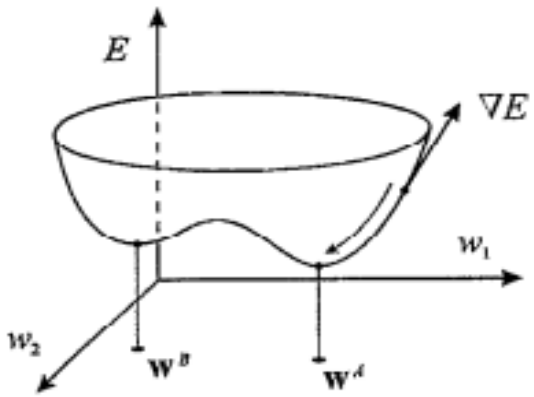

Figure 2.8: Schematic illustration of the error function E(w) as a surface above the weights w_i.

The derivatives with respect to the network's weights are used broadly from many error minimization algorithms. These derivatives are the components of the gradient vector ∇E of the error function. As there are substantial benefits for the training of the network in using gradient information there will be a more extensive discussion in the below paragraph about the ways to evaluate the derivatives and more specifically in the error backpropagation.

2.4.1 Back Propagation

Backpropagation algorithm (Bishop, 2006) is a very computationally efficient technique to calculate the derivatives, especially for nonlinear mapping function given by multi perceptron. Here for simplicity and better understanding, a general feed-forward network is considered with a single hidden layer, as shown in Equation (2.5), a nonlinear activation function, and the sum-of-squares error function, as indicated in Equation (2.8).

The error function can be written as a sum of terms, one for each data in the training data set (Equation 2.9)

$$E = \sum_{q=1}^{n} E^q(w) \tag{2.9}$$

For that reason, we can separately calculate the derivatives for each term and then sum over all of the terms in the data set together in order to get the required derivative.

Let's first consider that the output values y_k are linear combinations of the input x_n (Equation (2.10)).

$$y_k = \sum_i w_{ki} x_i \tag{2.10}$$

The error function for a particular term q can be written, as in the Equation (2.8), in the following way (Equation (2.11))

$$E^q(w) = \frac{1}{2}\sum_k \{y_k(x^q, w) - t_k^q\}^2 \tag{2.11}$$

The gradient of the error function for particular weight wji, taking into account the Equations (2.11) and (2.10), is given in Equation (2.12),

$$\frac{\partial E^q}{\partial w_{ji}} = \left(y_j^q - t_j^q\right) x_i^q \tag{2.12}$$

However, this evaluation was about one term of the error function. Let's now proceed in generalizing the above equation in a more complex feed-forward multilayered neural network. In a network like that, every unit computes the output as a weighted sum of the inputs while also considering the activation function of every layer. The output variable of the second layer (the layer of weights from hidden to the output layer) of the network can be written in the form (Equation (2.13)),

$$y_k = g'(a_k) \ , \ \ a_k = \sum_j w_{kj} z_j \tag{2.13}$$

, where z_j is the activation, or input, of a unit that sends its signal to the unit k, w_{kj} is the weight that connects the unit j with the unit k and g( ) is the nonlinear activation function of the unit k. There is no need to deal with the bias as it can be included in the sum with a fixed weight of +1.

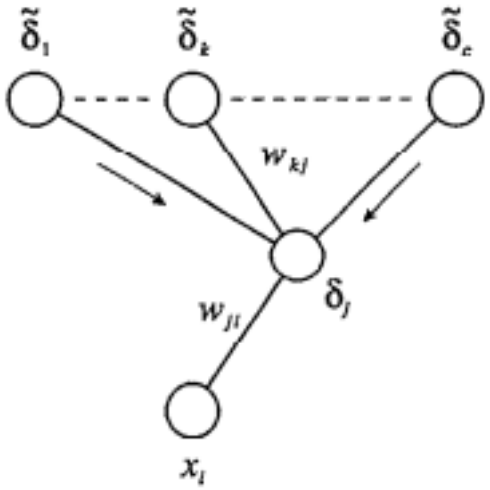

Figure 2.9: *Illustration of the calculation of δj for hidden unit j by backpropagation of the δ's from those units k to which unit j sends connections.*

The network being examined is assumed to be provided with the necessary input vector and it has calculated the activations of all hidden and output units just by applying the Equations (2.12) and (2.13) to the values. This process is most commonly referred to as forward propagation, as information flows forward in the network. The derivative with respect to the last layer weights w_{kj}, as it depends on the weight only via a_k, it can be written to the form,

$$\frac{\partial E^q}{\partial w_{kj}} = \frac{\partial E^q}{\partial a_k}\frac{\partial a_k}{\partial w_{kj}} \tag{2.14}$$

Now the following useful notation is introduced,

$$\delta_k \equiv \frac{\partial E^q}{\partial a_k} \tag{2.15}$$

, where δ can be considered to be the error as it expresses the difference between the network's output and desired value. Using the Equation (3.13) z_j can be written as,

$$\frac{\partial a_k}{\partial w_{kj}} = z_j \tag{2.16}$$

Then by making use of the previous 3 Equations (2.14), (2.15), and (2.16) the derivative becomes,

$$\frac{\partial E^q}{\partial w_{kj}} = \delta_k z_j \tag{2.17}$$

After that, an expression for the error of the hidden units can be found using Equations (3.11), (2.13), (2.15)

$$\delta_k = g'(a_k)(y_k - t_k) \tag{2.18}$$

Note that in Equation (2.17) the derivative is simply found with a multiplication of the values δ, for the unit at the output end of the weight, with the values z, for the unit at the input end of the weight. Note also that this equation takes the same form as the simple linear model discussed at the start of the paragraph, so the only thing that must be done to evaluate the derivative is to apply the Equation (2.17) in the values of the δ's in the hidden and output layers.

Now a similar equation must be found for the derivative with respect to weights in the first layer. Similarly to before the output variable of the second layer (the layer of weights from hidden to the output layer) of the network can be written in the form (Equation (2.19)),

$$z_j = g(a_j) \ , \ \ a_j = \sum_i w_{ji} x_i \tag{2.19}$$

, where x_i is the input vector of the network. Then the derivative can be written as below (Equation (2.20)),

$$\frac{\partial E^q}{\partial w_{ji}} = \frac{\partial E^q}{\partial a_j} \frac{\partial a_j}{\partial w_{ji}} \tag{2.20}$$

Now the following useful notation is introduced (Equation 2.21)),

$$\delta_j \equiv \frac{\partial E^q}{\partial a_j} \tag{2.21}$$

, and the other element of the equation can be written as (Equation 2.22)),

$$\frac{\partial a_j}{\partial w_{ji}} = x_i \tag{2.22}$$

So the derivative takes the form (Equation 2.23)),

$$\frac{\partial E^q}{\partial w_{ji}} = \delta_j x_i \tag{2.23}$$

, which is the same form as the Equation (2.17), so that the derivative that connects the input of the network with the hidden layer is the δ for the hidden layer multiplied by the input of the network.

Finally, the expression for the δ's is found just by using the chain rule for the partial derivatives (Equation 2.24)),

$$\delta_j = \frac{\partial E^q}{\partial a_j} = \sum_k \frac{\partial E^q}{\partial a_k} \frac{\partial a_k}{\partial a_j} \tag{2.24}$$

So by using the Equations (2.13), (2.15), and (2.18) we obtain the full back propagation expression (Equation 2.25)),

$$\delta_j = g'(a_j) \sum_k w_{kj} \delta_k \tag{2.25}$$

, which states that the value of δ for a hidden unit can be obtained by backpropagating the δ's from units higher in the network hierarchy. Figure 2.9 illustrates graphically the backpropagation that was described with equations above. Information during the forward propagation flows upwards in the figure while the black arrows show the direction of the error information during back propagation.

In general, the backpropagation algorithms can be summarized in the following steps:

1. Insert an input vector x_n in the network and then forward propagate the information in order to evaluate the activation functions of the output units, Equation (2.18), and the hidden units, Equation (2.13).
2. Evaluate the errors for the output unit using the Equation (2.18).

3. Backpropagate the δ's using the Equation (2.25) to find the values for the error of the hidden units.
4. Use Equations (2.17), and (2.23) in order to find the necessary derivatives.

The backpropagation method is used generally because of its computational efficiency. Supposing the number of the weights in the network is N, the scaling of the derivative with the number of the weights is needed to be found. As the error function $E^q(w)$, is associated with all the weights the evaluation of a single pattern will take O(N) steps. This means that the number of steps will increase like the weights. Accordingly, the evaluation of a derivative of the error function with respect to a single weight will take O(N) steps. There are N such derivatives so the number of steps would be $O(N^2)$ to calculate all the derivatives. On contrary, the backpropagation method evaluates all the derivatives only using a forward propagation, a backward propagation, and the use of the Equations (2.17) and (2.23). Since each of these calculations is done in O(N) steps, the evaluation of all the derivatives takes O(3N) steps. For a set of data with n patterns, the total number of derivatives for the error function E would be O(n3N) in comparison to the $O(nN^2)$ if it was calculated separately with direct evaluation. This might not make so much of a difference in a lower number of weights but since the number of weights in a network can range from a few hundred to many thousands in larger networks, the saving in time for the calculation is significant. Taking also into consideration that, even with the use of the backpropagation method for evaluating the error function, in a multilayered perceptron the computing is still demanding the use of the backpropagation is necessary (Bishop, 1994).

Some more advantages of the backpropagation algorithm, without giving details as they are of less importance, are:

1. It simplifies the network structure by removing weighted links.
2. It is fast and easy to program.
3. It does not require prior knowledge about the networks.
4. There is no need to specify the features of the function to be learned.
5. It allows efficient computation of the gradient at each layer.

2.5 Convolutional Neural Networks

Although ANNs of type Feed Forward like the Multilayer Perceptron that was discussed in the above chapters are very good at dealing with problems that take as input a one-dimensional vector, they have difficulty in those that have inputs of 2 or 3 dimensions. Such problems are image classification, image recognition, voice recognition, and more. This is where Convolutional Neural Networks, CNNs, come in.

CNNs can be considered as special case feed-forward neural networks. They don't differ as much from the already discussed ANNs, as they are made up of neurons with learnable weights and biases. The main difference is that instead of general matrix multiplications, as in ANNs, they use convolution in at least one of their layers. Also, their input is considered to be an image, instead of a one-dimensional matrix, which means that knowing the input from the beginning the network can be enhanced with different properties in its architecture that help in the whole classification process (Teuwen & Moriakov, 2020).

The basic use of CNNs is to classify images and patterns, like recognizing if in the picture there is a dog or a cat. Although before the model is ready to recognize classes in an image the image must be processed and the different features of the image must be extracted. Before CNN, some experts had to design their own feature extractor for the specific image, which was not only

costly but also time-consuming while being inconsistent as a method as every image could differ in a great way from the others (Kim, 2017).

With the introduction of the CNN, the feature extractor was included in the training method rather than being designed from scratch. This feature extractor consists of some convolution layers, which are the main feature extraction layer, and some pooling layers, that help reduce the dimensionality of the image. After the features are extracted they enter the classification network, which is consisted of Fully Connected Layers. It takes as input the features extracted, processes them, and then generates an output. These particular layers will be discussed in the next sections in detail.

2.5.1 An Introduction to images

Images consist of pixels, which are the picture's elements, and carry information about the color of the picture. In a 1920x1080 picture, the total number of pixels is 2.073.600, namely 2.1 megapixels. Each pixel has a size of eight bits or more and the ability to project millions of different colors and contain several channels depending on the colors that the picture has. There are 2 types of pictures.

Firstly there are the grayscale pictures. In these pictures every pixel has only one channel, which value ranges from 0, being the total black, to 255, being the total white. This way every pixel carries only the information about how high in the grayscale is the color that it represents, thus having only one value. A simple example of a greyscale picture is shown in the Figure 2.10 below.

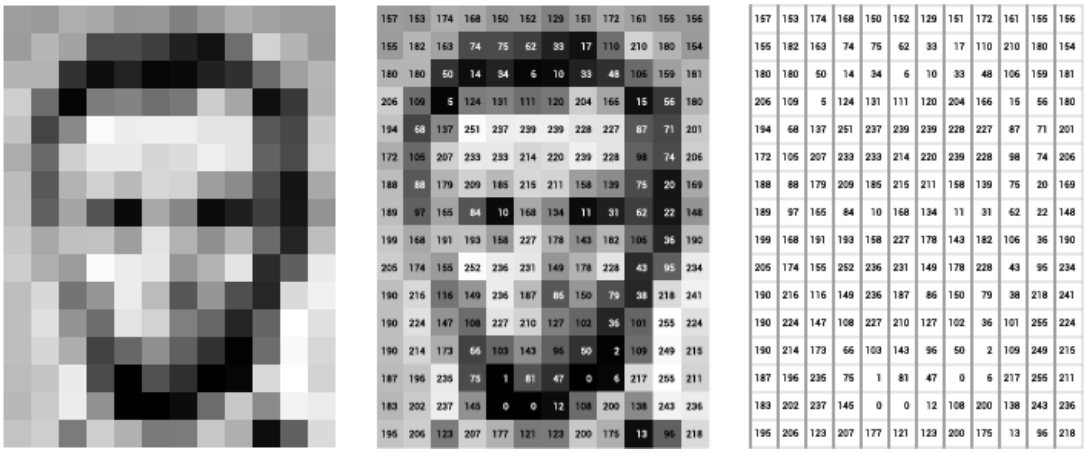

Figure 2.10: A grayscale image with the representation of the values of each pixel.

Although if instead of black and white the picture is in full color then the pixels would have 3 channels if the Red Green Blue (RGB) system is considered. Each channel in this system is a grayscale image of the same size as the color image, made just from one of the primary colors Red, Green, or Blue. Sounds confusing but let's suppose a grayscale that ranges from 0 to 255 and instead of the 0 being the total black it is the total Red or Green or Blue. This way the scale of the chosen color is created. Every pixel in fully colored images carries 3 values, ranging from 0 to 255 and each one shows how much of the specific color this part of the picture has. Then combining these 3 channels a great number of colors can be created. If in the same figure (Figure 2.10) there were all the colors of the canvas, considering the RGB color system, then the picture would be a two-dimensional vector with each cell holding an RGB triplet. In Deep Learning though it is more convenient to consider the picture to be a 3-dimensional vector with

each dimension holding a 2-dimensional vector for each color of the RGB system, namely a 12x16x3 array.

2.5.2 Convolution Layer

As the name implies, convolution layers are one of the most vital elements of a CNN. Through this layer, new pictures are generated, called feature maps. The features maps are just pictures similar to the starting one but they highlight the different unique features of the picture. The different feature maps are generated with the help of some filters that convert the picture. These filters are called kernels (O' Shea & Nash, 2015). They have small dimensions, usually 3x3 or 5x5, sometimes 1x1, but their effect is spread throughout the entirety of the input. The values of the kernels are trained and changing constantly throughout the training process just like the values of the weights in an ANN. That's because as the input hits the convolution layer the scalar product between the input and each kernel is calculated in order for the feature map to be generated. In the Figure 2.11 below an example of some feature maps of a handwritten number 9 generated by 5x5 kernels is shown.

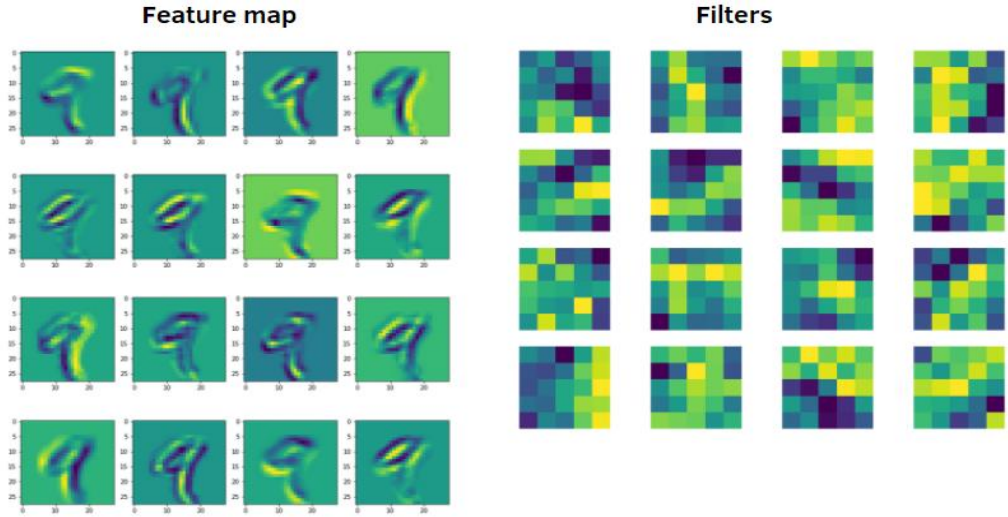

Figure 2.11: Examples of feature maps for a picture showing a handwritten 9 and the kernels that were used to create them.

A scalar product, or else dot product, is an element-wise multiplication of the filter, kernel, and the filter-sized part of the image. These values are then summed in order to result in a single value every time. That's the reason it is called a scalar product.

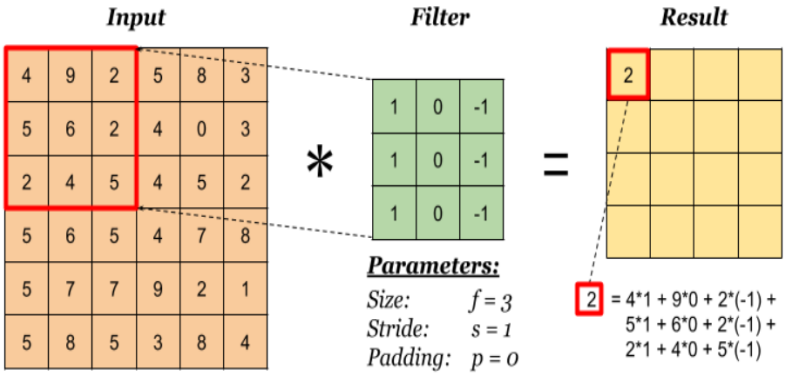

Figure 2.12: The beginning of the operation.

The convolution layer's operation is difficult to understand, as it involves calculations between two-dimensional vectors so a simple example will be shown. Firstly a 6x6 pixel image, as in Figure 2.12 with the form of a matrix is considered. A filter with a size of 3x3, the so-called kernel size, is also assumed. In the convolution process, the filter is multiplied element-wise with the same size patch of the input image, as shown in the Figure 2.12 below. The results of these operations are then added together in order for the single-valued output to be calculated. The operation is always starting from the top left corner of the matrix. After the result of this part is calculated the filter slides over, as shown in the Figure 2.13, both right and down by a number of elements, and does the same computation until the whole matrix is covered.

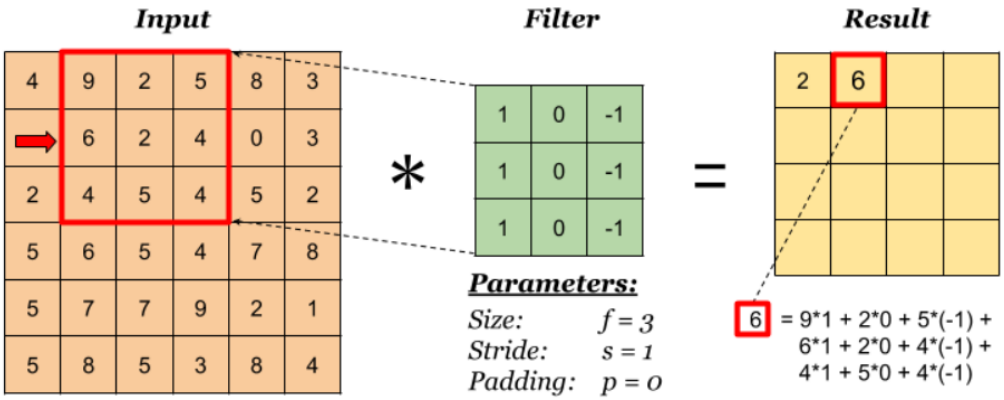

Figure 2.13: The calculations for the filter moving one step right.

Although when the input image has more than 1 channel, as in case of fully colored pictures, the calculation gets more complicated. The extra operation that must be done to get one number, as a result, is to apply the convolution operation for each channel and then add all the results together. An example is shown in the Figure 2.14 below.

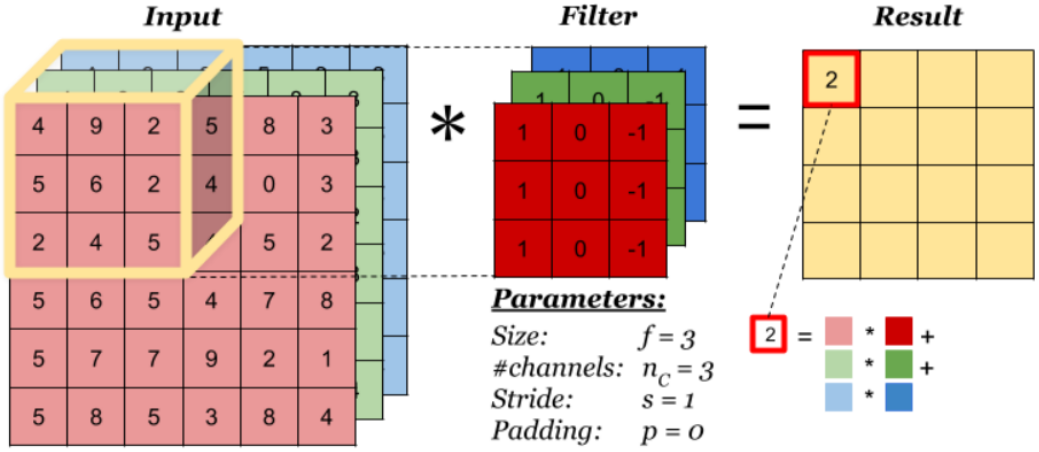

Figure 2.14: The operation for a 3 channeled vector.

2.5.2.1 Stride

A very significant parameter of the convolution layer's process is the stride. As discussed above, the operation starts from the top left corner and then continues to the rest of the picture by a step. The number of elements that the filter slides right to calculate the next result is called stride (O' Shea & Nash, 2015). In the previous example, the stride was set to 1. Although sometimes the filter can be moved more than one element at a time, skipping the intermediate locations (Figure 2.15).

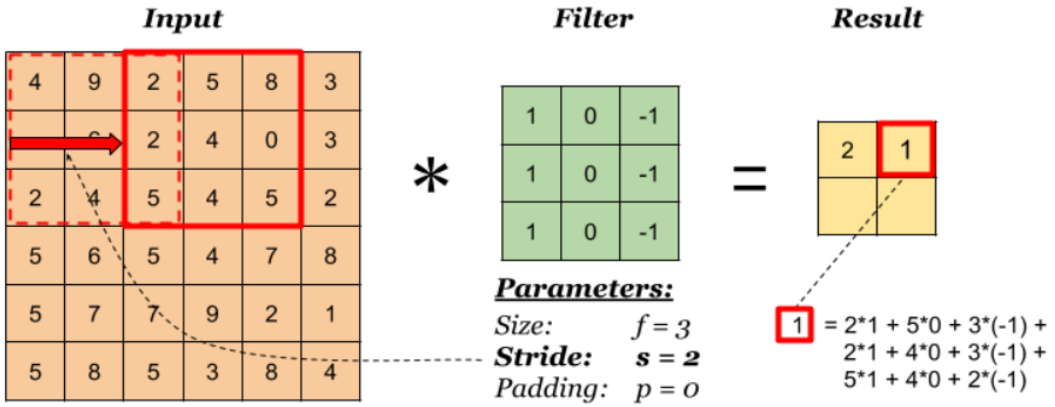

Figure 2.15: Example for a stride number different that the default 1.

Stride numbers different than the default 1 can have a positive effect on the computational efficiency of the network and also can be used in order to reduce the size of the output image, called downsampling if needed.

2.5.2.2 Padding

Another way to affect the size of the output is through padding. Padding is a technique in which extra pixels are added around the boundaries of the picture, as illustrated in the Figure 2.16. This might seem not so useful at first thought but let's discuss its practicality. In general, the kernels being used in CNNs have a width and height greater than 1, so after doing some consecutive convolutions the result is significantly smaller than the input (O' Shea & Nash, 2015). For example, an image with 240x240 pixels after 10 convolutions with a kernel size of 5x5 will end up with 200x200 pixels eliminating any information on the boundaries of the image that may be useful. A straightforward solution to this problem is just to add extra pixels around the picture so that the effective size of the image increases.

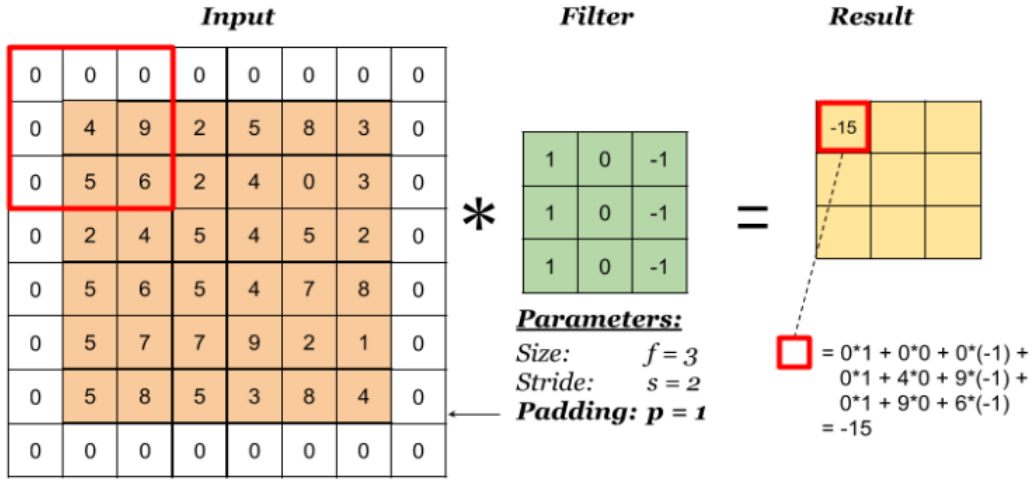

Figure 2.16: Example for padding equal to 1.

In CCNs when a kernel size with odd height and width values is used, like 1,3,5,7, then the dimensions of the image can stay the same. Some terminologies that are used in padding are the following:

- "valid", which means no padding
- "same", which means that the padding used is calculated so that the output has the same dimensions as the input.

2.5.2.3 Mathematical implementation of the Convolution operation

The convolution function (x*w)(a) is defined as (Equation (2.26)),

$$(x * w)(a) = \int x(t)w(t-a)da \tag{2.26}$$

, where a and t are the parameters of the problem, x is called the input, w is the so-called filters or kernels, as mentioned in the above sections, and the output is the feature map or activation.

In the above Equation (2.26) though the input and kernel are considered continuous functions. Images have a discrete number of pixels so it is useful to consider that the parameter t is discrete. So the discrete convolution can be written as (Equation (2.27)),

$$(x * w)(a) = \sum_{a} x(t)w(t-a) \tag{2.27}$$

In machine learning applications, the input is a multidimensional array of data, and the kernel is a multidimensional array of parameters. As the w includes the parameters of the network, that are finite in number, then the function w(a) is considered to be non-zero only for a finite amount of values a (O' Shea & Nash, 2015). This means that the above equation can be implemented as a finite sum. In images there is an interest in 2- (Equation (2.28)),or 3-dimensional convolutions (Equation (2.29)),

$$(I * K)(i,j) = \sum_{m}\sum_{n} I(m,n)K(i-m,j-n) \tag{2.28}$$

, where I is a 2-dimensional image input and K is a 2-dimensional kernel

$$(I * K)(i,j,k) = \sum_{m}\sum_{n}\sum_{l} I(m,n,l)K(i-m,j-n,k-l) \tag{2.29}$$

, where I is a 3-dimensional image input and K is a 3-dimensional kernel

As the Equations (2.26) and (2.27) are commutative, it is true that I*K=K*I, so the above Equation (2.28) can be written as (Equation (2.30)),

$$(I * K)(i,j) = \sum_{m}\sum_{n} I(i-m,j-n)K(m,n) \tag{2.30}$$

2.5.3 Pooling Layer

Convolution or convolutional layers apply some filters or kernels to an input image and create a feature map of the summary of these specific features in the image. These layers are helpful when they are stacked in a deep neural network. That is because those close to the input of the image can imprint low-level features in it, like lines, and the ones deeper in the network high-level features, of it, like shapes or even some specific objects.

Notwithstanding, convolutional layers do have some limitations. The different feature maps produced from these layers record the exact location of the feature in the image. This way with the slightest change in the position of a feature, which happens when rotating, shifting, and re-cropping, a different feature map will be produced complicating the generalization of the model. This problem can be addressed with a method called down sampling. In this method, the input image is turned into a lower-resolution one so it keeps all the useful features without the fine details that create more of a problem rather than having some use to the task.

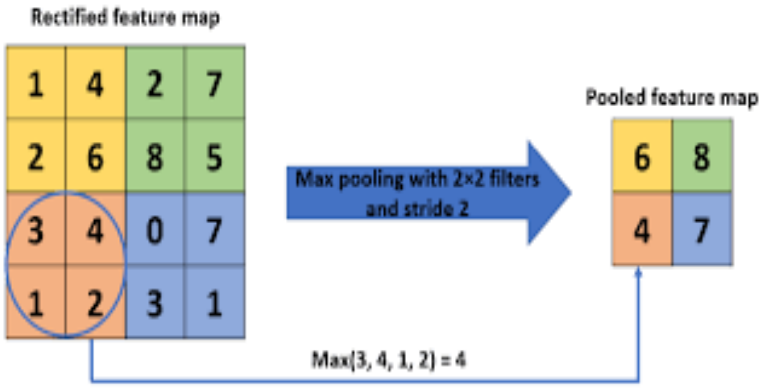

Figure 2.17: Pooling operation with a size of 2x2 pixels and stride of 2.

The most common method, except for changing the number of strides in the convolution, is by adding a pooling layer in the network. Pooling layers aim to reduce the size of the image, thus reducing the number of parameters and the computational complexity of the model (Kim, 2017). This is achieved by the combination of adjacent pixels of the image to create a single value. The pooling layer is usually added after the convolution layer and operates over all the feature maps thus creating a new set of feature maps with the same number as the starting one.

In these layers, the selection of the pooling operation is very important. The size of the operation must be smaller than the size of the image and is in most cases 2x2 pixels with a stride of 2 like in the following Figure 2.17. In the pooling operation 2 functions can be used, as shown in the Figure 2.18:

- Average Pooling, which calculates the mean values of each patch of the feature map.
- Maximum Pooling (or max), which calculates the maximum values of each patch of the feature map.

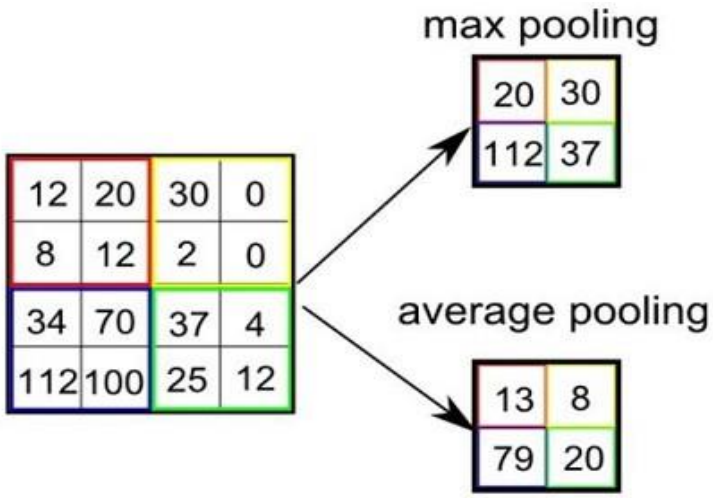

Figure 2.18: Illustration of maximum and average pooling.

The feature map produced from the 2 methods is different, but both of these methods are used in applications that can benefit from their advantages. The average pooling method smooths out the image so that some sharp features are not identified. Max pooling selects only the brighter pixels from the image and the ones that are more important like edges for example, so in cases when the background of the image is dark, this function is the one to go for when creating a CNN. The other way around when the background is too black min pooling is used, but is not that common, so it is not extensively discussed. A simple example in the Figure 2.19 is shown to pinpoint how differently they affect the image.

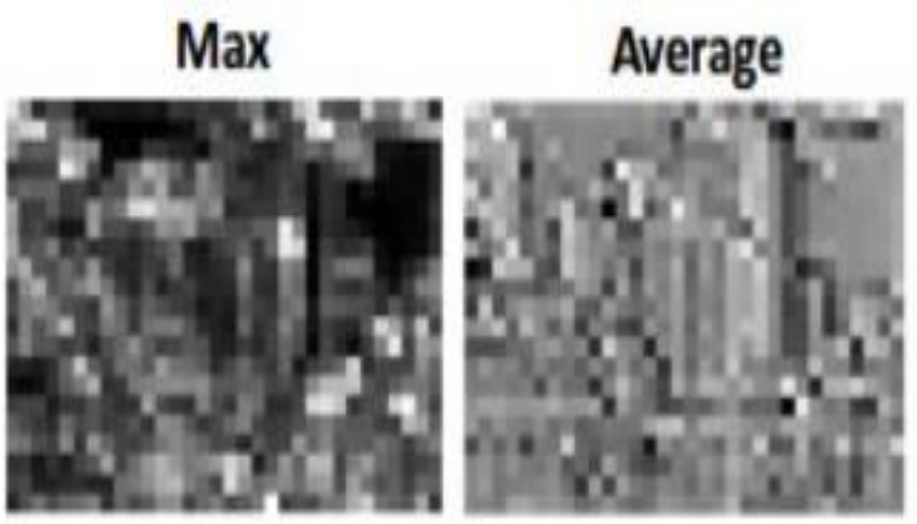

Figure 2.19: Effects of maximum and average pooling in an image.

2.5.4 Fully connected layers

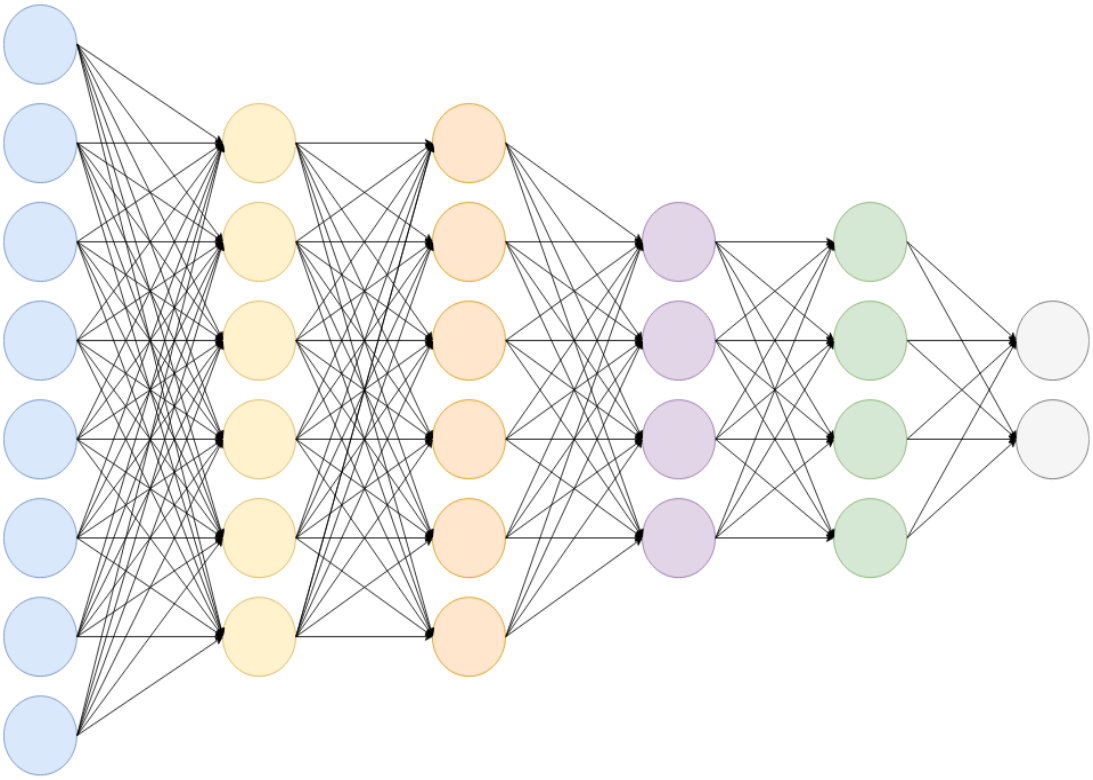

Figure 2.20: A representation of a fully connected network.

In the output of the last pooling layer or the convolution, in case there is no pooling, is usually applied a flattening, turning the multilayered array into a single 1-dimensional array of numbers. Then it is connected to one or more fully connected layers, which is known also as a dense layer, and is simply a feed-forward neural network (Teuwen & Moriakov, 2020), as the ones discussed in previous chapters. These layers are usually the last layers of a CNN and every neuron of each layer is connected to all the neurons of the next layer with a learnable weight, as depicted in the Figure 2.20. The fully connected layers are the ones that map the features that are extracted from the previous layers and produce the result of the CNN. Their output nodes are usually as many as the number of classes used in the model. Each one of the fully connected layers also uses a non-linear function as an activation function like the ones described in the Chapter Activation Functions. The output of a fully connected layer is shown in the below equation (Equation (2.31)):

$$FC(x) = f(wx + b) \tag{2.31}$$

, where f is the activation function, w is the weight, x is the input and b is the bias.

2.6 Generalization, overfitting and early stopping

Training in the CNNs world is called the process of adjusting the kernels of the convolution layers and the weights of the fully connected layers in order to minimize the difference between the output values and the input values. The topic of training is more extensively discussed in the above chapter Network training and error implementation so here a general view will again be given.

To start with CNN gives some random values to its learnable parameters. During the process of training, the network processes the data, which are labeled with their respective class, randomly and compares the output values of the class with the ones of the input image. If the output differs from the input, which happens more at the beginning of the training, then the network makes small adjustments in the learnable parameters in order for the input to match the output. A loss function calculates how well the network performs under specific kernels and weights through forward propagation and updates these parameters in accordance with the loss value with the use of the backpropagation algorithm, which was discussed in the above chapters.

The training data are processed multiple times by the CNN until sufficient accuracy is found or the loss is minimized. Each run of the training data is called epoch. The epochs that the data will be processed can be defined by the programmer or, in some cases of more advanced programs, the program can be set to stop when the accuracy reaches a max or the loss reaches a min. As the network improves epoch by epoch the loss between the input and the output becomes smaller and smaller so the adjustments done are decreasing.

The goal of the training process of the network is to achieve good performance. The network's performance depends on many variables, some of them being the size of the training data and the network. For example, if a large deep network is given a small amount of training data then it will be generally easy for it to learn the data. Although keep in mind that the network should "memorize" the general trends of the data in order to have good results when facing new data outside the training dataset (Bishop, 2006). Supposing the data of the training set have a noise, as is in most applications, then if the network accomplishes a very good fit in the data the network will have memorized the exact noise of the data and will have a poor performance in data not being included in the dataset. Thus for a network to have a good performance on new data it must have the required flexibility to learn the trends of the training data and not their noise.

To better understand the issues of generalization of a network the problem of fitting a polynomial curve through a set of data points that have a noise is considered. In the Figure 2.21 below a cubic polynomial curve and 10 points that have a noise in comparison to the curve are depicted. Considering the same equation (Equation (2.6)) as in the Chapter Network training and error implementation different numbers for the m are picked to see which fits better for the data. If the order m is too low, as in the cases of m=0 and m=1, then the curve produced gives an inadequate representation of the trends in the data, so its predictions with new data will be poor. If the order is increased to m=3 then, as shown in the Figure 2.21, the curve represents the general trends of the data much better. Although if the order is increased too much, as seen in the m=9 case in the Figure 2.21, then the problem of overfitting occurs. This means that the curve is very exact in representing the data in the training dataset but it has memorized the noise of the points and not their trends so it has a poor prediction of new data (Bishop, 1994).

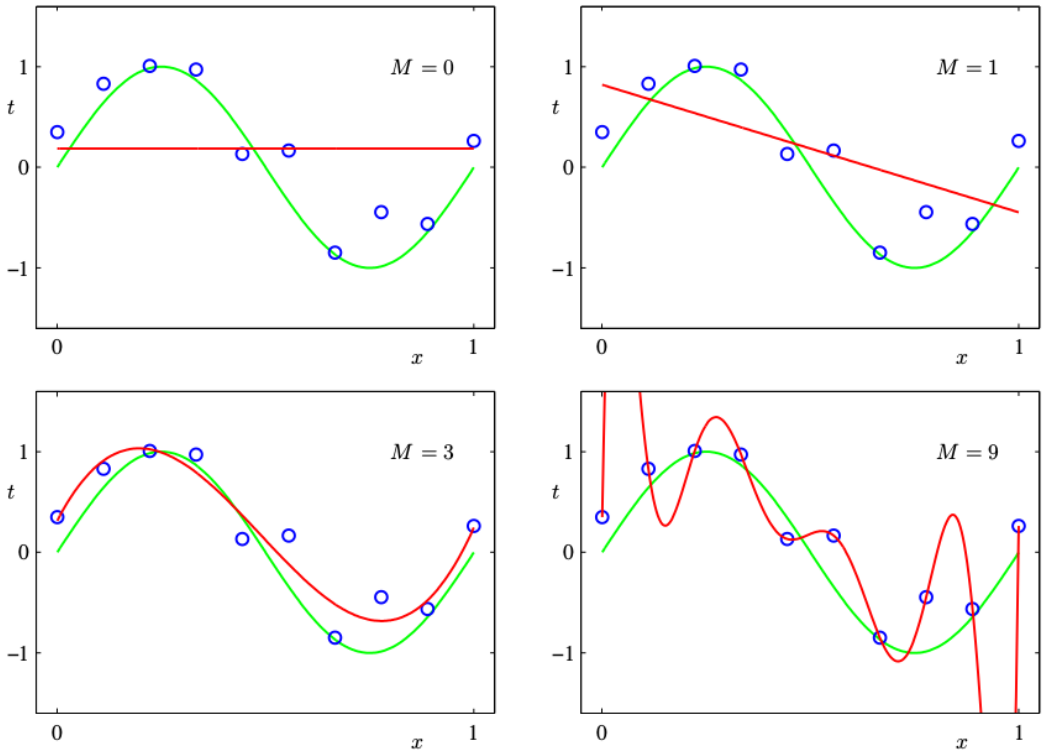

Figure 2.21: An example of curve fitting using successively higher-order polynomials.

The same situation can happen in neural networks. Overfitting is one of the main challenges in machine learning as an overfitted model can't generalize well in new data. A way to check if a model is overfitted on the training data is to see the accuracy and loss function of the training and compare them to the ones of the validation data (Yamashita et al., 2018). A diagram of a model overfitted is illustrated in the Figure 2.22. The loss of the training data is a function that is always decreasing as the iterations increase. However, that doesn't happen always in the loss of the validation data, as at some point it starts increasing. That point is where overfitting starts. To prevent this more data can be added to the training. Although this is not possible every time so the method of early stopping can be applied. In this method, the training is stopped at the point where the validation loss has the smallest value. This method is applied also in the network used in this thesis.

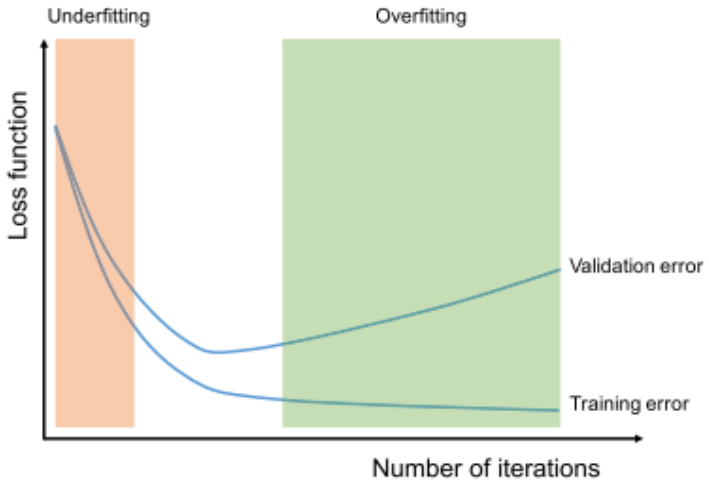

Figure 2.22: A diagram showing the relation between the error function and the number of iterations.

2.6.1 Training, validation, and test data

As a mention has been made about training data in the previous chapter, a more general discussion of the topic will be done in this section. Data are one of the most crucial components when creating a neural network. After having already created the network the next step is to collect the necessary data that the network will be trained on. This step is very important to the network although obtaining the right number of high-quality images as data is very time-consuming.

The total number of data collected is split into 3 sets usually. A training set is used to train the model and consists of pairs of an input vector and its corresponding output vector also called label. The loss is calculated using forward propagation comparing the output values of the network and the labels for each training data, then the learnable parameters are changed using backpropagation. A validation set is separated from the training model and is used for the already fitted model to predict the responses (Yamashita et al., 2018). This dataset provides an evaluation of the model while tuning the hyperparameters. Validation datasets are also used for regularization using early stopping. A test set is used, normally containing data not included in the training dataset, in the final model, as the training has ended to evaluate its performance. A normal ratio of training:validation data is 90:10 or 80:20, while the test data can be of any number as it doesn't affect the training.

2.7 Object Detection

The main fields in computer vision that interest the researchers are 3: image classification, image localization, and image detection. These different methods can be seen in the Figure 2.23 below. The potential and challenges of these tasks with the parallel incorporation of deep convolutional neural networks to the field of computer vision gave the spark for the immense increase in the work being done in the fields.

In image classification the main goal is to determine if there is an object in the image, turning a picture into a label. For example in image classification from a set of images with dogs and cats the ones that have a dog can be separated from the ones having a cat. However, the location of the different objects in the picture is still unknown. So here comes image localization to point out this position.

Although if in a picture there is more than one object, a cat and a dog, then with this method it can't be defined where this will be categorized with the methods above. This problem is solved with object detection, where in each picture not only the label of a single object can be found but also the presence of other objects of different labels or more objects of the same label. For example, in the Figure 2.23 below both the 2 cats, the duck, and the dog is found, labeled and their location is pointed out. Object detection is considered the first process in computer vision, having many applications in security systems, human detection, robotics, product detection, and so on.

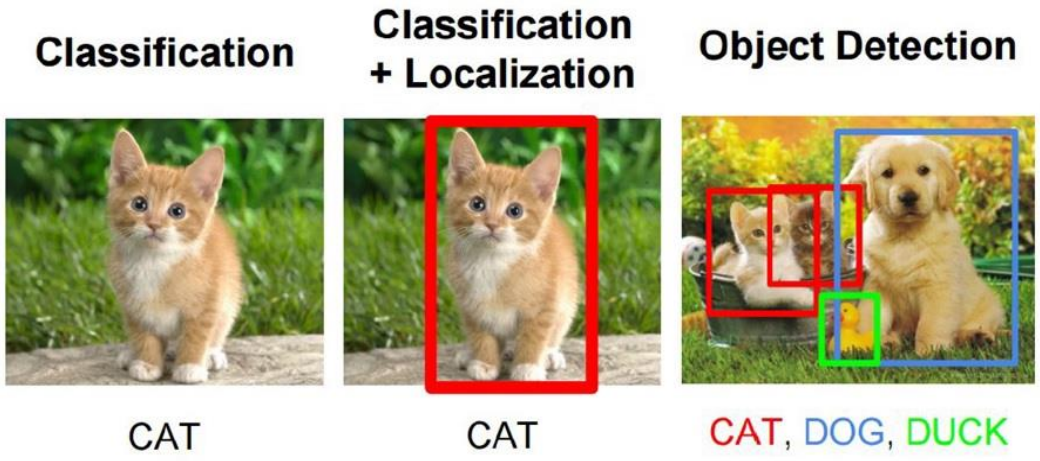

Figure 2.23: Illustration of the main fields of computer vision

People can learn things in many ways, as from their experience. Although for a machine to learn the different tasks it must be trained. The more proper the training is the better and faster the response of the system will be (Bhagya & Prof. Shyna, 2019). In the case of object detection, the training is done by training a classifier that can obtain even the slightest difference in the objects. In the input of this classifier, some areas are presented. These proposals, as they are called, have a significant role in the method and if they are improper they can have a negative effect on the output of the system. Thus these proposals are determined by deep neural networks.

2.7.1 Applications of Object Detection

Object detection is currently a very trending area of interest as the applications in real life are just endless (Kamate & Yilmazer, 2015). Some common applications are the following:

A. Face Detection and Face Recognition

This is one of the most popular applications and is mostly used by social media for detecting faces in an image (Albiol et al., 2001). The convolutional neural networks have a significant role in spotting the faces, extracting the facial details, and returning the output. In smartphones face recognition is used for unlocking the phone and face detection for recognizing the presence of a face when taking a picture. Also, banks, vaults, and other high-security places use face recognition when entering an area. Finally, in airports, retail stores, stadiums, and other facilities face recognition is used to prevent violence (Kamate & Yilmazer, 2015).

B. Security and Surveillance

Taking into consideration the rise of criminality levels in today's society this application can be very useful for detecting intruders or even explosives in a remote facility or a house. Furthermore, anomaly detection is a field that businesses spend a lot of money on so a quick and accurate program is needed (Kmieć & Glowacz, 2015).

C. Robotics

Robotics is the most obvious of the applications that object detection can be useful. For the robots to quickly respond to environmental changes they must be provided with a fast and accurate visual image processor and object detection is usually the first step in achieving this goal (Lu et al., 2017).

D. Object Tracking and Counting

Object detection can be used when tracking some objects like the ball in football games or the movement of a person in a video camera and so it can be used in security systems too(Kamate & Yilmazer, 2015). Apart from that, it is applied in traffic monitoring, animation, video communication, robots, and so on (YuanQiang et al., 2020).

E. Self-Driving Vehicles

Last but not least the more modern application of object detection is in self-driving vehicles. For the vehicle, mainly speaking about cars, to accelerate, brake, or turn, it must know all the objects that are in its vicinity and what they are. These objects are cars, pedestrians, animals, traffic lights, signs, trees, and so on. To make a good and fast decision the convolutional network must be fast and effective or else it won't have time to react. Although except cars nowadays research is being done in order for the ships to be made autonomous. The new fashion has already started but is not yet generalized (Naghavi et al., 2017).

2.7.2 Techniques employed in Object Detection

2.7.2.1 Sliding Window

The most common and easy method being used in object detection is the sliding window method (Vedaldi et al., 2009). The first to create a face detector using this method was Viola and Jones in 2001 (Viola & Jones, 2001) and then in the next years, more progress is done on the topic with the works of Dalal and Triggs with the Histogram of Gradient Detector in 2005 (Dalal & Triggs, 2005) and Felzenswalb in 2010, who did an object detection system that represents highly variable objects using mixtures of multiscale deformable part models (Felzenszwalb et al., 2010). In the sliding window method, a rectangle is created, with respect to the object that is searched in the image, and then the rectangle scans the whole image starting from the top left corner and moving to the right. The box is usually much smaller than the image and it is enlarged every time it scans the picture by a standard value. This procedure is done continuously until a certain condition is reached. It is obvious that with this method the number of windows created for a single object will be very high and also the windows that have to be taken into consideration in order to find all, or at least the majority, of the objects in the picture, is massive. Thus this method is considered to be very computationally expensive as well as giving inaccurate bounding boxes for the objects.

2.7.2.2 Regional Convolutional Neural Networks (R-CNN)

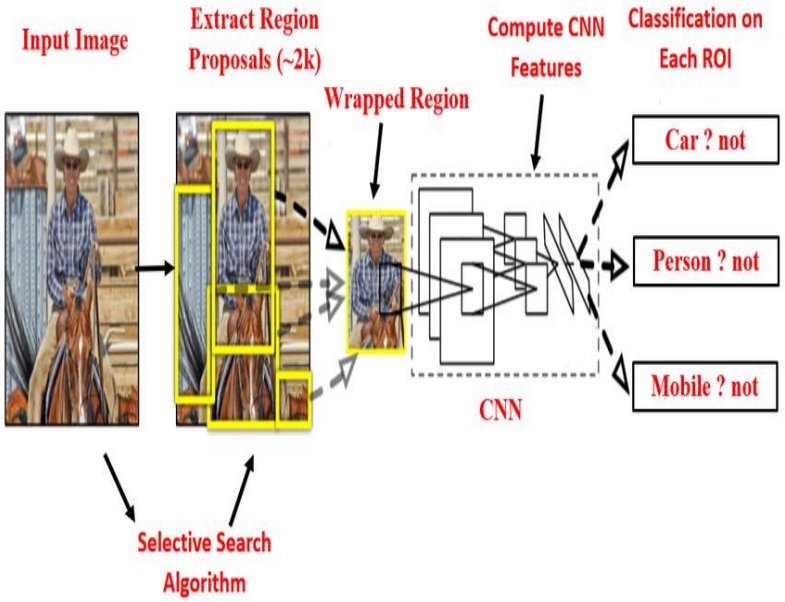

Figure 2.24: R-CNN architecture.

In 2013 R. Girshick tried to solve the problems of the sliding window method by presenting his new method for object detection called Regional Convolutional Neural Networks (R-CNN) (Girshick et al., 2014). To limit the number of windows produced by the program, this method chooses only some regions that are more important for the CNN to run on. The regions that are proposed for the CNN are extracted with the selective search algorithm (Uijlings et al., 2013). Selective search is an algorithm that takes as an input the image segments it, depending on similarity, and returns some regional proposals. In contrast with the sliding window method these regions are less in number and produce a higher recall. The processes of this method are visualized in the Figure 2.24. Despite that in R-CNN the regions proposed are reduced, the method as a whole has some disadvantages. The selective search algorithm is very rigid and it can't learn from the images input so the proposals are sometimes incorrect. It also takes a lot of time to train and the training has many stages. Thus it is very slow to detect and can't be used in real-time applications as it takes 50 seconds approximately to compute an image.

2.7.2.3 Spatial Pyramid Pooling Network (SPP-Net)

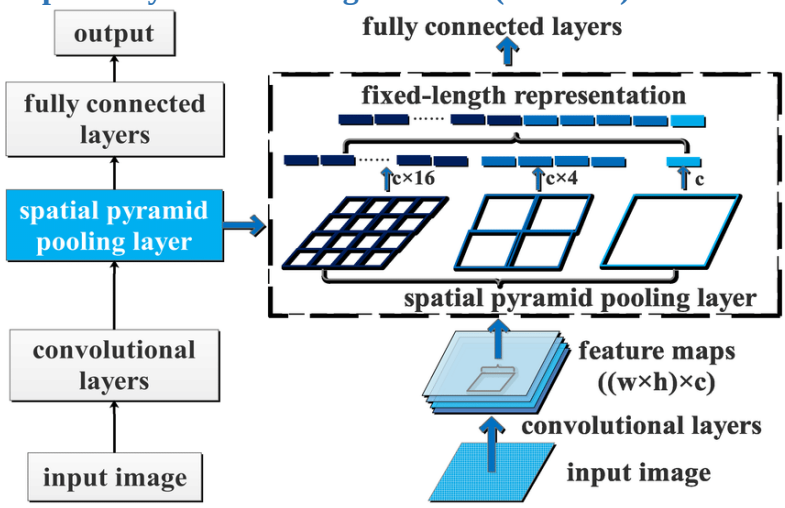

Figure 2.25: SPP-net Architecture with a more explanatory way of how the spatial pyramid pooling layer work.

Later in 2015 Kaiming He proposed another method for pooling called Spatial Pyramid Pooling (SPP) and a network based on this method called Spatial Pyramid Pooling-Network (SPP-Net) (He et al., 2015). CNNs in general can't accept images of varied sizes as the fully connected layers have as input fixed-sized images. If the images are of size larger or smaller than the ones that the fully connected layers want they are reshaped to reach the desired size, thus losing some important features. This problem is solved with SPP-net. Firstly the feature maps of the input image are generated, using a number of Convolutional layers. Those feature maps can then pass through the SPP pooling layer. The SPP pooling window and stride are relative to the input image so that a fixed-sized output is created. Furthermore, these layers apply a couple of different output-sized pooling operations and then combine them all in order to continue in the next layer, the fully connected as illustrated in the Figure 2.25. SPP-Net is considered to be faster than R-CNN as the only time-consuming part is the CNN, although they lack accuracy for very deep neural networks.

2.7.2.4 Fast Regional Convolutional Neural Networks (Fast R-CNN)

Later in 2015 Girshick (Girshick, 2015) came up with the idea of a new model, called Fast R-CNN, which purpose was to overcome the problems that occurred in R-CNN and SPP-Net. The approach is very similar to the R-CNN method but instead of passing all the proposed regions to the CNN, the image is considered as an input in order to create the regions of interest. The proposals are then concatenated and, with the use of a Region of Interest (RoI) pooling layer, feature vectors with a fixed size are created and inserted into the Fully Connected Layers. After that, a softmax layer is used to predict the class of the proposed region and its bounding box. The whole process is depicted in the Figure 2.26 below. This method is considered to be faster than the former R-CNN as it doesn't make a massive amount of region proposals in the CNN and also gives a higher quality object detection than SPP-Net and R-CNN.

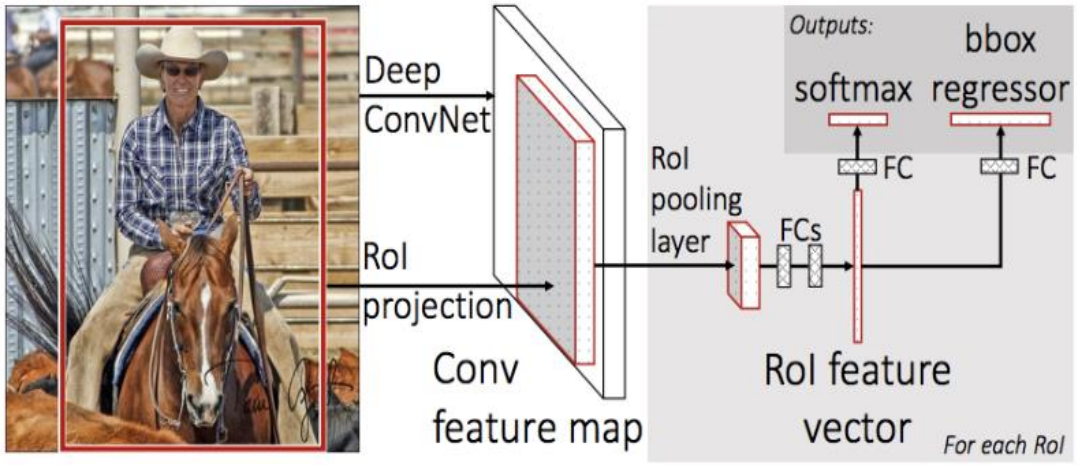

Figure 2.26: A representation of the Fast R-CNN model.

2.7.2.5 Faster Regional Convolutional Neural Networks (Faster R-CNN)

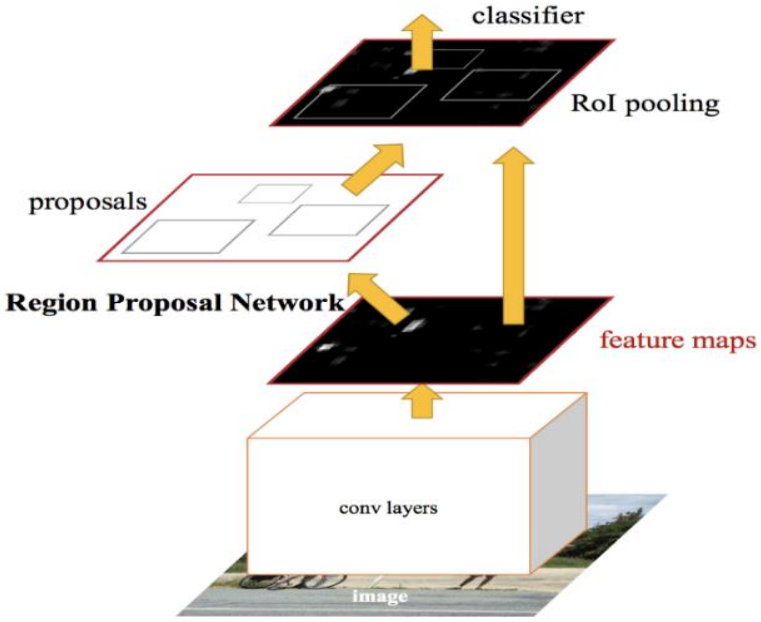

Figure 2.27: Faster R-CNN Architecture

Both R-CNN and Fast R-CNN have high computational speed due to the use of selective search algorithm to find the region proposals, as this algorithm is slow and time-consuming. For this exact reason, Shaoqing Ren (Ren et al., 2016) managed to create a model that is faster than both the previous ones and so called it Faster R-CNN. As in the Fast R-CNN, the whole image is passed through the CNN and a convolutional feature map is created. Although instead of using the selective search algorithm to produce the region proposals he assigned a separate region proposal network to do this exact process. Then the candidates are reshaped, with a RoI pooling layer and passed to a Fully Connected Layer for classification. The architecture is illustrated in the Figure 2.27. Even though this model is known for its speed it faced some challenges regarding the accuracy of the bounding boxes that were outputted.

2.7.2.6 You Only Look Once (YOLO)

All the algorithms that were presented above perform two-stage object detection. With these methods, object detection is divided into 2 stages. The first one is the detection of all the regions that have a higher probability to contain an object, and then the classification of the image takes into consideration the regions proposed in the previous step. Therefore these networks don't process all the image but only a piece of it. YOLO algorithm is much different than the ones above as a single CNN predicts the bounding boxes and the probabilities of each one.

YOLO was created in 2016 by J. Redmon (Redmon et al., 2016) in order to limit the error of the bounding boxes of the previous method. The idea behind YOLO is to split the image into an SxS grid. For each grid cell, the network does the classification and detection outputting a class probability and the values of the bounding box. The cell that has the center of the bounding box for a specific object is the one that is responsible for the object. The bounding boxes that have a probability score above a threshold are the ones that are picked by the network and used to find the objects in the image. This method has high speed and high accuracy, with less background loss that the other methods so it can easily be used in real time applications. Although it can't easily detect multiple objects in a single cell and sometimes it might detect an object twice. Although there are methods that are used to reduce these problems of YOLO.

To understand the way YOLO works, the most significant terminologies used in the algorithms perceptive will be analyzed.

Bounding box

YOLO algorithm predicts the bounding boxes for all the objects that lie in the image and the class to that they belong. The name bounding box refers to a rectangular box that contains an object in it. Each bounding box can be defined with 4 descriptors. Its center, b_x and b_y, its width b_w, its height b_h, and the number c that corresponds to its class. A class is every type of object that is considered in the problem. For example, in autonomous cars, the classes are cars, trees, pedestrians, animals, and so on. Each class comes with a probability, p_c, showing how probable is an object being in the bounding box. An example of a bounding box is presented in the Figure 2.28 below. Each cell predicts B bounding boxes that consist of 5 parameters and the class probabilities for the C classes. Taking into consideration the SxS grid, the total output of model parameters of the YOLO will be S*S*(5*B+C).

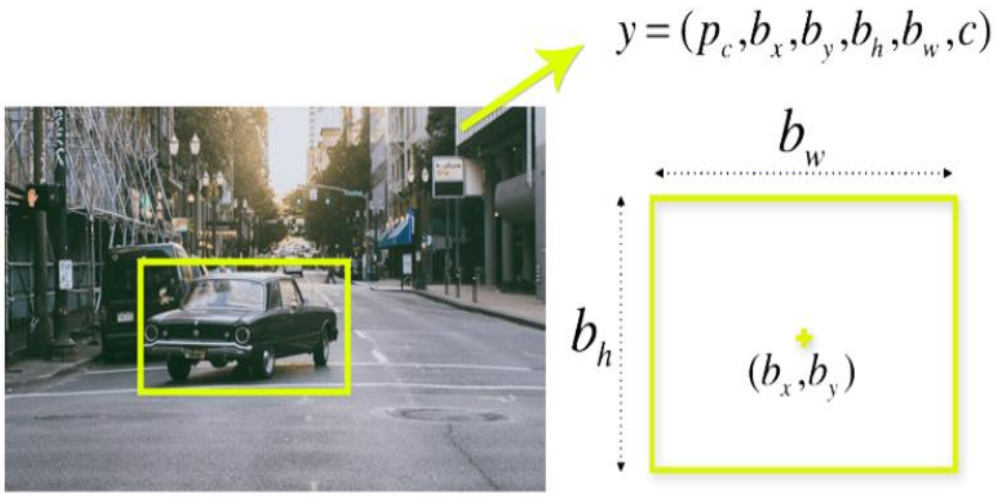

Figure 2.28: Illustration of a bounding box.

As is already discussed YOLO doesn't search the image for a specific region of high importance, but instead, it splits the image into cells and each cell is responsible for a number of bounding boxes. Being responsible for a bounding box means that the center of the bounding box lies in the cell. That is the reason that the coordinates of the center of the bounding box are relative to the responsible cell. The weight and the height on the other hand are relative to the whole image.

The architecture of the YOLOv1 (Redmon et al., 2016) uses the architecture of Darknet that processes all the features extracted and is followed by 2 fully connected layers that make the predictions of the bounding boxes. The author uses grid S=7, B=2 bounding boxes, and 20 classes C=20, so the output is of dimensions S*S*(5*B+C)=7*7*30. The architecture is shown below in the Figure 2.29.

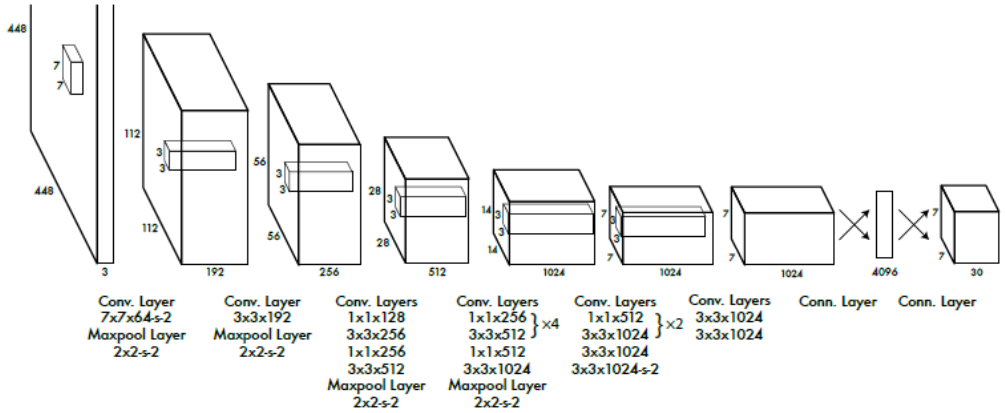

Figure 2.29: YOLOv1Architecture.

Intersection over Union (IoU)

The problem that emerges here is that the number of bounding boxes for each class will be very high when the program needs only one. To eliminate the bounding boxes that are less representative of the object, the Intersection over Union (IoU) method is used (Zhu et al., 2020). In this method, all the bounding boxes that were created are compared to the correct bounding box in the training set, called ground truth. From that comparison, 2 areas will be outputted. The area where the 2 bounding boxes intersect and the union area. The intersection area is divided by the union area and the IoU number is calculated, as illustrated in the following Figure 2.30 where the B2 is the predicted bounding box and the B1 is the ground truth. If the IoU is 0 then the 2 bounding boxes don't intersect and if it is 1 then the prediction is perfect. Although IoU number of 1 is never reached in practice. To have a way to tell whether a bounding box is good enough the ones that have an IoU greater than 0.5 are considered descent, the ones greater than 0.7 are considered pretty good and the ones with IoU greater than 0.9 are considered almost perfect Let's specify that this operation is done for the bounding boxes of a single class and is done for all the classes. After all the IoUs are calculated the bounding boxes that have an IoU number lower than a threshold, determined by the programmer, are discarded. This way the number of bounding boxes is significantly reduced.

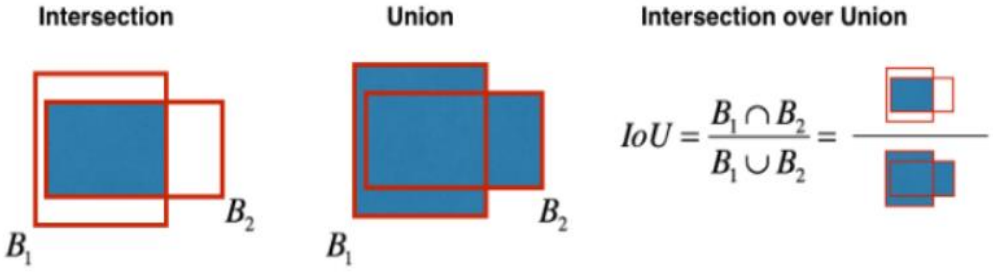

Figure 2.30: Calculation of IoU number.

Non-max Suppression

Even though the above method reduces the number of bounding boxes that refer to a certain class, the algorithm in the end has to output only one bounding box for an object. This is where Non-Max Suppression comes out. Non-max suppression is a method used to eliminate all the bounding boxes that predict the same object. This is done by firstly taking the bounding box that has the higher probability and this is then compared with the other bounding boxes of the same class. The IoU number between the 2 bounding boxes is calculated and the boxes that

have an IoU number higher than a threshold are discarded. In the following figure (Figure 2.31) after the IoU is applied, there are 3 bounding boxes left that represent the same object. After applying the Non-max Suppression there is only one left as the others have an IoU, with the one with the highest probability score, higher than a threshold.

Figure 2.31: Non-max suppression

Let's now summarize the way YOLO classifies which bounding box corresponds better for the objects. The network predicts some bounding boxes. The ones that have a probability score lower than a threshold are discarded. Then the ones whose IoU, with the correct bounding box from the training set, is lower than a threshold are discarded. Lastly, the ones that have an IoU, with the predicted bounding box of the highest probability, higher than a threshold are also discarded and there is only one bounding box left that corresponds to the object.

2.7.2.7 Single Shot MultiBox Detector (SSD) and Region-based Fully Convolutional Network (RFCN)

In order to overcome the problems of YOLO Angelov D. (Liu et al., 2016) proposed the single-shot mutlibox detector, or SSD, that could detect objects of different scales. These objects could be predicted as boxes of different scales are passed to the different layers of the CNN, so every layer could predict objects of different scales. Although it worked well on bigger objects it didn't predict smaller objects that well, as it didn't produce higher-level features. The architecture of this method is presented in the following Figure 2.32.

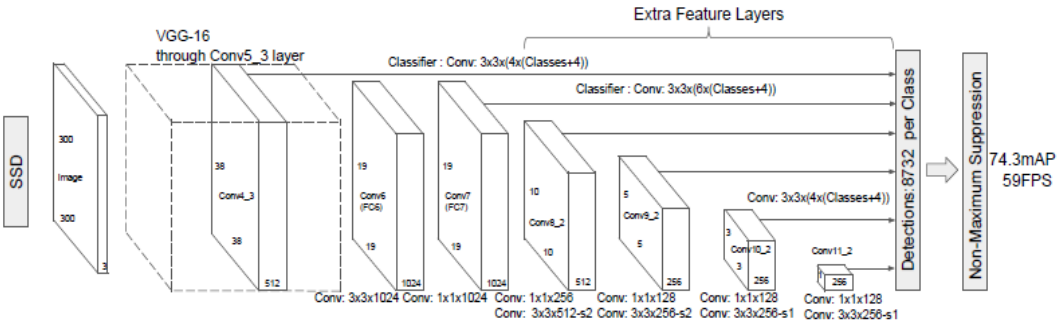

Figure 2.32: SSD architecture

Also in 2016, a Region-based Fully Convolutional Network was proposed by Jifeng Dai (Dai et al., 2016). For it to overcome the problems of SSD and to improve accuracy in general it shared the results, so it was better than RCNN and fast RCNN even though it used them as a base architecture. It made training simpler, reducing complexity and also increasing accuracy. Its architecture is shown below.

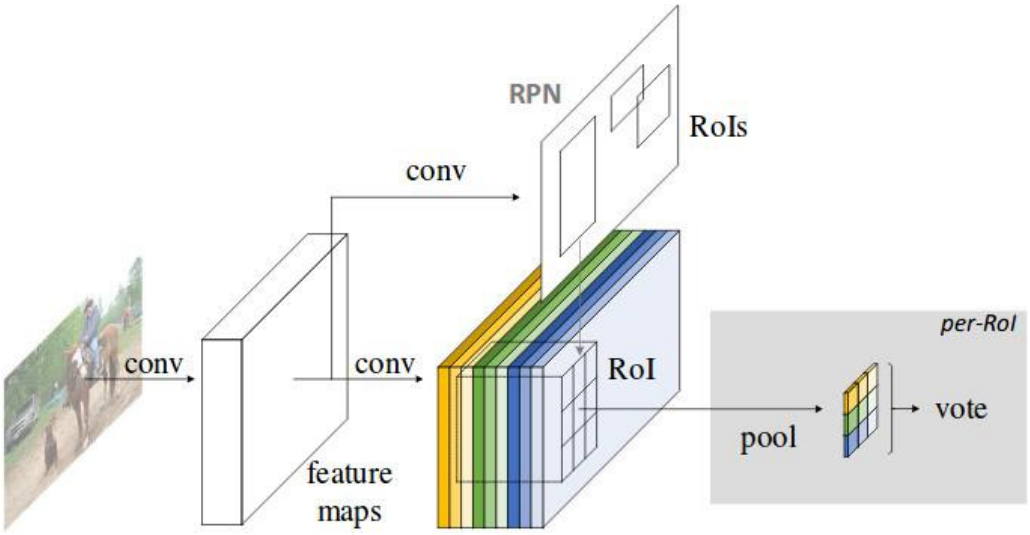

Figure 2.33: RFCN architecture

2.7.3 YOLO Versions

Until now the discussion on YOLO pertained to the release version of the algorithm, version 1. Although, today 5 versions of the YOLO algorithm have been released. Each version was just an upgrade of the previous one as it used the most modern and advanced ideas from the computer vision research community. Except for adding new ideas to the model, the old ones that were not performing so well were removed in order to reach the best accuracy and speed possible. This way YOLO is one of the best algorithms for object detection. Before using the YOLOv5 model, it seems necessary that a representation of the older version must be done and especially in the new ideas that were introduced in them.

2.7.3.1 YOLOv2

YOLOv2 was released in 2016 by Joseph Redmon and Ali Fahradi (Redmon & Farhadi, 2016), the creators of the YOLOv1. It was named 9000 in the original article, which name was given because it could predict over 9000 different objects and still run in real-time. The different features introduced here are mentioned below.

Batch Normalization

Batch normalization is a very common technique for deep neural networks that helps in accelerating the training and making it more steady by stabilizing the spread of the input layers. This idea normalizes the features, outputted from each layer, with the empirical mean and variance of each mini-batch being:

$$\mu = \frac{1}{m}\sum_{i=1}^{m} x_i \quad and \quad \sigma^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu)^2 \tag{3.32}$$

Each layer's dimension, represented with k, is normalized separately:

$$\hat{x}_i^{(k)} = \frac{x_i^{(k)} - \mu^{(k)}}{\sqrt{\sigma^{(k)^2} + \varepsilon}} \tag{3.33}$$

, where ε is added for numerical stability. The result has a mean of 0 and a variance of 1 if the ε is not taken into account. To restore that, the following transformation is introduced:

$$\bar{x}_i^{(k)} = \gamma^{(k)} \hat{x}_i^{(k)} + \beta^{(k)}$$

, where γ and β are parameters that are learned in the optimization process.

With the use of batch normalization, the training time is reduced, and also the mean Average Precision (mAP), which is a metric that will be discussed in the following chapter, increased by 2% according to the authors (Redmon & Farhadi, 2016).

High-Resolution Classifier

In version 1 the 20 first convolutional layers were trained on the 224x224 image and the feature extraction was done. Then the 4 extra convolutional layers and the 2 fully connected were trained on the 448x448 image. The thing that changed is that after the 20 first layers were trained with the 224x224 image the training continued for 10 more epochs in the 448x448 image. This way the transition was better and the overall accuracy in mAP increased by 4%.

Convolutional with anchor boxes

YOLOv1 uses a grid cell to be responsible for a bounding box. Although this way a grid cell can't be assigned to 2 bounding boxes if 2 objects are centered in the same cell. To solve this problem, in YOLOv2 the concept of anchor boxes was introduced. An anchor box is a list of boxes that match the objects. The bounding boxes were predicted not only from the ground truths, as in version 1, but also from the anchor boxes.

YOLOv1 also didn't restrict the bounding boxes that it predicted, so when the parameters are set in the initial step randomly then the boxes can be located away from the desired object. In YOLOv2 the sigmoid function was used to restrict the center of the bounding box between 0 and 1, which helped to locate the bounding box around the grid cell. An image representation is depicted in the Figure 2.34 below, where the sigmoid function is used to restrict the center of the bounding box. pw and ph are the height and the width of the bounding box. cx and cy are the offset position of the grid cell from the top left corner.

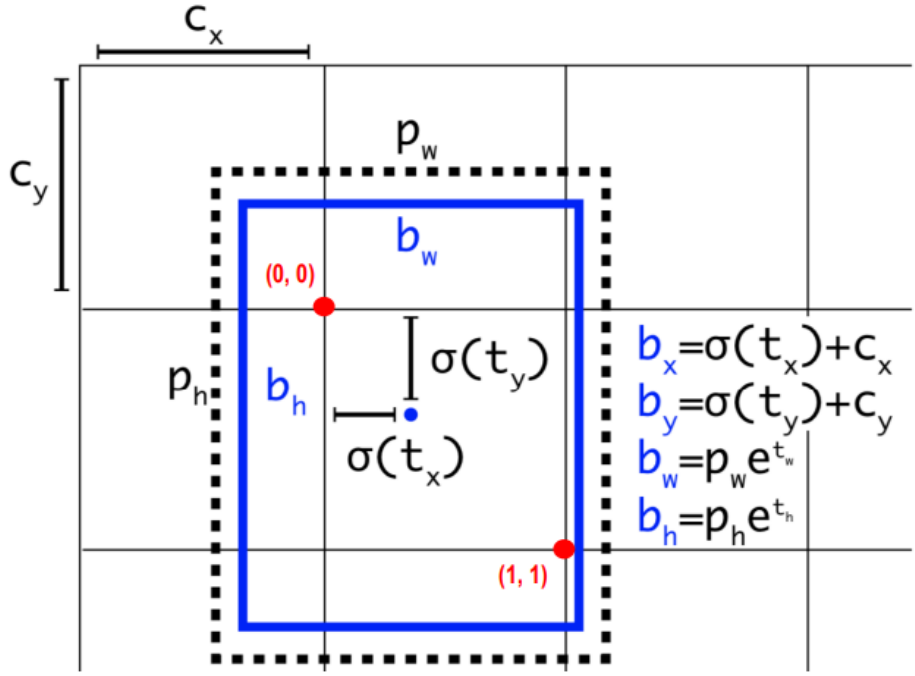

Figure 2.34: Representation of anchor boxes

Since the bounding boxes' positions were restricted close to the grid cell where the object was the parameters were easier to be learned and so the impact to mAP was big, 5% (Redmon & Farhadi, 2016).

2.7.3.2 YOLOv3

The version that changed the architecture of YOLO was released in 2018 by Joseph Redmon and Ali Fahradi (Redmon & Farhadi, 2018). The different features that changed are listed below.

Darknet-53 and ResNet

YOLOv2, even though it increased the number of layers from 19 of YOLOv1 to 30, some features were lost as the image was downscaled passing through the network. For this problem, ResNet introduced a way of skipping some connections to reduce the vanishing features.

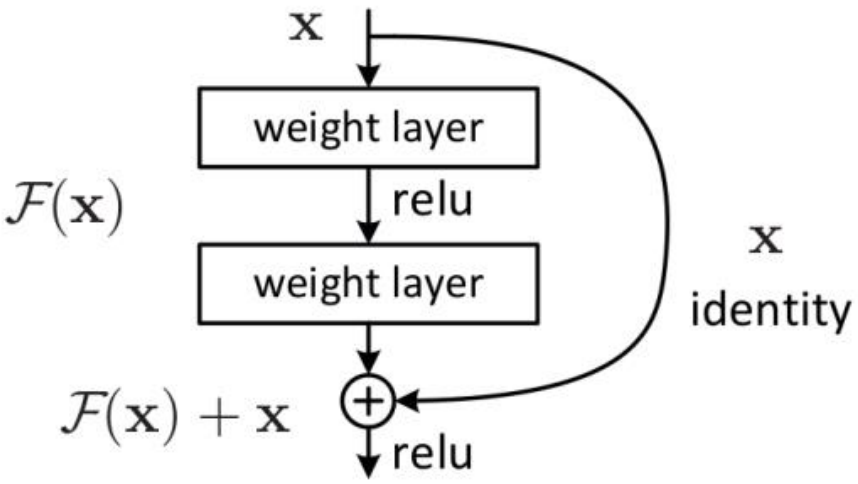

Figure 2.35: ResNet's architecture of skipping connections.

Using this feature and also combining YOLOv2 and DarkNet-53, the YOLOv3 architecture was created using a bottleneck architecture of 1x1 followed by a 3x3 convolutional network in each residual block.

YOLOv3 used DarkNet-53 which had 53 layers in the beginning, as in Figure 2.36. Although after that another 53 layers were added so a 106-layered network was created.

	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Figure 2.36: YOLOv3 architecture

Multi-scale detector

In YOLOv2 the input image is firstly trained on the DarkNet and the features are extracted. Then it goes to some more layers and lastly the prediction is done in the last layer, the object detector. In YOLOv3 the prediction layers are appended and 3 feature vectors are created, which are forwarded to the detector, instead of stacking the prediction layer in the last layer (Redmon & Farhadi, 2018). The architecture of the network is shown in the Figure 2.37 below.

The 3 different detections are done in the 79, 91, and 103 layers where the first has a grid of 13x13, the second 26x26, and the third 52x52. Using these 3 detections helps in detecting the different-sized objects in the image. The higher-sized feature maps, 52x52, are more detailed and are used in detecting the smaller object and the 13x13 is used to detect the larger ones. By concatenating layers that are closer to the start of the network with the ones that have already passed through some layers of the network the fine-grained features from the previous layers don't vanish which helps in detecting small objects. In the Figure 2.37, a concatenation is done before the 91st layer is reached with the 61 and one before the 103rd layer is reached with the 36th layer.

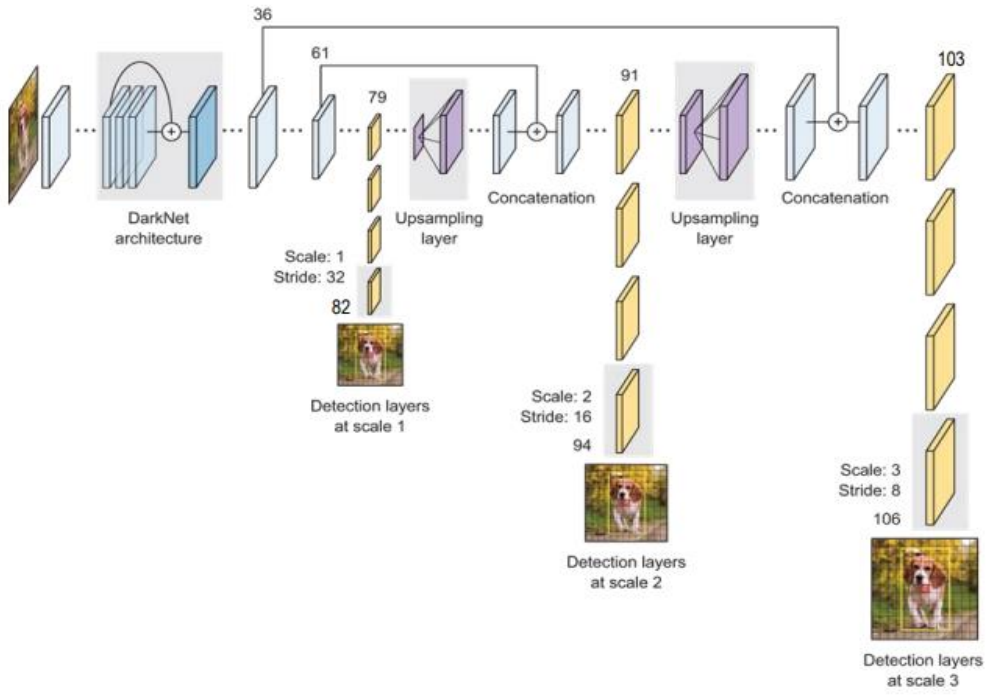

Figure 2.37: YOLOv3 network architecture

2.7.3.3 **YOLOv4**

The fourth version of YOLO was introduced by Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao in 2020 (Bochkovskiy et al., 2020) after Joseph Redmon stopped working in the field of computer vision. This version of YOLO is considered to be the continuation of the YOLO family even though it was created by different scientists. The different features represented in this version are discussed below.

By the time YOLOv4 was introduced many other object detection algorithms had achieved remarkable results. These algorithms had a common point that the input is firstly passed through the backbone, which is the feature extractor, and compress. Then the features are aggregated and mixed in the Detection Neck, or simple Neck, to prepare for the detection done in the Detection Head.

Backbone

In the backbone of the model the CSP Darknet53, CSP stands for Cross Stage Partial, was used. This network consists of k dense layer, where the output of each dense layer is concatenated with its input and then becomes the input for the next layer and the last layer goes to a transition layer where convolution and pooling are done. The only peculiarity is that the input feature map is separated into 2 different pieces where one passes through the dense layers and the other is concatenated in the end at the partial transition layer, as illustrated in the Figure 2.38 below.

With the use of CSP Darknet53 in YOLOv3, the residual blocks were removed and in their place, dense layers were placed. With the help of CSP, the model maintained some fine-grained features and also reused some other features making the learning easier for the model.

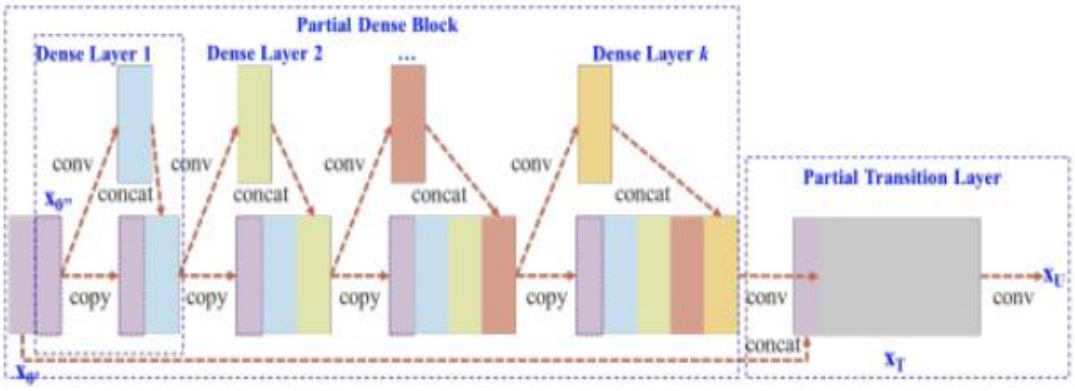

Figure 2.38: Cross Stage Partial DenseNet

Neck

New SPP Block

After the Dense layers, the feature maps have to go through an additional block called Spatial Pyramid Pooling (SPP) block in order to separate the most important features. SPP was discussed before in the 2.7.2.3 section and the architecture here is the same but as the fully connected layers have been removed from YOLOv2 turning the feature maps into a one-dimensional vector wasn't useful. So the new SPP concatenates the features into feature maps of size $size_{feature\ map} * size_{feature\ map} * 512$, after a 3-scale max pooling, and also keeps and includes the input feature map to the output feature map to keep all the information needed (Bochkovskiy et al., 2020). A representation of the new SPP is depicted in the Figure 2.39.

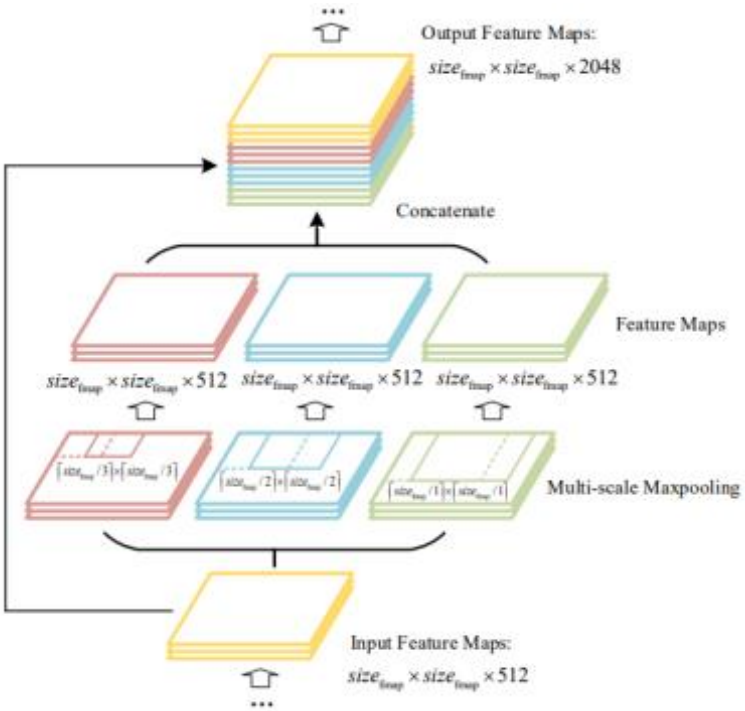

Figure 2.39: The new SPP block used in YOLOv4

Feature Aggregation with PANet

As the input image goes through the model semantic features are produced. These features are becoming more and more complex as the network goes deeper and also their resolution is decreased so some important information might be lost in the process. So to resolve this the Path Aggregation Network (PAN) was introduced (Bochkovskiy et al., 2020). This network is an advanced Feature Pyramid Network (FPN), in which method the semantic features are transferred through a top-down path, and then they are concatenated with the fine-grained features of the low-level layers to better predict smaller objects.

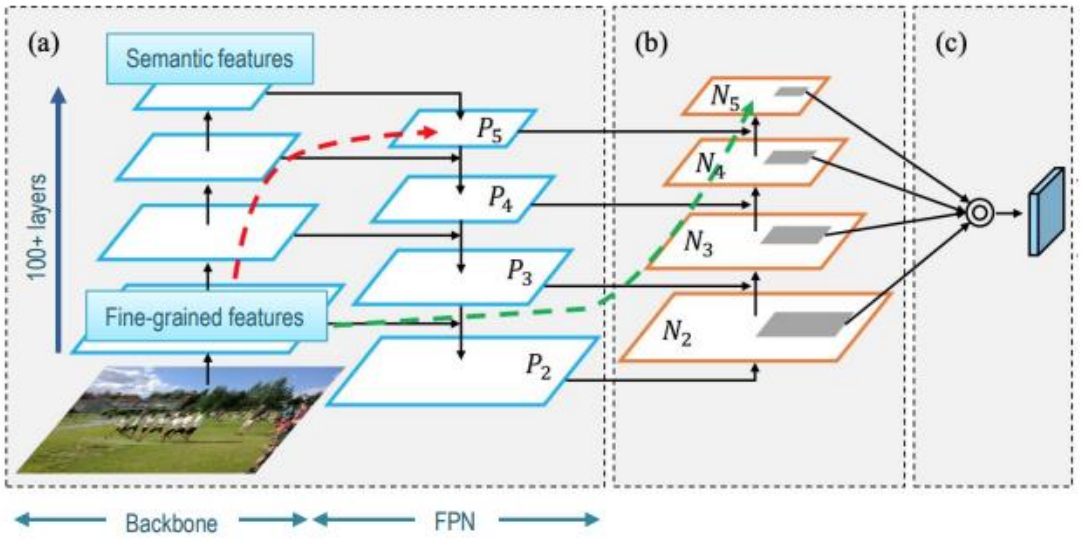

Figure 2.40: PANet architecture (a) FPN backbone, (b) augmentation path, (c) feature pooling

The PAN used in YOLOv4 had an FPN backbone followed by a bottom-up augmentation path that connected the fine-grained features with the semantic features of the high-level layer quicker, without having to pass through 100 layers of the backbone of the model, thus creating a shortcut (Figure 2.40). This bottom-up augmentation path is identical to the FPN with each layer creating feature maps of the same size. The different features produced from the layers are added in the normal PAN but in the YOLOv4 version of it, they are concatenated (Figure 2.41).

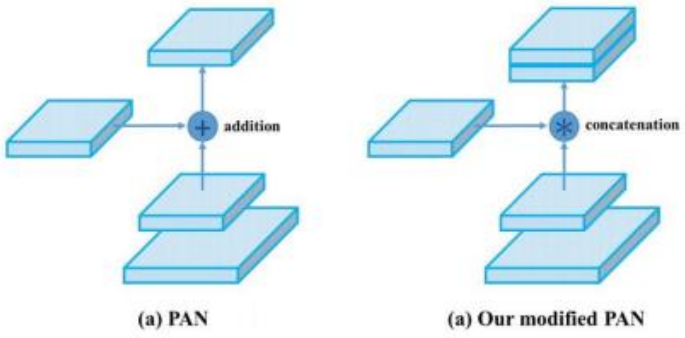

Figure 2.41:(a) original PAN, (b) YOLO v4 version

Head

The head is where all the predictions are done and vectors that contain all the bounding boxes' coordinates, the confidence scores, and class probabilities are produced. In the case of YOLOv4, the head of the algorithm was kept the same as the YOLOv3.

2.7.3.4 **YOLOv5**

Yolov5 was released one month later from YOLOv4 by the researcher Glenn Jocher, who is the CEO of Ultralytics LLC. The creator of YOLOv5 didn't post a paper with the advancements of YOLOv5, but only posted a repository on GitHub (Redmon, 2020). Ultralytics managed to implement the YOLOv5 in PyTorch, which is the most famous framework in deep learning and is written in Python language.

As versions 4 and 5 were released in a very short period of time, only one month (March-April 2020) there are not many differences between the 2 as both of them used the methods that were the most efficient at the time. However, YOLOv5 has some advantages compared to previous versions. Firstly it is written, as mentioned, in Python which is easier to install and integrate on devices, instead of C of the previous ones. Furthermore, it has high accuracy and fast speed, reaching 140 frames per second. Moreover, YOLOv5 is much smaller compared toYOLOv4, it is 90% smaller in fact, which makes it suitable for every device used in object detection. Further explanation of the architecture of the network will be done in the next chapter.

Adaptive anchor boxes

YOLOv5 architecture, as in all the previous versions, has included all the most recent techniques in the computer vision field. In this algorithm, as it was implemented during the same period as the YOLOv4, there are not so many noticeable differences in the methods used.

One point to mention is the adaptable anchor boxes. In YOLOv2 the anchor boxes were extracted by using the k-means clustering algorithm to pick the 5 best anchor boxes for each class from the COCO dataset and use them as a default. Although when these anchor boxes are applied in a unique dataset that doesn't contain the same classes as the COCO dataset then the adaptation of the anchor boxes to match the ground truths is slow. So Joseph Redmon proposed to run the k-means algorithm in the unique dataset and find the best anchor boxes for this set. This way the program automatically learns the best anchor boxes for the specific applications and uses them in the training (Solawetz, 2020).

3 Methods

3.1 YOLOv5s Architecture

YOLOv5 network is the latest version of YOLO algorithm and it was created only 2 years ago, in 2020. It contains 4 different architectures, named YOLOv5s, YOLOv5m, YOLOv5l, and YOLOv5x. The difference between them is the amount of feature extraction modules and the size of convolution kernels used in the model. The size of the network and the parameters in these different networks increase as the network gets larger. In the small version the size and the parameters are 14 MB and 7.2 million respectively, in the medium it is 41 MB and 21.2 million, in the large 89 MB and 45.6 million, and lastly, in the extra-large, it is 166 MB and 86.7 million.

Since the problem considered in this thesis has a small number of classes and images for each class the YOLOv5s model was used in the implementation. Also, it is fast, accurate, and takes up the least space in the device, so it was considered a perfect candidate for this thesis.

YOLOv5s consists of 3 components: backbone, neck, and detect network. The backbone consists of 16 modules, 1 focus, 4 convolutional layers, 8 bottleneckCSP, and 1 SPP.

The first block is the Focus. Here the input 3 channel image, which is in default of size 3x640x640, is divided into 4 slices with a size of 3x320x320 each, to accelerate the training reducing the calculations of the model. Then these slices are concatenated with a concat layer and forwarded to a convolution layer with 32 kernels (Yan et al., 2021). The feature map created has a size of 32x320x320 which goes through a BN, Batch Normalization, with a Hardswish activation function, see 3.1.1, and continues onto the next block, Figure 3.1.

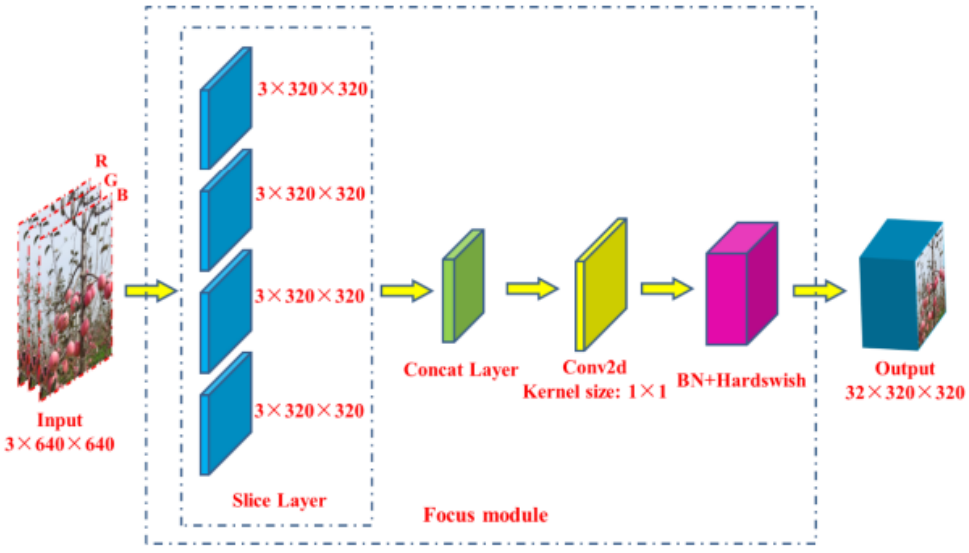

Figure 3.1: Structure of Focus Block

The next block that needs to be explained is the third in the row and is the BottleneckCSP block. This block is more complex as it contains another block inside of it. This block is the Bottleneck that connects a convolution layer of kernel size 1x1, with a BN and a Hardswish activation function, with another of size 3x3. Then the feature map produced by these 2 layers is added to the input feature map of the block, Figure 3.2.

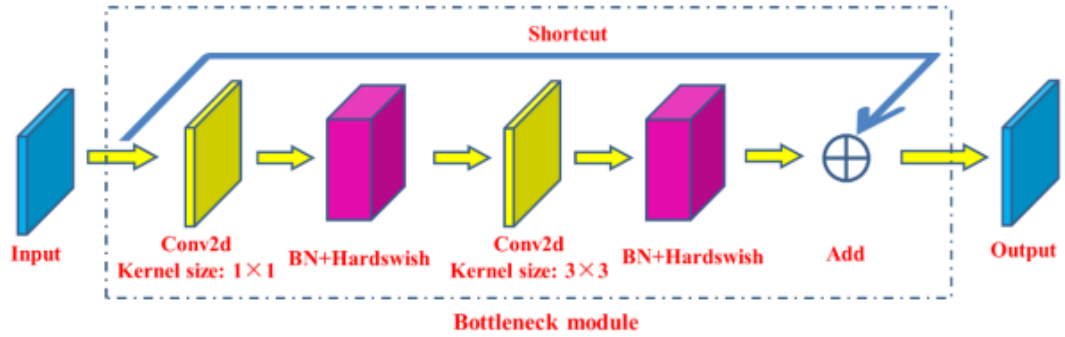

Figure 3.2: Architecture of the Bottleneck block

The input of the BottleneckCSP block is firstly input to 2 different paths. The first passes through a convolution layer, of kernel size 1x1 and a BN with Hardswish. Then it is input in a convolution layer, of size 1x1, after it is output from the bottleneck block. The second feature map passes through the shortcut and then these two feature maps are connected by a concat layer and the output is produced after it passed through the BN with LeakyReLU activation function and a convolution layer (Figure 3.3).

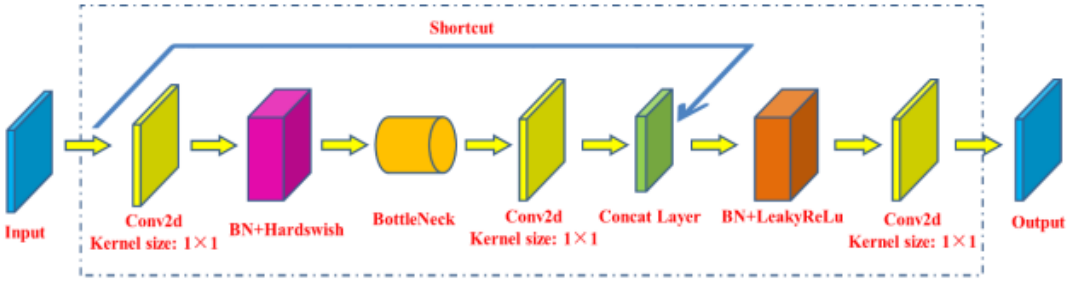

Figure 3.3: Structure of BottleneckCSP block

The ninth block of the Backbone is the Spatial Pyramid Pooling, SPP. As it has been discussed the SPP has a purpose to convert the feature maps into a vector of fixed size, which is of size 512x20x20 for the current model. The feature map passes through a kernel size 1x1 convolutional layer, with a BN and Hardswish, and then it is input into 3 max-pooling layers of size 5x5, 9x9, and 13x13 (Yan et al., 2021). Then the output is concatenated with the output of the BN and Hardswish layer. Lastly, as it goes through a convolutional layer, with kernel size 1x1 and BN and Hardswish, the result is produced (Figure 3.4).

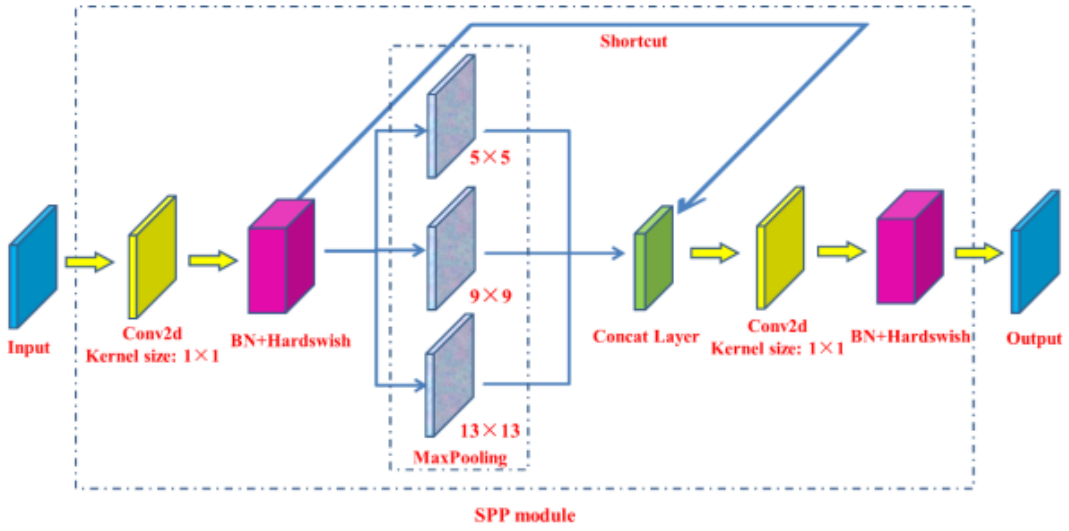

Figure 3.4: SPP block architecture

The neck network of the model mainly consists of feature aggregation layers. The feature maps produced in the neck are combined with several of the backbone but also from the ones created in the neck. It uses many Feature Pyramid Networks, FPN, which were explained in the section 2.7.3.3. This way the fine-grained features are kept and the detection of features of different scales is enhanced (Yan et al., 2021). Upscale blocks increase the scale of the feature map in order to match the one that is connected within the concat layer.

The detect network is where the final detection is done. There, anchor boxes are applied to the feature map of the previous layer and the output of the whole network is produced. The output contains the position and size of the bounding boxes, the class predictions, and the probability. There are 3 detect layers for the network to be able to easier detect objects of different sizes. The input of these layers is feature maps of sizes 20x20, 40x40, and 80x80 respectively. Each detect layer outputs a 24-channel vector in the specific application, that consists of 3 classes, 1 class probability, and 4 bounding box coordinates. These 8 numbers are for each anchor box. Taking into consideration that the anchor boxes produced in YOLOv5s are 3 then the number of channels in the output vector is 24. Lastly, the bounding boxes in the image are generated and labeled. An overview of the network is illustrated in the Figure 3.5 below.

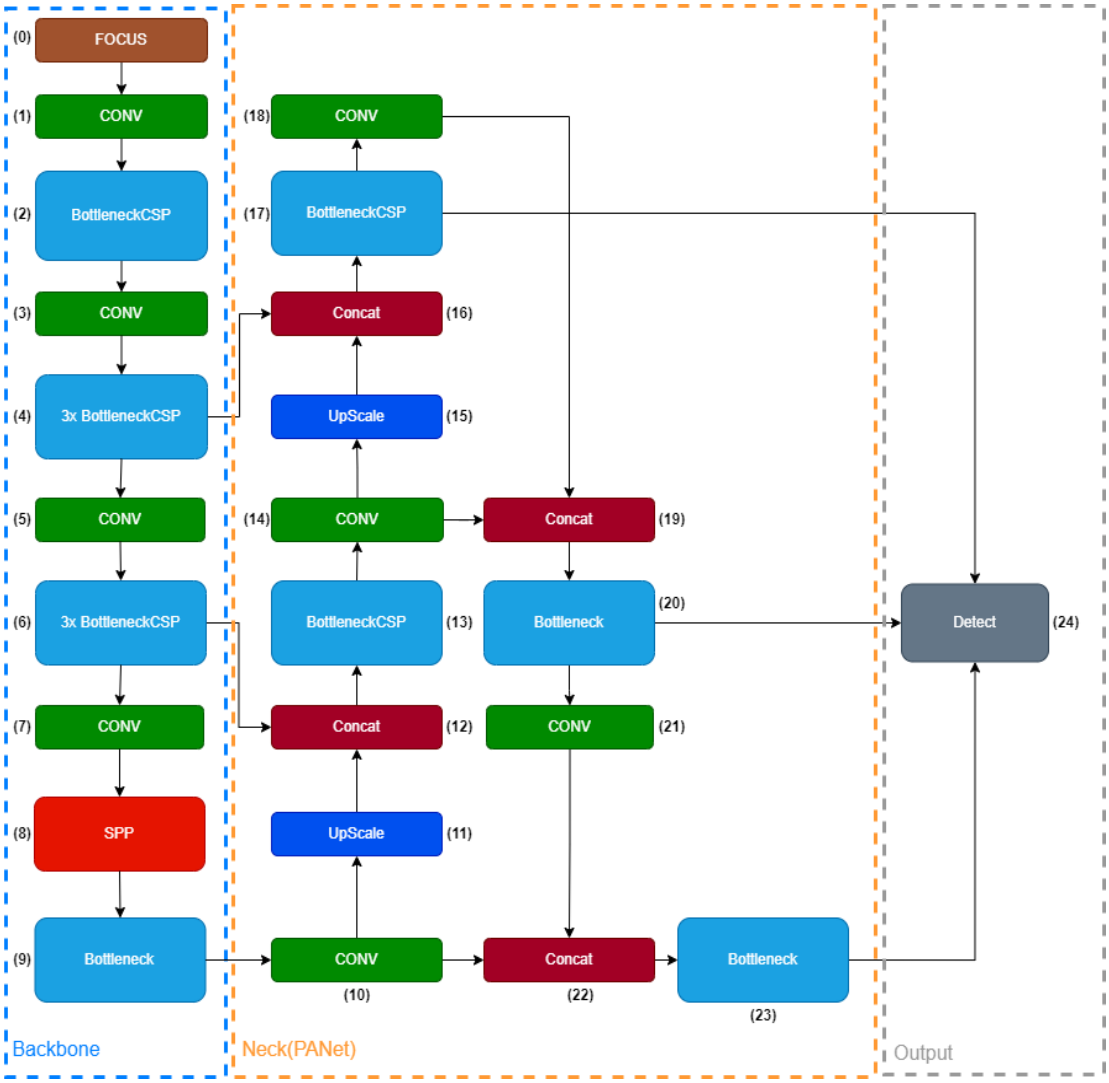

Figure 3.5:YOLOv5s Architecture

3.1.1 Hardswish activation function

The Hardswish activation function is a function that applies the Hardswish transformation (Pytorch):

$$Hardswish(x) = \begin{cases} 0 & if\ x \leq -3 \\ x & if\ x \geq +3 \\ \dfrac{x * (x+3)}{6} & otherwise \end{cases}$$

This function is based on Swish activation function, $f(x) = x * sigmoid(\beta x)$, but replaces the computationally expensive sigmoid function with a linear analog. The swish function is a smooth, non-monotonic, function that matches or outperforms ReLU applications of deep neural networks such as image classification. In general, swish is better than ReLU for networks with more than 40 layers and on every batch size. Hardswish managed not only to keep all the advantages of the Swish activation function but also to require fewer computations as it removed the sigmoid function.

3.1.2 Metrics

In object detection, the main purpose is to detect all the objects in the image that they are presented with. They do so by placing bounding boxes around the objects. Bounding boxes

have 3 attributes: the location of the center of and the dimensions of the bounding box, the object class, and the probability score which is how confident the algorithm is of the prediction with values from 0 to 1. There are 2 different sets of bounding boxes:

- A set of ground truth bounding boxes, which are the ones given to the model in the training dataset,
- A set of predicted bounding boxes, which are the ones that the object outputs as his predictions.

Now the main metrics of the model will be presented in order to better understand the results that follow.

3.1.2.1 Precision and Recall

It is obvious that not every bounding box that the model predicted is correct. Let's start by classifying the bounding boxes. Each of them can be categorized in one of the following categories:

- True Positive (TP), which is a correct prediction of a ground truth bounding box.
- False Positive (FP), which is an incorrect detection of a non-existing object or a misplaced detection of an existing object.
- True Negative (TN), which is a non-existing bounding box that was not predicted.
- False Negative (FN), which is a ground truth bounding box that was not detected.

In object detection, the TN bounding boxes are not used as there are many bounding boxes that should not be predicted in an image. Correct and incorrect predictions are classified by the IoU that was introduced in Chapter 2.7.2.6. After the classification is done the Precision and Recall can be calculated.

Precision (P) is the ability of the model to predict only relevant objects and is the percentage of the correct predictions. Recall (R) is the ability of the model to predict all the ground truth bounding boxes and is the percentage of the correct prediction among the ground truth bounding boxes (Padilla et al., 2020). These metrics are defined as,

$$P = \frac{\sum TP}{\sum TP + \sum FP} = \frac{\sum TP}{all\ detections}$$

$$R = \frac{\sum TP}{\sum TP + \sum FN} = \frac{\sum TP}{all\ ground\ truths}$$

3.1.2.2 Average Precision

As discussed before, the confidence level is taken into consideration when calculating Precision and Recall. The predictions that have a confidence score higher than the confidence threshold, with value t, can be considered positive, and respectively the ones with a lower score negative. Both TP and FP decrease as t increases, so fewer detections pass the threshold. On the contrary, FN will increase as more detections will be less than the threshold. However the number of all ground truths, $\sum TP + \sum FN$, will stay the same as t alters. Thus R is a decreasing function of t, but for the P nothing can't be said as both the numerator and denominator are decreasing as t increases. It becomes obvious that the graph of Precision and Recall has a zig-zag form (Padilla et al., 2021).

A good object detector should find all the ground truths, namely FN=0 which means high recall, while finding all the relative objects, FP=0 which means high precision. Therefore an object detector is good if the recall and precision remain high as the t decreases. High recall

and precision are indicated by a large area under the curve (AUC). Although this area is difficult to calculate as it is not monotonic, as described above, having a zig-zag form. To eliminate this problem the data are preprocessed. The Average Precision, AP, is the area under the curve of those preprocessed data.

Image Detection	Confidence Score	TP or FP	Precision	Recall	Maximum Precision for any Recall r´≥r
1	0.92	TP	1/1	1/3	1
3	0.85	FP	1/2	1/3	
2	0.83	TP	2/3	2/3	3/4
4	0.75	TP	3/4	3/3	3/4
5	0.72	FP	3/5	3/3	

Figure 3.6: Detections are ranked depending on their confidence score and the way precision and recall are calculated for each.

Firstly, to compute the AP the predicted bounding boxes are ranked according to their confidence scores, from the highest to the lowest and then the recall and precision for all the predictions are calculated. An example of these processes for the P and R is shown below (Padilla et al., 2020)Figure 3.6: Detections are ranked depending on their confidence score and the way precision and recall are calculated for each. (Figure 3.6). After that, the AP can be calculated by the below Equation 3.1,

$$AP = \sum_{n} (R_{n+1} - R_n) * P_{interp}(R_{n+1}) \tag{3.1}$$

, where

$$P_{interp}(R_{n+1}) = \max_{\tilde{R}:\tilde{R} \geq R_{n+1}} P(\tilde{R}) \tag{3.2}$$

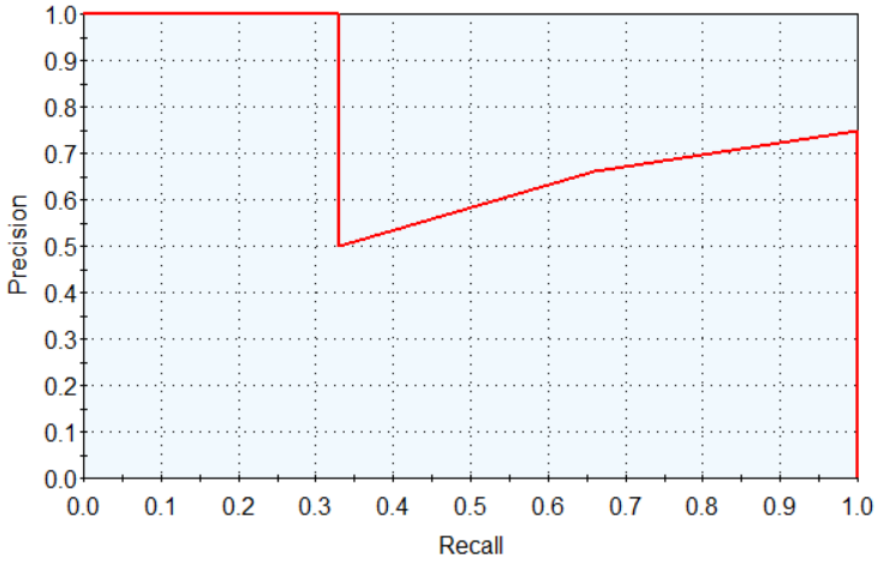

Figure 3.7: Precision-Recall Curve before applying the preprocessed data.

In this method, AP is calculated using the maximum precision values whose recall value is greater than R_{n+1} or equal to it. Considering the above example the AP should be the area below the curve of the Figure 3.7, $AP = \frac{1}{3} + \frac{\left(\frac{2}{3}+\frac{1}{2}\right)*\frac{1}{3}}{2} + \frac{\left(\frac{3}{4}+\frac{2}{3}\right)*\frac{1}{3}}{2} \approx 0.764$. Although after applying the Equations 3.1 and 3.2 the Precision and Recall curve turns into the one displayed in Figure 3.8 in gold color. The Precision values taken were only 2, for the first two recalls the value of precision was 1 as it was the maximum of 1 and 1/2. Then for the other values, the

precision value was 3/4, as it was the maximum between 2/3 and 3/4 and also the maximum of 3/4 and 3/5. So finally the value for AP is 0.833.

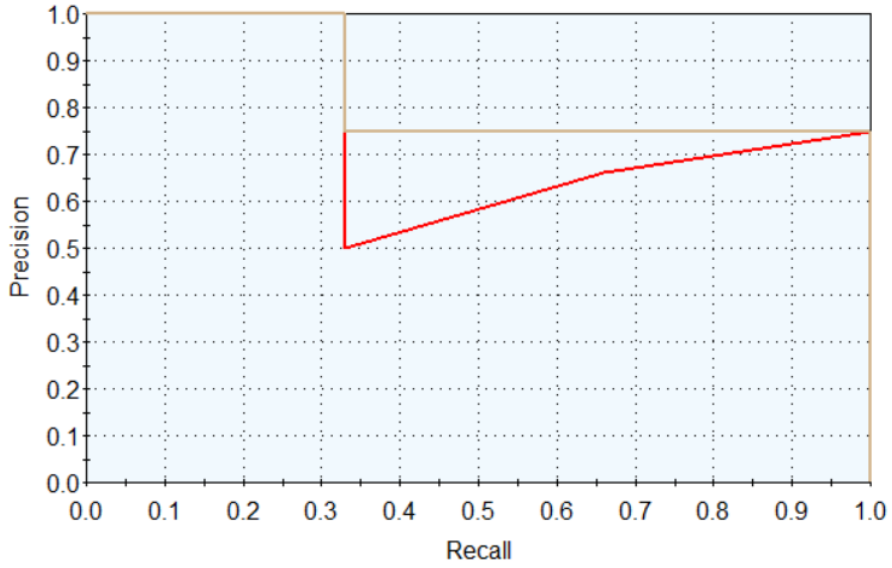

Figure 3.8: Precision-Recall Curve before applying the preprocessed data, with red color, and after applying the preprocessed data with gold.

3.1.2.3 Mean Average Precision

Average Precision is calculated individually for each class. In most object detection applications, although, the number of classes is surely more than one. In these cases, the mAP is a useful metric that is the average AP of all the classes in the problem. The below Equation (3.3) shows how to compute it,

$$mAP = \frac{1}{C} * \sum_{i=1}^{C} AP_i \tag{3.3}$$

, where AP_i is the AP computed for the i class with a total of C classes.

In some cases, the mAP can be seen in the form of $mAP_{0.5}$ or some other values for the index. This index shows the IoU threshold that the AP was calculated on. Another way that this metric can be found is $maP_{0.5:0.95}$, which means that the mAP is calculated for all the IoU values between 0.5 and 0.95 with a step of 0.05 and the average is calculated.

3.1.2.4 Loss Function

The Loss function used in YOLOv5 is the Binary CrossEntropy With Logits Loss (BCEWithLogitsLoss). Firstly the Binary CrossEntropy Loss (BCELoss) will be explained. Binary CrossEntropy is the negative average of the log of corrected predicted probabilities. What that means is that it compares the predicted probabilities of the model for each sample with the actual class that they belong to and then calculates how far from the actual value the predicted is, which is 0 or 1. The mathematical Equation of this loss is shown below, Equation (3.4):

$$l(x, y) = -\frac{1}{N}\sum_{i=1}^{N} y_i * \log(p_i) + (1 - y_i) * \log(1 - p_i) \quad (3.4)$$

, where:

N is the number of samples,

p_i is the probability of class 1

y_i is the actual class that the object belongs to

This Equation calculates the loss when there is only one class. What changes when there are multiple classes is that the sum is done for all the classes in the problem. In the case of the BCEWithLogits the sigmoid function, which is explained in previous sections, is combined with the above BCE in order to create better numerical stability. The Equation is shown below (Equation 3.5).

$$l(x, y) = -\frac{1}{N}\sum_{i=1}^{N} y_i * \log(\sigma(p_i)) + (1 - y_i) * \log(\sigma(1 - p_i)) \quad (3.5)$$

, where σ is the sigmoid function.

3.1.3 Scenarios

In this thesis, the main purpose is to recognize all the objects that a ship will meet in its path. For this reason, some of the objects are considered as the classes of the problem. These classes are ships, as these are the most common object in collision avoidance, rocks, and floating objects. As floating objects are considered humans, containers, and buoys.

The different scenarios that are taken into consideration in this thesis are:

1. Images with only one class of objects.
2. Images containing at least 2 of the classes of the objects.
3. Testing in a real-time video.

In the first case, the network was trained with only one class of objects. Firstly the network was trained to recognize ships. For this reason, it was input with 20 images of ships, to acquire a first impression of the network. In these images, there are only images that contain only one ship in them. Then, 50 images of the ship were imported, with some of them being photos of more than one ship in them, to see how good it comprehends with many objects of the same class. Lastly, the input was 100 images and the impact on the confidence score of the system and the accuracy was inspected, in comparison to the 50. The same was done for the other 2 classes, where some images of the objects in the night were also considered. In every case, 90% of the data are for training and 10% are for validation.

In the second case, the model was trained on images with a combination of the objects. A class is rarely alone in an image when dealing with real-life problems so in this case firstly 50 images of all the object classes, or at least 2 of the classes, being in every one of them were collected. Secondly, the 100 images of ships, 100 of rocks, and 100 of floating objects are added all together as the training set to observe the model's performance in determining what the class of the individual objects is. After that, the model was trained on 50 images of ships at night and in fog where the vision is very obscured. Lastly, all the photos that were collected, including

the 50 that had all or at least 2 classes in them and also 50 images of ships captured at night, with a total of 400 photos were input into the model.

In the third case, a video of ships in the sea was found and the model was tested to determine the frames per second that it can get and so if it can be applied in real-time detections.

3.1.4 Image Data Acquisition

In order for the network to work in the best way possible the images that it is trained on must be very accurate in what they represent. For that to be done, images recorded by a camera placed on the bridge of a ship were necessary. Although that wasn't possible in this thesis, so images containing the classes needed from the internet were carefully picked. An example of images is shown in the Figure 3.9 below.

Figure 3.9: Example of images used in the training dataset.

After the images were collected, they had to be labeled. This was done with a program called labelimg, which was obtained by the repository of the program from GitHub (Tzutalin, 2015). In this program, the images are selected and the user can create a square that contains the desired object and also the class that the object belongs to (Figure 3.10). Then a text file, that contains the position of the center of the bounding boxes of the image, their length, their height, and the class every box belongs to, is created (Figure 3.11).

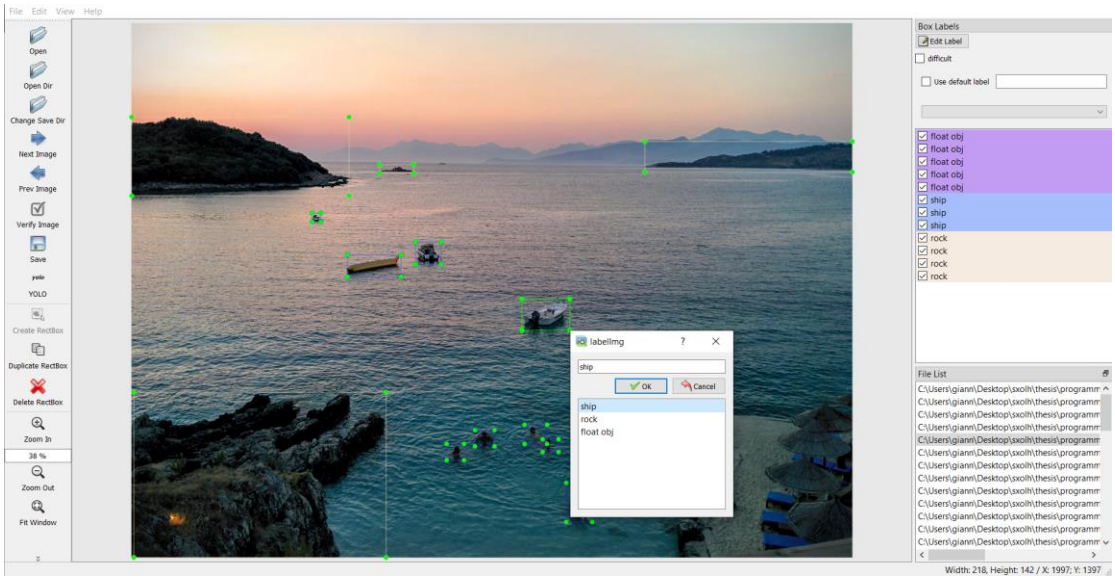

Figure 3.10: An example of labeling in labelimg.

Αρχείο Επεξεργασία Μορφή Προβολή Βοήθεια

```
1 0.699545 0.602442 0.184545 0.268657
1 0.826818 0.712347 0.173636 0.268657
1 0.797727 0.582768 0.042727 0.145183
1 0.974091 0.862958 0.051818 0.274084
0 0.375455 0.733379 0.049091 0.028494
0 0.237273 0.563772 0.089091 0.191316
0 0.184545 0.626187 0.030909 0.017639
```

Figure 3.11: An example of a .txt file that is output from the labelimg.

After the labeling was done, the images had to be put in folders. For the model to run the position of the folder containing the training data, the labels, the validation data, and also the number of classes must be determined. This is done by a YAML file, which is a data serialization language that is often used for writing configuration files, inside the model where these exact positions are stated (Figure 3.12). After these processes were completed the model was run in a jupyter notebook with python using the PyTorch framework.

dataset.yml

C: > Users > giann > Desktop > sxolh > thesis > programms > object_detection > yolov5 > dataset.yml

```yaml
# Train/val/test sets as 1) dir: path/to/imgs, 2) file: path/to/imgs.txt, or 3) list: [path/to/imgs1, path/to/imgs2, ..]
path: ../data # dataset root dir
train: images # train images (relative to 'path')
val: val_images # val images (relative to 'path')
test:  # test images (optional)

# Classes
nc: 6  # number of classes
names: [ 'ship', 'rock', 'float obj','merchant','passenger','tanker' ]  # class names
```

Figure 3.12: An example of the YAML file that the program needs to operate.

4 Results

Now that the theoretical part that refers to the artificial neural networks, the CNNs, and the object detection with the models that exist for solving such problems is covered it is about time that the results of the YOLOv5s model are represented. There won't be a commented representation in every one of the cases. The results of the scenarios not commented on in the next chapter will be cited in the Appendix B: Results of different cases at the end of the thesis.

4.1 Ship images

4.1.1 20 images in the training dataset

Firstly the model was trained for 300 epochs, with a batch size of 8, without early stopping and the input of the model was 20 images of ships. The results of the metrics are shown in the Figure 4.1 below.

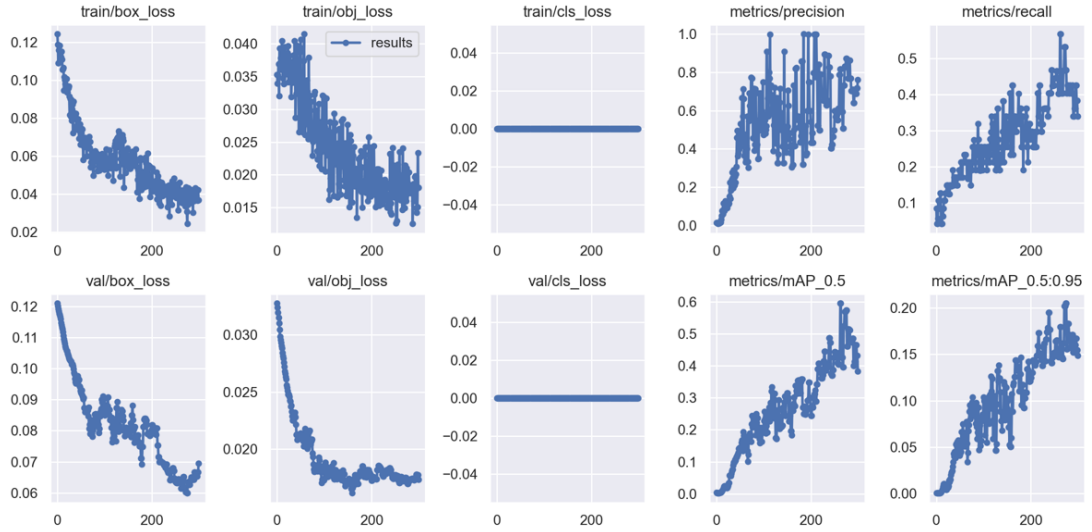

Figure 4.1: Metrics results for 20 images in the training dataset.

Box_loss is the bounding box regression loss, using the mean square error loss function. Obj_loss is the confidence of object present loss and is calculated using the Binary CrossEntropy, while the cls_loss is the classification loss using the Cross-Entropy loss function.

As it is observed the precision reached a value of 70% on average, while recall and mAP had a maximum value of 55% and 60% respectively. This is expected as the number of images that the model was trained on was significantly low. On the contrary, the loss functions reached low values, with the box_loss in the training and validation dataset being 0.02 and 0.06 respectively while the training and validation obj_loss were 0.015 and 0.018 respectively. Lastly, the training and validation cls_loss is 0 as only 1 class is considered in these first scenarios so the model is not provided with different classes to classify. The metrics also have some variation between the epochs, which is a result of the low number of images in the training dataset. The reason that this such a scenario was done is to comment on the effect of the number of images in the model with the same number of epochs. The Figure 4.2 contains predictions of the model on some of the test data.

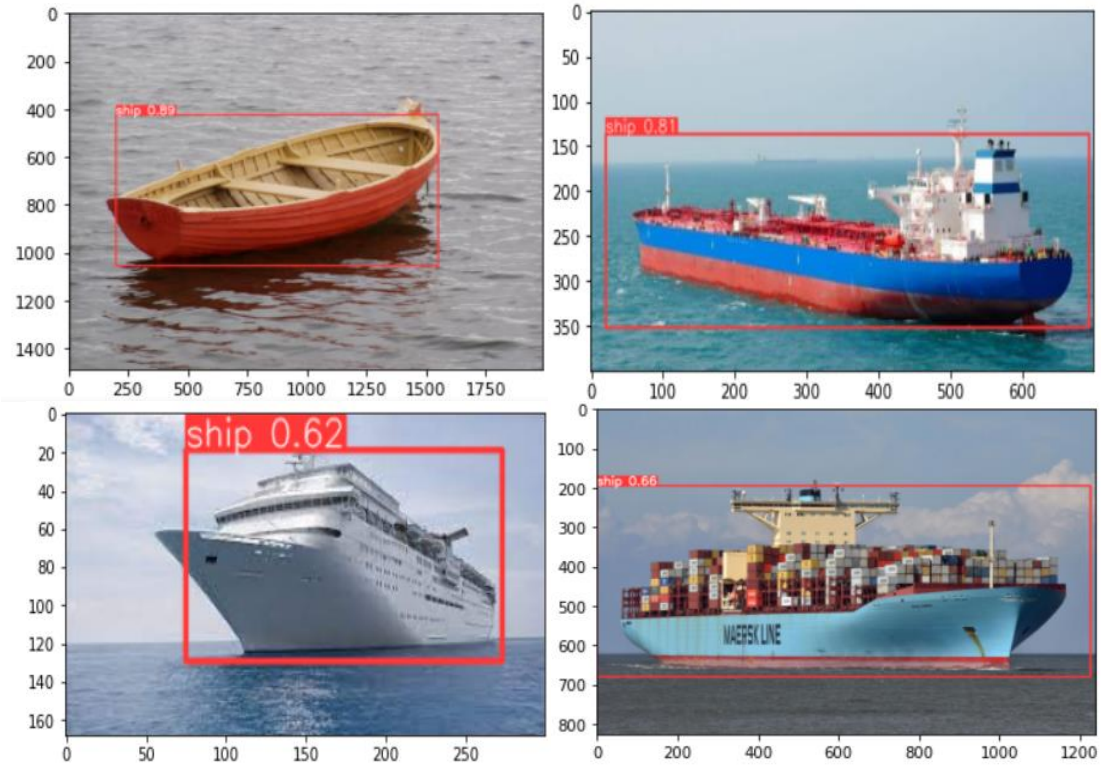

Figure 4.2: Results of the model detection in the test dataset, with 20 images of ship in the training dataset.

In the images, the number refers to the confidence of the network that the bounding box contains an object of the mentioned class. The model in general returned good results for images that weren't in the training and validation dataset, with 89% confidence for the top left image, 81% for the top right and 62%, and 66% for the bottom left and right respectively. Lastly, it can be observed that the bounding boxes of the different ships are larger than the ship they contain, meaning they are not that accurate. This is an effect of the small dataset.

It must be mentioned that the test dataset, and especially these 4 images, will stay the same in the next scenarios so that a comparison between the different training dataset and their effect on the model's predictions will be done.

4.1.2 50 images in the training dataset

In the next case, 50 images were presented in the model as the training dataset. Below the metrics of the model are presented (Figure 4.3).

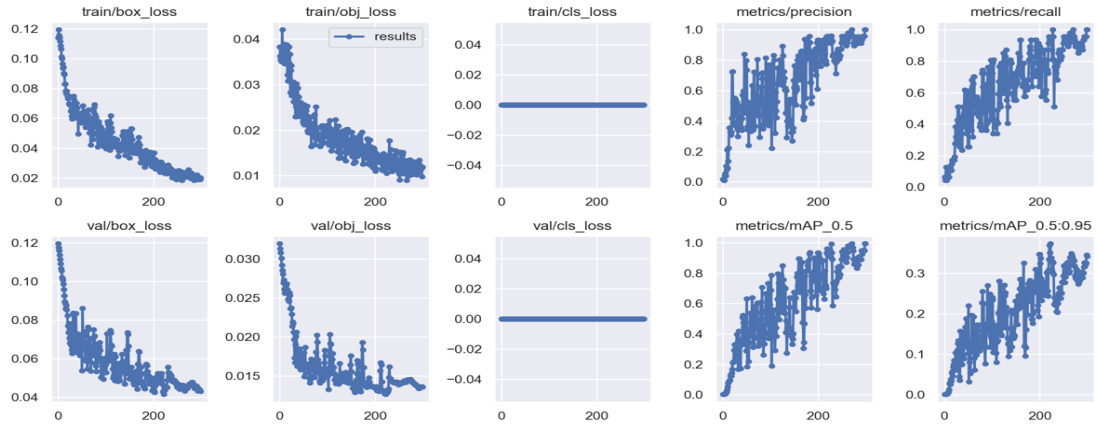

Figure 4.3: Metrics results for 50 images in the training dataset.

The precision, recall, and mAP in this scenario reached a value of 100% in the last epoch, with a 20% increase in the precision, and 50% in the recall, which means that the model can easier predict all the ground truth bounding boxes, and a 40% increase in the $mAP_{0.5}$. In the loss functions, there weren't any noticeable differences, with the box_loss in the training and validation dataset being 0.02 and 0.04 respectively while the training and validation obj_loss were 0.01 and 0.015 respectively. The variation of the values in the precision is smaller and as the model trains for more epochs, it is getting even lower.

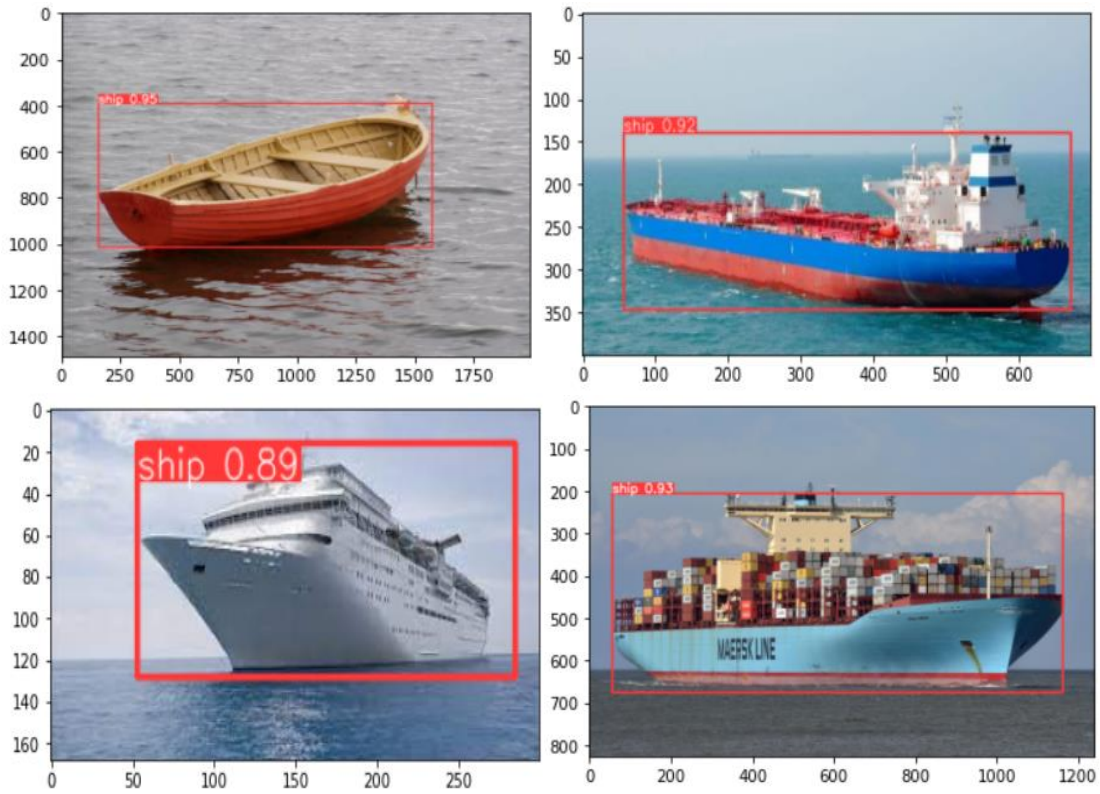

Figure 4.4: Results of the model detection in the test dataset, with 50 images of ships in the training dataset.

The predictions that the model made in the same 4 images are better with an increase of 5-10% in the confidence score on the upper images, reaching 95% for the upper left and 92% for the upper right, and an almost 30% increase in the bottom image in the confidence of the model, with values 89% in the bottom left and 93% in the right (Figure 4.4:). Lastly, the bounding boxes are much better as they contain the objects more accurately. This outcome is logical taking into consideration the increase in the metrics.

4.1.3 100 images in the training dataset

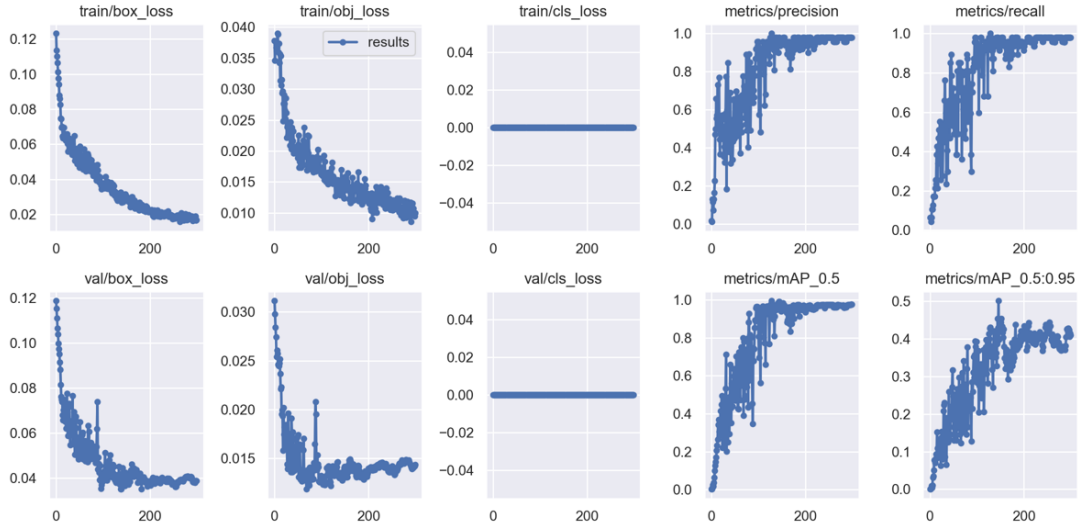

Figure 4.5: Metric results for 100 images in the training dataset.

In the last case, with 100 images of ships in the training dataset, the results are quite interesting, as the model not only reaches 100% for the precision, recall, and mAP but also does that in the first 200 epochs approximately, which is significantly faster than the previous scenarios that were discussed. In the loss function, there isn't any notable difference with all the losses keeping approximately the same values as in the last case (Figure 4.5).

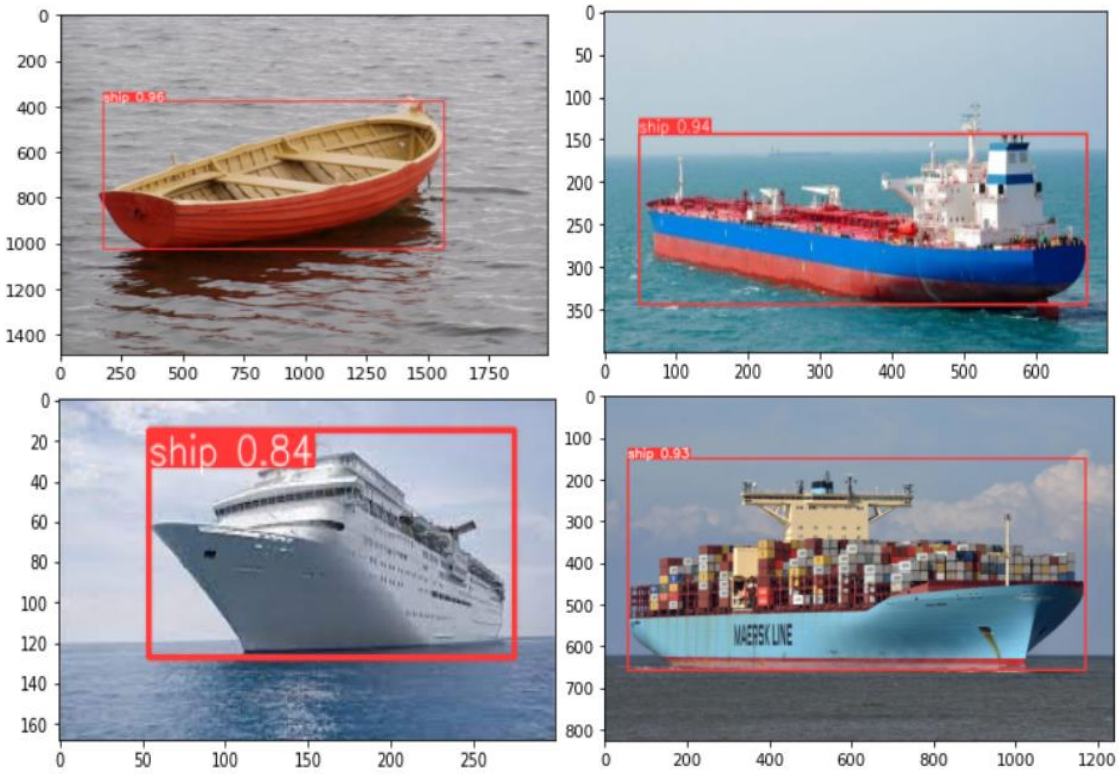

Figure 4.6: Results of the model detection in the test dataset, with 100 images of ships in the training dataset.

In this scenario, the top right image was predicted with 96% confidence, the upper right with 94% confidence, the bottom left with 84% confidence, and the bottom right with 93% confidence. The general increase is only 2% which isn't high, but the bounding boxes, in this case, are even more precise containing exactly the shape of the ship meaning that the model is already at a good stage for predicting images of ships, that are close to the ones that it was trained on (Figure 4.6).

One interesting result also is how the model responds when there is a higher number of ships in the image. The predictions of the model for such an image for the different training datasets are illustrated in the Figure 4.7 In this figure the first prediction is from the model trained with 20 images, the second prediction is from the model trained with 50 images, and the third one is from the model trained with 100 images. The difference between the predictions of the model is significant. The model in the first image didn't manage to detect any of the ships, which can be explained by the fact that the number of such images in the training dataset was too few. In the second prediction, where the model had more images in the training dataset that contained more than one object of the same class in them, predicted the majority of the objects, although with not that precise bounding boxes. In the last case, with a training dataset of 100, the model not only predicted all the ships in the image with a 10% increase in confidence compared to the previous case but also managed to output more accurate bounding boxes that contain only the ships with less background.

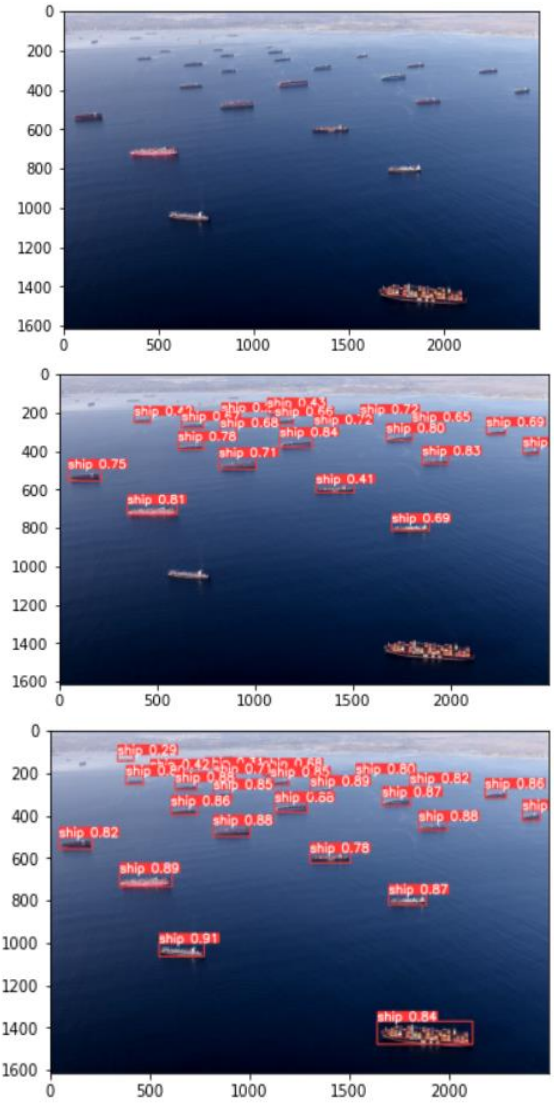

Figure 4.7: Comparison of the predictions of all the models in an image that contains more objects of the class ships.

4.2 Rocks and Floating Objects images

For these 2 cases, the results will be cited in the Appendix B: Results of different cases as they are almost the same as those of the ship so they won't be extensively commented on here. Rocks are considered some tall stones that stick out of the sea and also some images of land. As floating objects are considered buoys, humans at sea and some containers that might have dropped from ships. The images that the network was trained on were 20, 50, and 100 for 300 epochs and a batch size of 8.

4.3 50 photos of limited visibility

One more case that was considered in this thesis was the case of detecting ships with limited visibility. The training dataset included 50 images of ships taken at night or with fog and again the model run for 300 epochs with batch size 8 and considered only ships as the class. The results for the metrics (Figure 4.8) and some predictions of the model in the test images (Figure 4.9) are illustrated below.

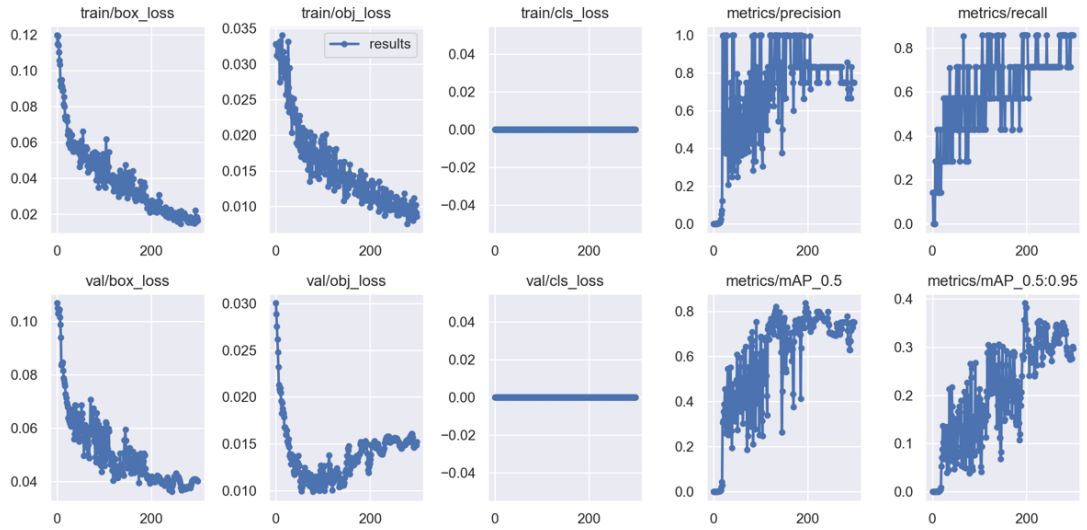

Figure 4.8: Metric results for 50 images of ships with limited visibility in the training dataset.

The precision and mAP reached a value of 80% on average while the recall had an average of 75%. The box_loss for the training and validation had values of 0.04 and 0.02 respectively, while the training and validation obj_loss reaches 0.01 and 0.015 respectively. What is interesting is that the validation loss starts to rise, which means that the model started to overfit on the training dataset. The results for the metrics have high variance in comparison with the dataset that contained 50 ships in daylight, while the values of the metrics differ greatly from the other case with 50 images in the training dataset. This is expected when the model was run with images other than the ones in daylight as image detection is so dependent on the images themselves that every change in the training dataset returns other results.

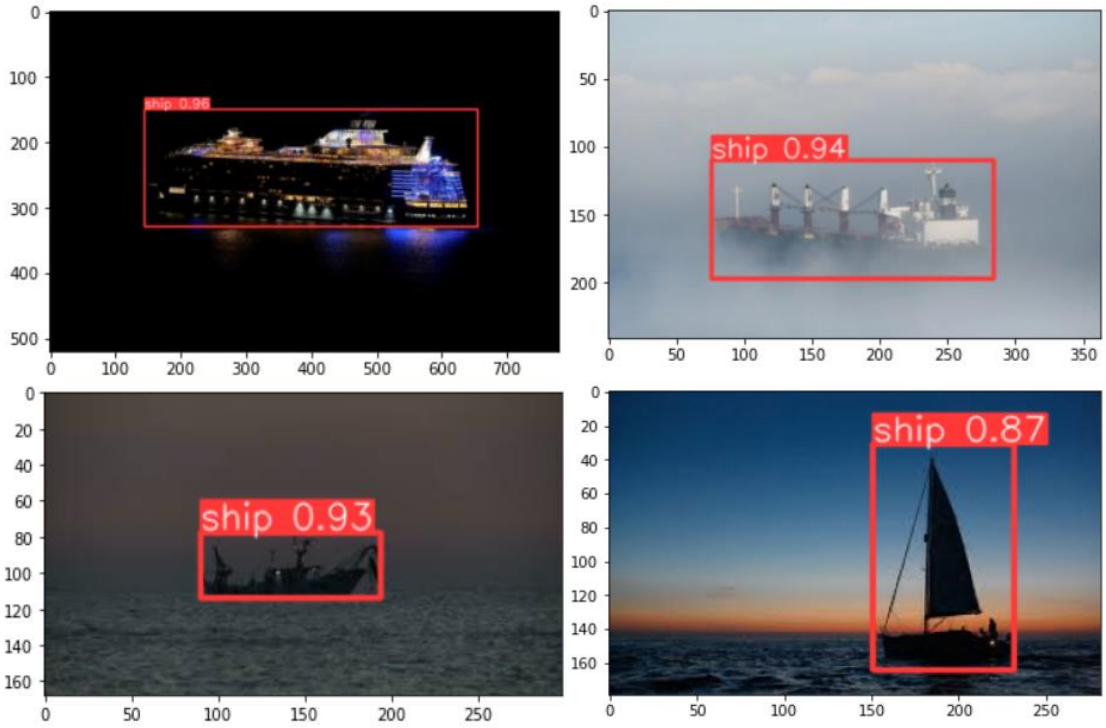

Figure 4.9: Results of the model detection in the test dataset, with 50 images of ships taken in limited visibility in the training dataset.

The model in this scenario predicted the images of the test dataset with good confidence, namely 96% for the upper left, 94% for the upper right, 93%, and 87% for the bottom left and right respectively even though the number of images for this difficult case was pretty limited. This case was added to the total scenarios that were analyzed as the ship must be able to find all the objects in its path regardless of the visibility conditions. These images were also included in the bigger dataset, of 400 images, that will follow.

4.4 300 images of all the classes

Next up the model was tested with 300 images in the training dataset. These images are the ones that the model was trained on in the previous cases individually, namely the 100 ships, the 100 rocks, and the 100 floating objects. In this case, the classes considered are 3, ship, rock, and floating object. The model was trained again for 300 epochs with 8 batch size. The results can be seen in the Figure 4.10 below.

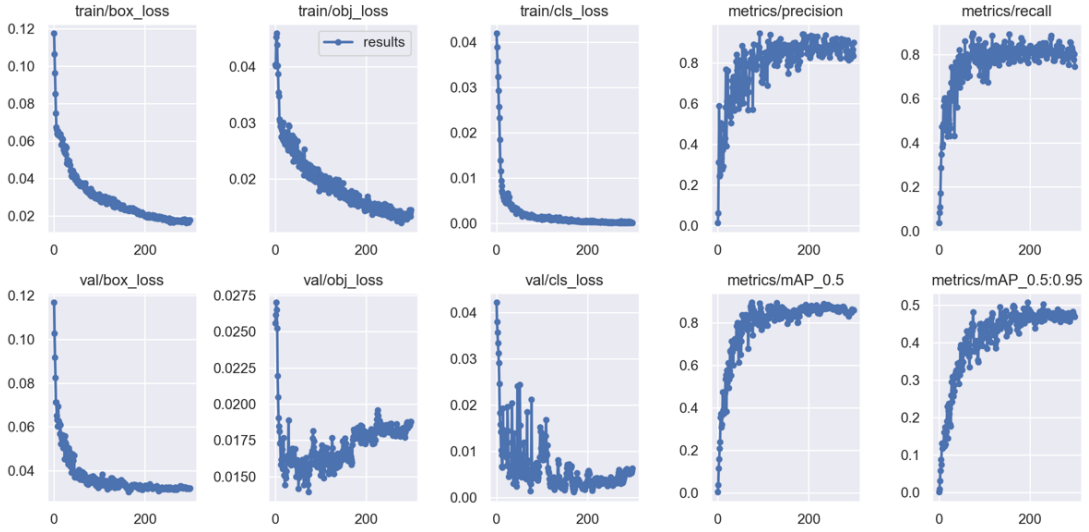

Figure 4.10: Metrics results for a 300 image training, with images of the ship, rocks, and floating objects.

The training seems to have good results in the metrics with the precision, recall, and mAP reaching 0.8, and the loss being very low both in the training and in the validation. What is interesting to discuss is that the model doesn't reach 100% for the values of the precision, recall, and mAP. This happens as the model was trained considering the presence of all the classes that were considered in the problem so it is more difficult to reach 100% precision. The network seemed to have a noticeable variance in the loss of the classes in the beginning, although in the end, it minimized. The model also seems to have started to overfit in the data as the validation obj_loss start to rise as the one in the training dataset lowers. This means that the number of epochs is sufficient as the object loss will have a serious effect on the performance of the model if the training continues for more epochs.

The predictions that the model made for images in the test dataset (Figure 4.11) were generally good with the confidence being above 80% in all images. Although in some cases where the objects were too small or there all the classes coexist in the image the model was making mistakes in its detections regarding the class that the objects refer to as in the left image it considered some floating objects as ships, and didn't recognize the land in the background, while in the left it didn't manage to recognize any of the small objects, as humans and ships(Figure 4.12).

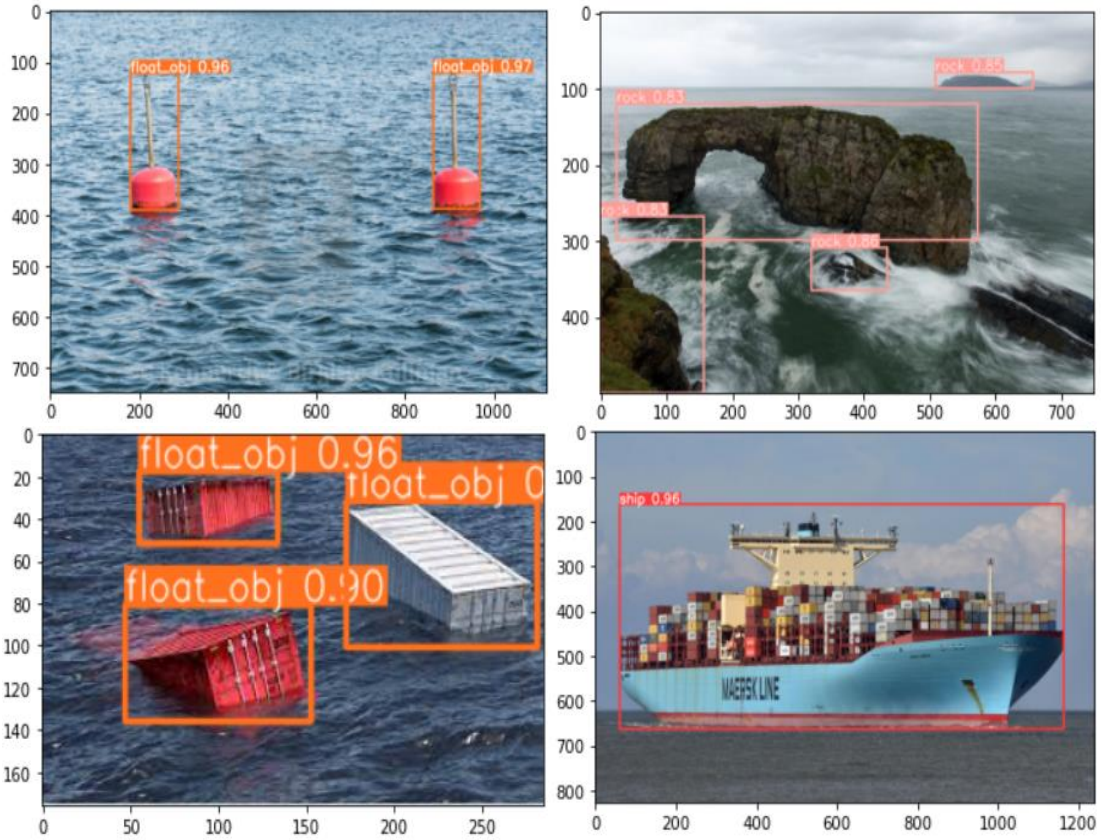

Figure 4.11: Results of the model detection in the test dataset, with 300 images of ships, floating objects and rocks in the training dataset.

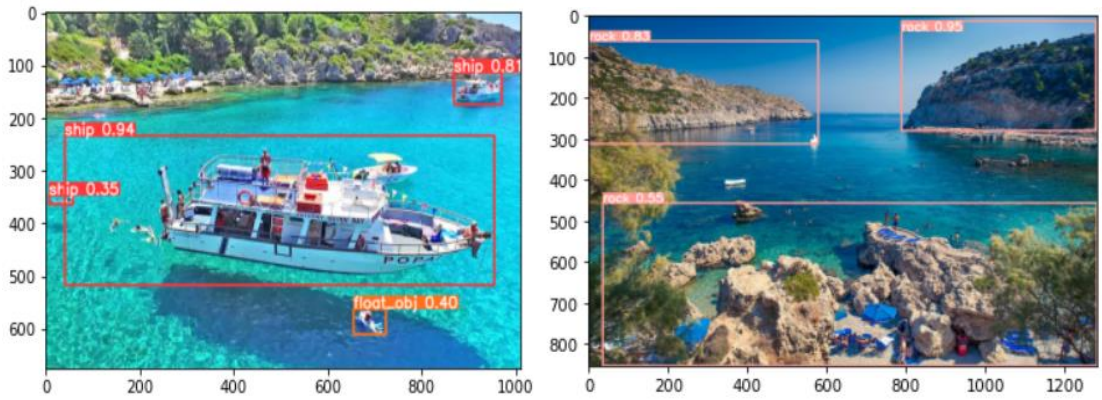

Figure 4.12: Examples of inaccurate predictions of the model.

4.5 400 images of all the classes

In order to overcome the previous case problems, 50 extra images were added to the dataset, which contained objects of all the classes, or at least 2 of them, in each one, and also the ones of ships taken in limited visibility. The model was trained for 300 epochs with a batch size of 8 and a total number of 400 training images.

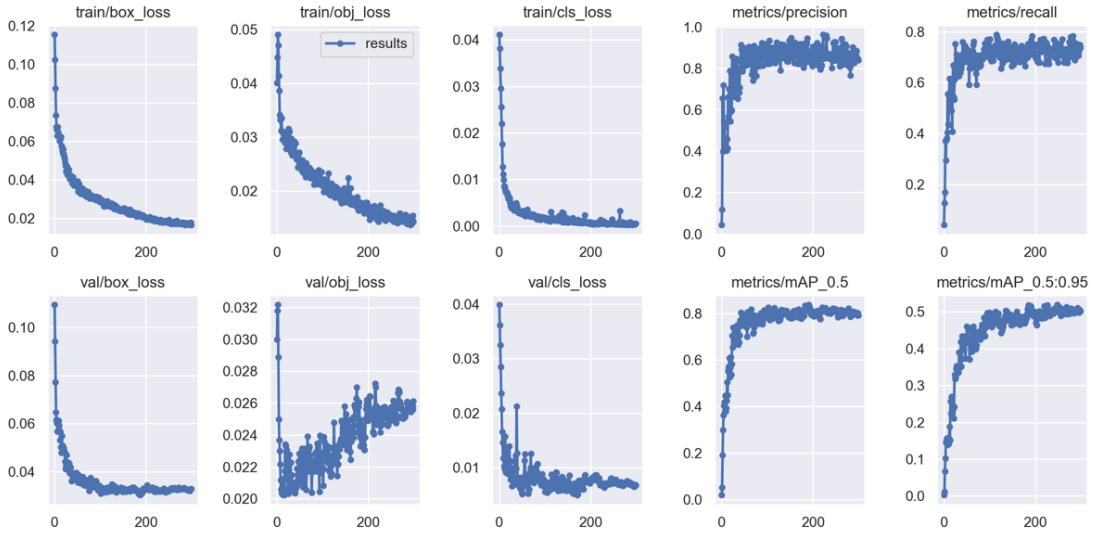

Figure 4.13: Metrics results for 400 images in the training dataset, containing images of all the previous cases.

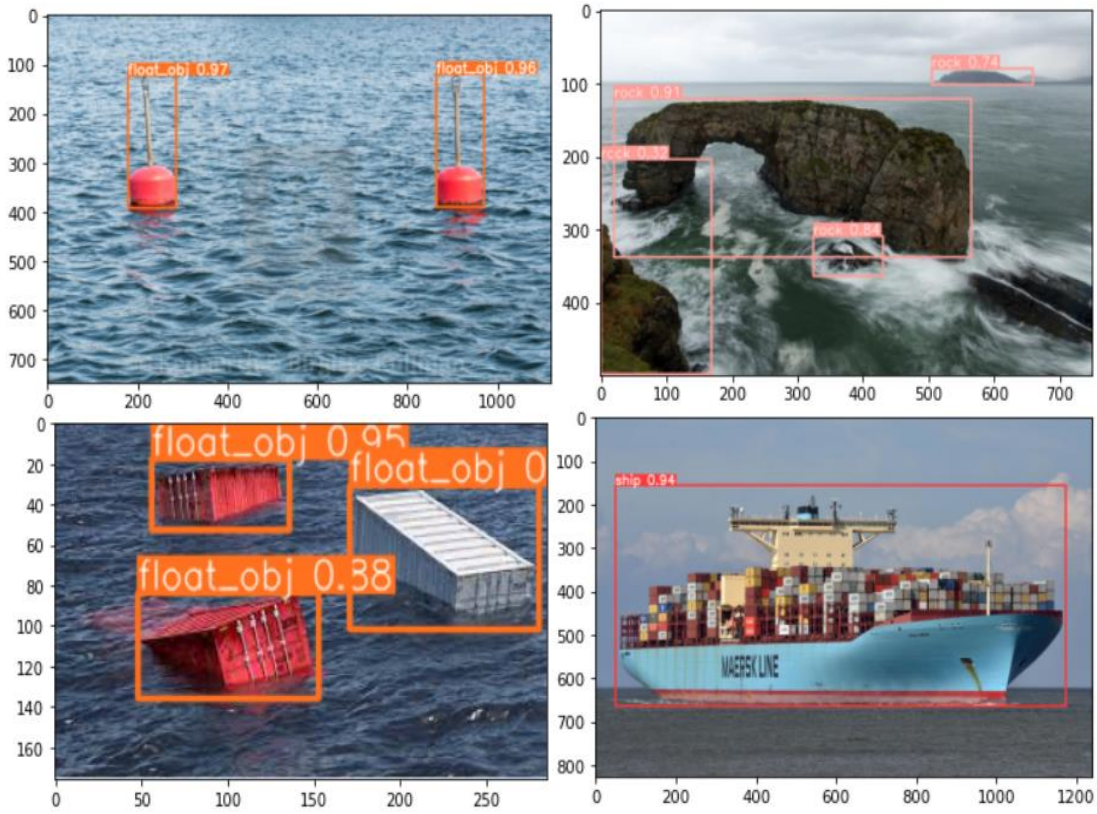

Figure 4.14: Results of the model detection in the test dataset, with 400 images of combination of the previous scenarios in the training dataset.

The model is very fast in its response reaching almost 1 in precision, 0.8 in recall, and mAP in the first 50 epochs and from then it stays the same (Figure 4.13). This is why the validation loss starts rising from then on as the model reached the max values for these metrics and overfits. The other losses don't have some interest as they keep as usual low values, 0.02 and 0.04 in box_loss for training and validation respectively, 0.015 for the training obj_loss, 0 and 0.01 in for the cls_loss.

There is not so much of a difference to be observed between this case and the last one for the confidence of the model in the predictions (Figure 4.14). Although there is a massive difference in the way the model distinguishes between the classes that the object belongs (Figure 4.15). The model now not only predicts the correct class but also can predict more objects than before and with higher confidence. That shows again how important are the images for the training process in object detection.

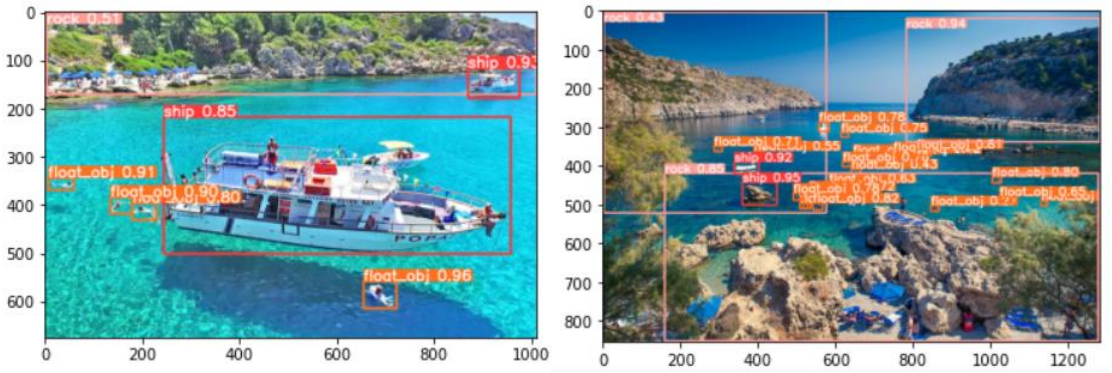

Figure 4.15: Examples of images from the test dataset that the model predicted better than before.

What was also done in this case was to compare the results of YOLOv5s with those of the YOLOv5m. The difference in the architecture is that the YOLOv5m has more layers, especially more BottleneckCSP blocks, and generally the kernels in each of the convolutional layers were bigger than the ones in the YOLOv5s making the detection more precise and better. The YOLOv5m model was trained for the same amount of images for the same epochs and the same batch size as the YOLOv5s, namely 400 images, 300 epochs, and batch size 8. The metrics of the model didn't have much of a difference, from the ones in the YOLOv5s case and are cited in the Appendix B: Results of different cases. Although the predictions of the model are better in most of the cases.

In the following figure, Figure 4.16, the results of some predictions of the model are presented with the YOLOv5s predictions being on the right and the other on the left. In the first pair of images, the YOLOv5m's confidence score is higher than the smaller model by 5% and in some cases 40%. In the second pair, it is observed that the right image scores are the same as the left ones in the ship classes, but the medium size model predicted the correct floating object. In the third pair, the medium model seems to have predicted all the objects in the image with an increase of 10% in the confidence score, in comparison to the small model that didn't find some of them. In the last pair of images, it is obvious that the medium has done a better prediction all in all as it predicted all the ships in the image and with an increase in confidence score of 2% and in some cases 10% compared to the small, which predicted a floating object that didn't exist.

Generally, the predictions done with the YOLOv5m model were better than the ones done with the YOLOv5s as more layers help in analyzing the image better and finding all the important features. Although the increase in size, with YOLOv5s being 14MB and YOLOv5m

being 41MB, the increase in the time spent with the same setup in the computer, with the YOLOv5s taking 4 hours to run 300 epochs with 400 images and batch size 8 and the YOLOv5m taking 11, with the same parameters, made the model not so efficient for the current project.

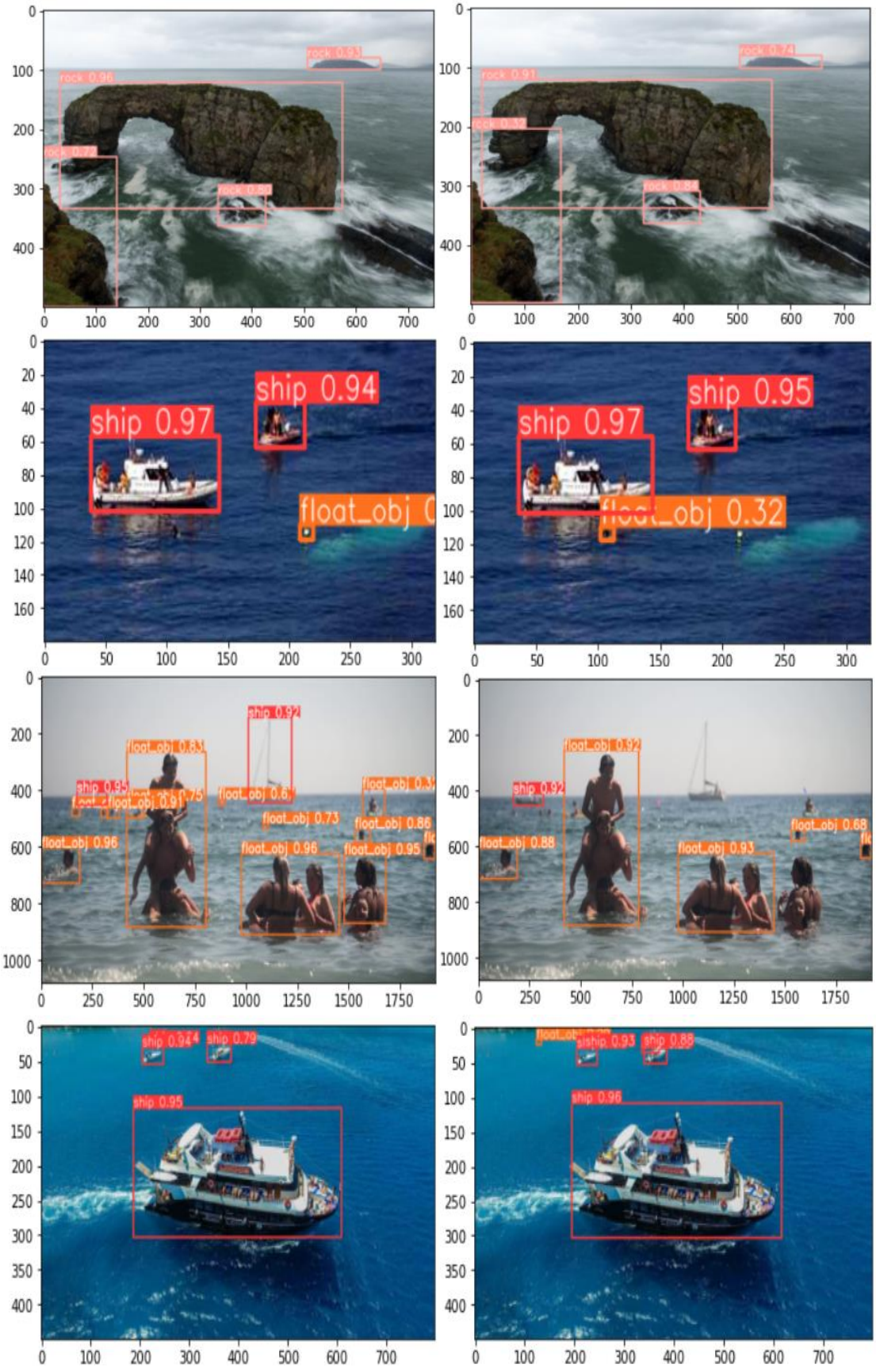

Figure 4.16: Comparison between YOLOv5s and YOLOv5m trained on the same 400 images for 300 epochs and batch size of 8. The right images are the results from the YOLOv5s model and the left are from YOLOv5m

4.6 Video

Except for images, the model was tested on a video in order to see its performance. What was noticed is that even with only 400 images on the training dataset the model's detections were very good not only in the accuracy of the bounding boxes, which contained the ships in most of the cases precisely but also in the confidence of the detections, which in most cases had values of 80% and higher. That combined with a zero fps reduction caused by the model, as the frames per second remain 25, the same as the original video, can prove that YOLOv5s can easily be used in real-time applications and return promising values. Some images from the video with the detections can be seen in the Figure 4.17: Images from the detections of the model in the video application below.

Figure 4.17: Images from the detections of the model in the video application

This video had some snapshots of the sea life and showed images from the bridge of the ship with other ships. These contained a ship being alone in the sea or a couple of ships and also ships at night. In the top left image, the model predicted the ship with an 85% confidence score, while in the top right the confidence is above 70% in most cases. In the bottom left the model output confidence of 69% and 56% for the right-most and the left-most ship and 84% and 94% for the other 2. In the last image, the model predicted both of the ships with confidence of 85% and 90%, even though it was an image taken in low visibility.

4.7 Comparison with other studies

Lastly, to have a way to compare the results of this model for the metrics, some other projects that also used YOLOv5s were considered. Although all of the projects were trained on a lot more images than the one in this thesis a comparison of the metrics can be done. In the project of Bin Yan (Yan et al., 2021) the precision of the model is 83%, the recall 91%, and the mAP 87%, while in the study of Emmanuel Vasilopoulos (Vasilopoulos et al., 2022) the metrics were 86%, 62%, and 66% respectively. In some other studies mentioned in the Literature Review at the beginning of this thesis, one model achieved an mAP value of 71.6% (Zhang et al., 2022), another project achieved mAP 86.5% (Zheng et al., 2022), while the last one discussed resulted in mAP 89.8% (Kim et al., 2022). The results in this thesis on average were, 83% for precision, 78% for recall, and 83% for the mAP. Considering that these results were close enough and even in some cases higher than the other studies and also that this work was done on a very small dataset, the metrics of this thesis can be considered sufficient for the number of images. With more training, the model can be more accurate and also return better results.

5 Conclusion and Future Work

5.1 Conclusion

In this thesis, the use of an object detection algorithm, especially the method YOLO, in detecting objects, mainly ships, rocks, and other floating objects, was researched. The model used for that reason was the newest version of YOLO, YOLOv5, developed by Ultralytics. The model was trained for different scenarios and the results for the metrics and the model's predictions were analyzed and compared. Some interesting results from this work were extracted and are presented here collectively.

The number of images plays a significant role for the model. As the training dataset grows the model's metrics, precision, recall, and mAP, reach higher values for the same epochs and the variance of the values with each epoch is lesser. For example, 20 images of ships reached a precision of 70% in comparison to the 100% in the 100 images. Furthermore, the highest values for the metrics are reached sooner as the training set expands. In the case of 50 images, the higher values for the metrics weren't reached with 300 epochs while in the scenario of 100 and 300 images, they were reached in less than 200 epochs and 50 epochs respectively. It is also observed that when the model is trained with different images, but the same in number, then the results for the metrics are different in some cases. This means that the model is very dependent on the quality, the background of the images, and the conditions under which the images were taken, namely if the vision is limited.

The time spent for the training to be done also increases with the number of images in the training dataset by a significant amount. A model trained for 300 epochs with a dataset of 100 images takes 1 hour for its training while for the same amount of epochs 400 images take 4 hours, making it 1 hour extra for every 100 images added to the dataset. Also, the amount of time spent for the training of the model increases exponentially as the model gets larger and deeper, with 400 images taking 4 hours in the small network and 11 hours in the medium network trained for the same 300 epochs.

The confidence of the network in detecting objects in images never seen by it is enhanced as the training set is provided with a more wide variety of images that are relevant to the ones that the model is supposed to detect. It was observed that the model was making mistakes in its predictions in images where all the classes coexisted when trained with 300 images of ship, rocks and floating objects, but with no image having the objects coexist. When such images with all of the classes were included, like in the scenario of 400 images in the training dataset, the model managed to have a 2% increase in its confidence scores and 10% in some cases, while also predicting much more of the smaller objects with high confidence and accurate bounding boxes.

Lastly, the size of the network is also very crucial for the results, as the more the network grows the easier it is to find the correct objects and have higher accuracy. When the same dataset was trained in the small and the medium model, improvements were observed not only in the confidence of the model, which reached a 20% increase in some cases but also in the number of objects the model could predict.

5.2 Future work

In this thesis, it became clear that the YOLOv5 algorithm can be used for object detection in ship navigation not only in detecting images of ships and other objects but also in real-time applications, such as in autonomous ships. This thesis can only be the spark of new works regarding the field of object detection application in ships, so some ideas for future works regarding this topic are presented here.

The number of images that the model was trained on was very insufficient for the model to be able to work properly as it requires 1500 images per class minimum. In future studies, the model should be enhanced with more images for each class to acquire better results.

Moreover, the images used should be more representative of the case that is considered, mainly focusing on trying to find images that have the same angle and height of a camera placed on a ship and also be of the same analysis as the model requires. The training dataset should also be tested in the larger versions of YOLOv5 to have better results. The parameters of the model should be changed too and see where the best results are acquired.

After the model was trained and better results were obtained, the next step should be to combine the results of the model with the EGDIS system that can find other obstacles' courses in the vicinity but can't recognize what those objects are. This way a full picture of the obstacles in the way of the ship can be obtained. An AI system can acquire these results and propose a course for the ship in order to avoid collision with these obstacles. This AI system then should be able to could change the speed of the ship and steer it, to avoid collision according to the results course it proposed. This whole system afterward should be tested on a model in a tank.

References

Albiol, A., Torres, L., & Delp, E. J. (2001). An unsupervised color image segmentation algorithm for face detection applications. https://doi.org/10.1109/ICIP.2001.958585

Bhagya, C., & Prof. Shyna, A. (2019). An Overview of Deep Learning Based Object Deteection Techniques.

Bishop, C. M. (1994). Neural Networks and their applications. *Rev. Sci. Instrum*, *65*, 1803-1830.

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*.

Bochkovskiy, A., Wang, C.-Y., & Liao, H.-Y. M. (2020). YOLOv4: Optimal Speed and Accuracy of Object Detection.

Chang, L., Chen, Y.-T., Wang, J.-H., & Chang, Y.-L. (2022). Modified Yolov3 for Ship Detection with Visible and Infrared Images. *Electronics*, *11*(5). https://doi.org/10.3390/electronics11050739

Chang, Y.-L., Anagaw, A., Chang, L., Wang, Y., Hsiao, C.-Y., & Lee, W.-H. (2019). Ship Detection Based on YOLOv2 for SAR Imagery. *Remote Sensing*, *11*(7). https://doi.org/10.3390/rs11070786

Chen, D., Sun, S., Lei, Z., Shao, H., Wang, Y., & Chen, X. (2021). Ship Target Detection Algorithm Based on Improved YOLOv3 for Maritime Image. *Journal of Advanced Transportation*, *2021*, 1-11. https://doi.org/10.1155/2021/9440212

Chen, X., Qi, L., Yang, Y., Postolache, O., Yu, Z., & Xu, X. (2019). *Port ship detection in complex environments* 2019 International Conference on Sensing and Instrumentation in IoT Era (ISSI),

Cui, H., Yang, Y., Liu, M., Shi, T., & Qi, Q. (2019). Ship Detection: An Improved YOLOv3 Method.

Dai, J., Li, Y., He, K., & Sun, J. (2016). R-FCN: Object Detection via Region based Fully Convolutional Networks.

Dalal, N., & Triggs, B. (2005). Histograms of Oriented Gradients for Human Detection.

Farley, B. G., & Clark, W. A. (1954). Simulation of Self-Organizing systems by digital computer.

Felzenszwalb, P. F., Girshick, R. B., McAllester, D., & Ramanan, D. (2010). Object detection with discriminatively trained part-based models. *IEEE Trans Pattern Anal Mach Intell*, *32*(9), 1627-1645. https://doi.org/10.1109/TPAMI.2009.167

Girshick, R. (2015). *Fast R-CNN* 2015 IEEE International Conference on Computer Vision (ICCV),

Girshick, R., Donahue, J., Darrell, T., & Malik, J. (2014). *Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation* 2014 IEEE Conference on Computer Vision and Pattern Recognition,

Han, X., Zhao, L., Ning, Y., Hu, J., & Chen, C.-H. (2021). ShipYOLO: An Enhanced Model for Ship Detection. *Journal of Advanced Transportation*, *2021*, 1-11. https://doi.org/10.1155/2021/1060182

Hass, F. S., & Jokar Arsanjani, J. (2020). Deep Learning for Detecting and Classifying Ocean Objects: Application of YoloV3 for Iceberg–Ship Discrimination. *ISPRS International Journal of Geo-Information*, *9*(12). https://doi.org/10.3390/ijgi9120758

He, G., Wang, W., Shi, B., Liu, S., Xiang, H., & Wang, X. (2022). An Improved YOLO v4 Algorithm-based Object Detection Method for Maritime Vessels. *International Journal of Science and Engineering Applications*, *11*(04), 50-55. https://doi.org/10.7753/ijsea1104.1001

He, K., Zhang, X., Ren, S., & Sun, J. (2015). Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition. *IEEE Trans Pattern Anal Mach Intell*, *37*(9), 1904-1916. https://doi.org/10.1109/TPAMI.2015.2389824

Hebb, D. O. (1949). *The organization of Behaviour*.

Hopfield, J. J. (1982). Neural networks and physical systems with emergent collective computational abilities. *Biophysics*, *79*, 2554-2558.

Hopfield, J. J. (1984). Neurons with grabed response have collective computational properties like those of two-state neurons. *81*, 3088-3092.
Jie, Y., Leonidas, L., Mumtaz, F., & Ali, M. (2021). Ship Detection and Tracking in Inland Waterways Using Improved YOLOv3 and Deep SORT. *Symmetry*, *13*(2). https://doi.org/10.3390/sym13020308
Kamate, S., & Yilmazer, N. (2015). Application of Object Detection and Tracking Techniques for Unmanned Aerial Vehicles. *Procedia Computer Science*, *61*, 436-441. https://doi.org/10.1016/j.procs.2015.09.183
Kim, J.-H., Kim, N., Park, Y. W., & Won, C. S. (2022). Object Detection and Classification Based on YOLO-V5 with Improved Maritime Dataset. *Journal of Marine Science and Engineering*, *10*(3). https://doi.org/10.3390/jmse10030377
Kim, P. (2017). Convolutional Neural Network. In *MATLAB Deep Learning* (pp. 121-147). https://doi.org/10.1007/978-1-4842-2845-6_6
Kmieć, M., & Glowacz, A. (2015). Object detection in security applications using dominant edge directions. *Pattern Recognition Letters*, *52*, 72-79. https://doi.org/10.1016/j.patrec.2014.09.018
LeCunn, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-Based Learning Applied to Document Recognition.
Li, L., Jiang, L., Zhang, J., Wang, S., & Chen, F. (2022). A Complete YOLO-Based Ship Detection Method for Thermal Infrared Remote Sensing Images under Complex Backgrounds. *Remote Sensing*, *14*(7). https://doi.org/10.3390/rs14071534
Li, T., Zhang, K., Li, W., & Huang, Q. (2019). *Research on ROI Algorithm of Ship Image Based on Improved YOLO* 2019 International Conference on Artificial Intelligence and Advanced Manufacturing (AIAM),
Li, Z., Zhao, L., Han, X., Pan, M., & Hwang, F.-J. (2020). Lightweight Ship Detection Methods Based on YOLOv3 and DenseNet. *Mathematical Problems in Engineering*, *2020*, 1-10. https://doi.org/10.1155/2020/4813183
Liu, T., Pang, B., Zhang, L., Yang, W., & Sun, X. (2021). Sea Surface Object Detection Algorithm Based on YOLO v4 Fused with Reverse Depthwise Separable Convolution (RDSC) for USV. *Journal of Marine Science and Engineering*, *9*(7). https://doi.org/10.3390/jmse9070753
Liu, W., Anguelov, D., Erhan, D., & Szegedy, C. (2016). *SSD: Single Shot MultiBox Detector*. https://doi.org/10.1007/978-3-319-46448-0
Lu, K., An, X., Li, J., & He, H. (2017). Efficient deep network for vision-based object detection in robotic applications. *Neurocomputing*, *245*, 31-45. https://doi.org/10.1016/j.neucom.2017.03.050
Ma, Z., Zeng, Y., Wu, L., Zhang, L., Li, J., & Li, H. (2021). *Water Surface Targets Recognition and Tracking Based on Improved YOLO and KCF Algorithms* 2021 IEEE International Conference on Mechatronics and Automation (ICMA),
McCulloch, W. S., & Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, *5*.
Minsky, M., & Papert, S. (1969). *Perceptrons, An Introduction to Computational Geometry*.

Naghavi, S. H., Avaznia, C., & Talebi, H. (2017). Integrated Real-Time Object Detection for Self-Driving Vehicles. *10th Iranian Conference on Machine Vision and Image Processing*, 154-158.
Nie, X., Yang, M., & Liu, W. R. (2019). Deep Neural Network-Based Robust Ship Detection Under Different Weather Conditions. *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, 47-52.
O' Shea, K., & Nash, R. (2015). An Introduction to Convolutional Neural Networks.
Padilla, R., Netto, S. L., & da Silva, E. A. B. (2020). A Survey on Performance Metrics for Object-Detection Algorithms. https://doi.org/10.1109/iwssip48289.2020
Padilla, R., Passos, W. L., Dias, T. L. B., Netto, S. L., & da Silva, E. A. B. (2021). A Comparative Analysis of Object Detection Metrics with a Companion Open-Source Toolkit. *Electronics*, *10*(3). https://doi.org/10.3390/electronics10030279

Pytorch. *Pytorch Hardswish activation function*. https://pytorch.org/docs/stable/generated/torch.nn.Hardswish.html

Redmon, J. (2020). *Ultralytics YOLOv5*. https://github.com/ultralytics/yolov5

Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You Only Look Once:

Unified, Real-Time Object Detection. *Computer Vision Foundation*, 779,788.
Redmon, J., & Farhadi, A. (2016). YOLO9000: Better, Faster, Stronger.
Redmon, J., & Farhadi, A. (2018). YOLOv3: An Incremental Improvement.
Ren, S., He, K., Girshick, R., & Sun, J. (2016). Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks.
Rochester, N., Holland, J. H., Haibt, L. H., & Duda, W. L. (1956). Tests on a cell assembly theory of the action of the brain using a large digital computer.
Rosenblatt, F. (1958). The Perceptron: A probabilistic model for information storage and organization in the brain. *Physcological Review*, *65*, Article 6.
Rosenblatt, F. (1961). *Pricniples of neurodynamics, Perceptron and the theory of brain mechanisms*.
Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning represantations by back-propagating errors. *NATURE*, *323*, 533-536.
Schmidhuber, J. (1992). Learning complex, extended sequences using the principle of history compression. *Neural Computation*, *4*, 234-242.
Shao, Z., Wang, L., Wang, Z., Du, W., & Wu, W. (2020). Saliency-Aware Convolution Neural Network for Ship Detection in Surveillance Video. *IEEE Transactions on Circuits and Systems for Video Technology*, *30*(3), 781-794. https://doi.org/10.1109/tcsvt.2019.2897980
Sharma, S., Sharma, S., & Athaiya, A. (2020). Activation functions in neural networks. *International Journal of Engineering Applied Sciences and Technology*, *4*(12), 310-316.
Solawetz, J. (2020, June 29, 2020). YOLOv5 New Version - Improvements And Evaluation. https://blog.roboflow.com/yolov5-improvements-and-evaluation/

Teuwen, J., & Moriakov, N. (2020). Convolutional neural networks. In *Handbook of Medical Image Computing and Computer Assisted Intervention* (pp. 481-501). https://doi.org/10.1016/b978-0-12-816176-0.00025-9
Tzutalin, D. (2015). *labelimg*. https://github.com/tzutalin/labelImg

Uijlings, J. R. R., van de Sande, K. E. A., Gevers, T., & Smeulders, A. W. M. (2013). Selective Search for Object Recognition. *International Journal of Computer Vision*, *104*(2), 154-171. https://doi.org/10.1007/s11263-013-0620-5
Vasilopoulos, E., Vosinakis, G., Krommyda, M., Karagiannidis, L., Ouzounoglou, E., & Amditis, A. (2022). A Comparative Study of Autonomous Object Detection Algorithms in the Maritime Environment Using a UAV Platform. *Computation*, *10*(3). https://doi.org/10.3390/computation10030042
Vedaldi, A., Gulshan, V., Varma, M., & Zisserman, A. (2009). Multiple Kernels for Object Detection.
Viola, P., & Jones, M. (2001). Rapid Object Detection using a Boosted Cascade of Simple Features.

Wang, Y., Ning, X., Leng, B., & Fu, H. (2019). *Ship Detection Based on Deep Learning* International Conference on Mechatronics and Automation,
Weng, J., Ahyja, N., & Huang, T. S. (1992). Cresceptron: A Self-Organizing Neural Network Which Grows Adaptively. 576-581.
Weng, J. J., Ahyja, N., & Huang, T. S. (1993). Learning Recognition and Segmentation of 3-D Objects from 2-D Images. 121-128.

Yamashita, R., Nishio, M., Do, R. K. G., & Togashi, K. (2018). Convolutional neural networks: an overview and application in radiology. *Insights Imaging*, *9*(4), 611-629. https://doi.org/10.1007/s13244-018-0639-9

Yan, B., Fan, P., Lei, X., Liu, Z., & Yang, F. (2021). A Real-Time Apple Targets Detection Method for Picking Robot Based on Improved YOLOv5. *Remote Sensing*, *13*(9). https://doi.org/10.3390/rs13091619

Yang, J., Li, Y., Zhang, Q., & Ren, Y. (2019). *Surface Vehicle Detection and Tracking with Deep Learning and Appearance Feature* 2019 5th International Conference on Control, Automation and Robotics,

Yegnanarayana, B. (1994). Artificial neural networks for pattern recognition. *Sadhana*, *19*, 189-238.

YuanQiang, C., Du, D., Zhang, L., Wen, L., Wang, W., Wu, Y., & Lyu, S. (2020). *Guided Attention Network for Object Detection and Counting on Drones* Proceedings of the 28th ACM International Conference on Multimedia,

Zakria, Z., Deng, J., Kumar, R., Khokhar, M. S., Cai, J., & Kumar, J. (2022). Multiscale and Direction Target Detecting in Remote Sensing Images via Modified YOLO-v4. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, *15*, 1039-1048. https://doi.org/10.1109/jstars.2022.3140776

Zhang, T., Zhang, X., Shi, J., & Wei, S. (2019). HIGH-SPEED SHIP DETECTION IN SAR IMAGES BY IMPROVED YOLOV3. 149-152.

Zhang, X., Yan, M., Zhu, D., & Guan, Y. (2022). Marine ship detection and classification based on YOLOv5 model. *Journal of Physics: Conference Series*, *2181*. https://doi.org/10.1088/1742-6596/2181/1/012025

Zheng, J. C., Sun, S. D., & Zhao, S. J. (2022). Fast ship detection based on lightweight YOLOv5 network. *IET Image Processing*, *16*(6), 1585-1593. https://doi.org/10.1049/ipr2.12432

Zheng, R., Zhou, Q., & Wang, C. (2019). *Inland River Ship Auxiliary Collision Avoidance System* 2019 18th International Symposium on Distributed Computing and Applications for Business Engineering and Science (DCABES),

Zhou, W., & Lu, L. (2022). An Efficient Ship Detection and Classification Algorithm based on YOLOv4. *International Core Journal of Engineering*, *8*(4), 163-173. https://doi.org/10.6919/ICJE.202204_8(4).0023

Zhu, H., Wei, H., Baoqing, L., Yuan, X., & Kehtarnavaz, N. (2020). A review of Video Object Detection: Datasets, Metrics and Methods. *applied sciences*. https://doi.org/10.3390/app10217834

Appendix

Appendix A: Hardware and Software Specifics

Hardware Specifics	
CPU	Intel(R) Core(TM) i5-6300HQ
CPU Clockspeed	2.3 GHz
CPU threads	4 cores (4 threads)
GPU	Nvidia(R) GeForce(R) GTX 960M
GPU Clockspeed	2.6 Ghz
VRAM	4GB GDDR5
Storage	450 GB SSD
Software Specifics	
OS	Windows 10 Pro (x64)
Python version	3.9.7
Cuda version	release 10.2 version 10.2.89
Pytorch version	1.10.2
Jupyter notebook version	6.4.6

Appendix B: Results of different cases

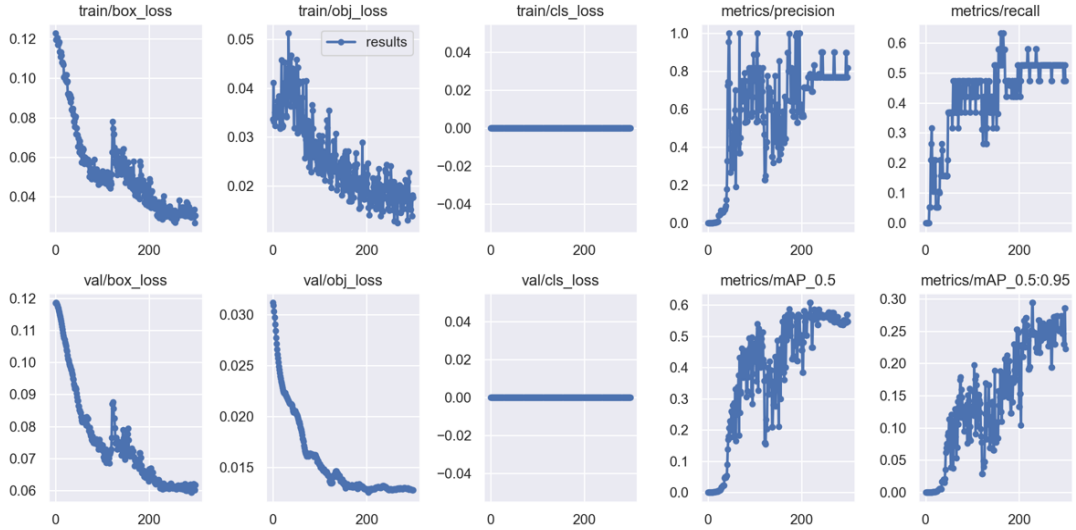

Figure 0.2: Metric results for a training dataset with 20 images of floating objects.

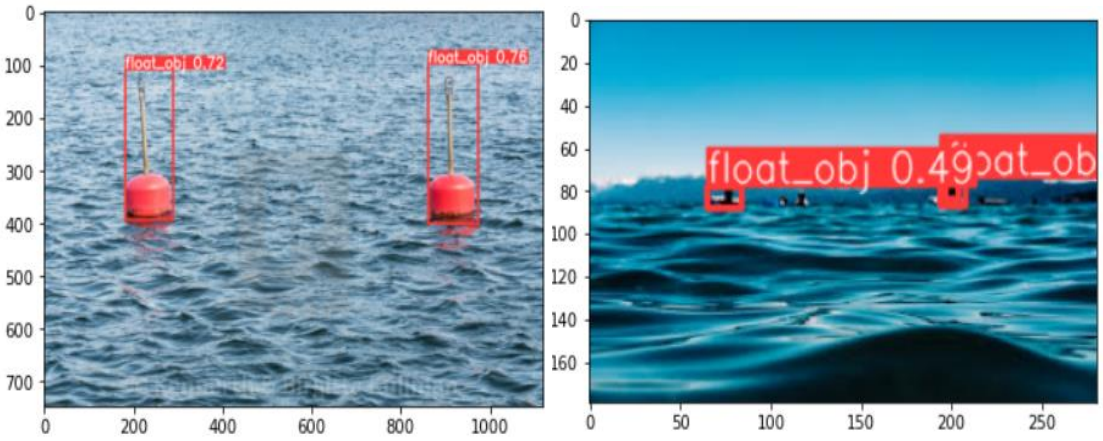

Figure 0.1: Results of the model detections, with the 20 images of floating objects in the training dataset.

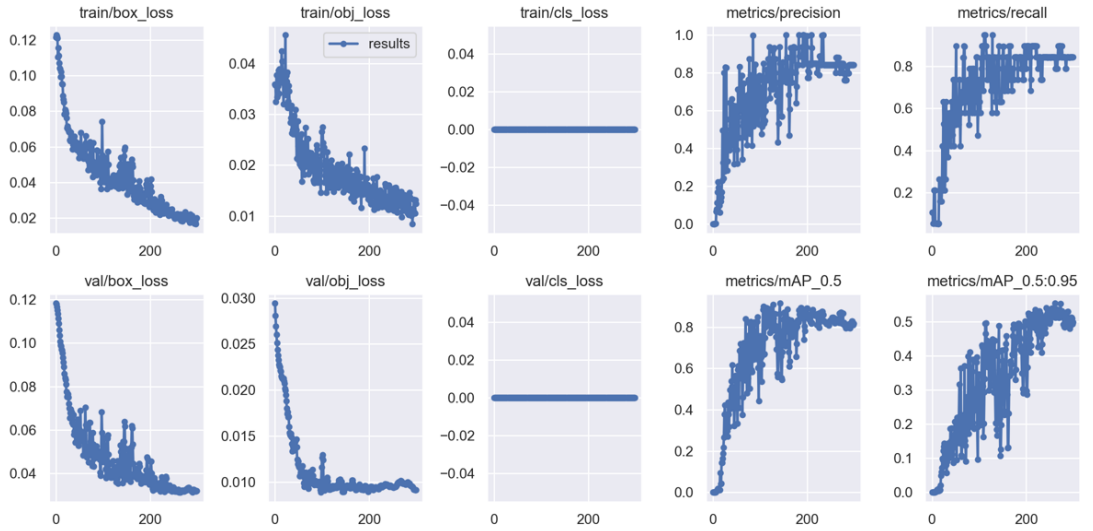

Figure 0.3: Metric results for a training dataset with 50 images of floating objects.

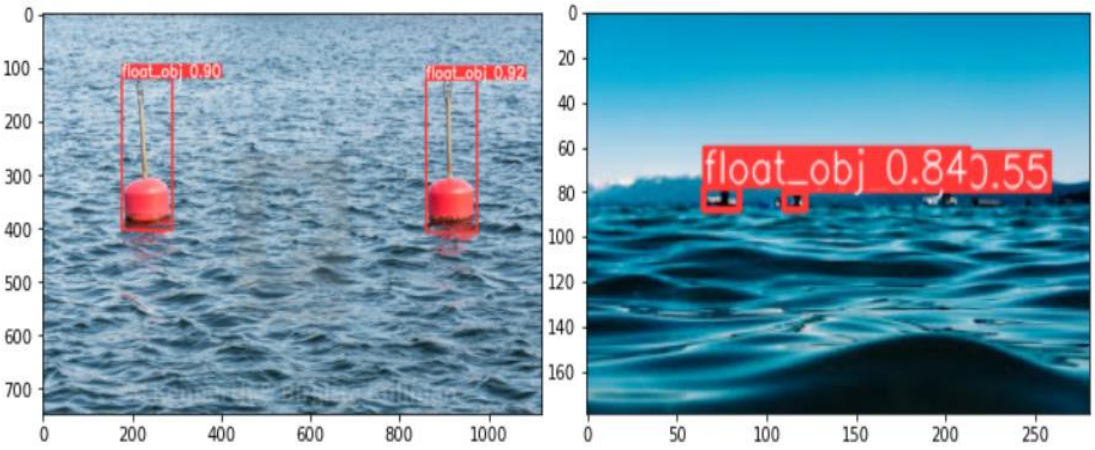

Figure 0.4: Results of the model detections, with the 50 images of floating objects in the training dataset.

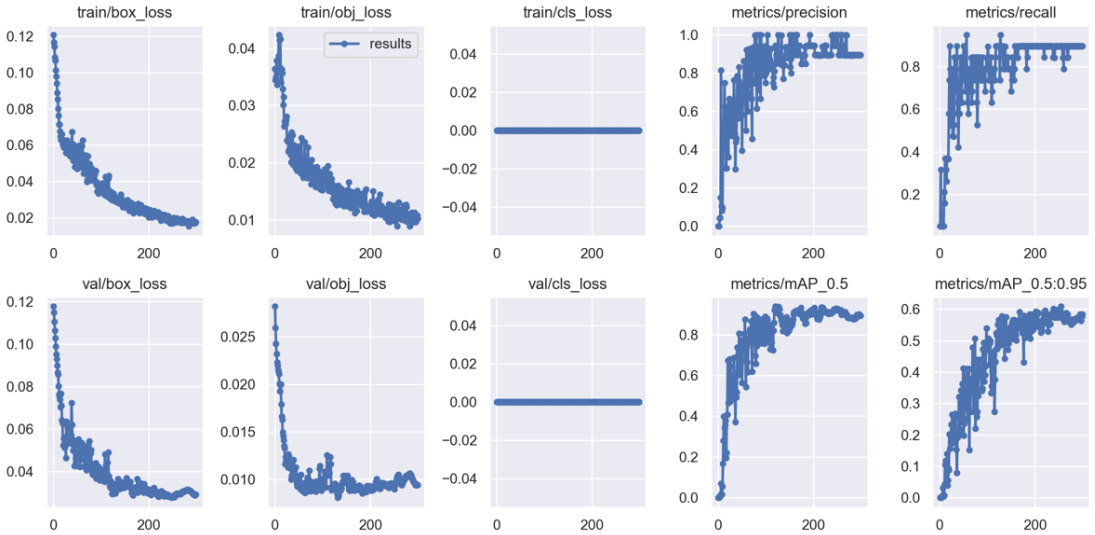

Figure 0.5: Metric results for a training dataset with 100 images of floating objects in the training dataset.

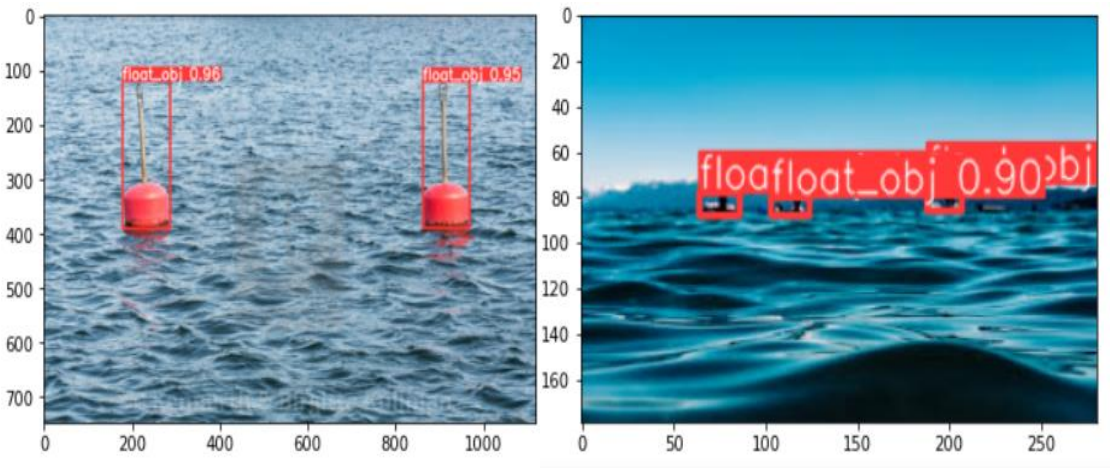

Figure 0.6: Results of the model detections, with the 100 images of floating objects.

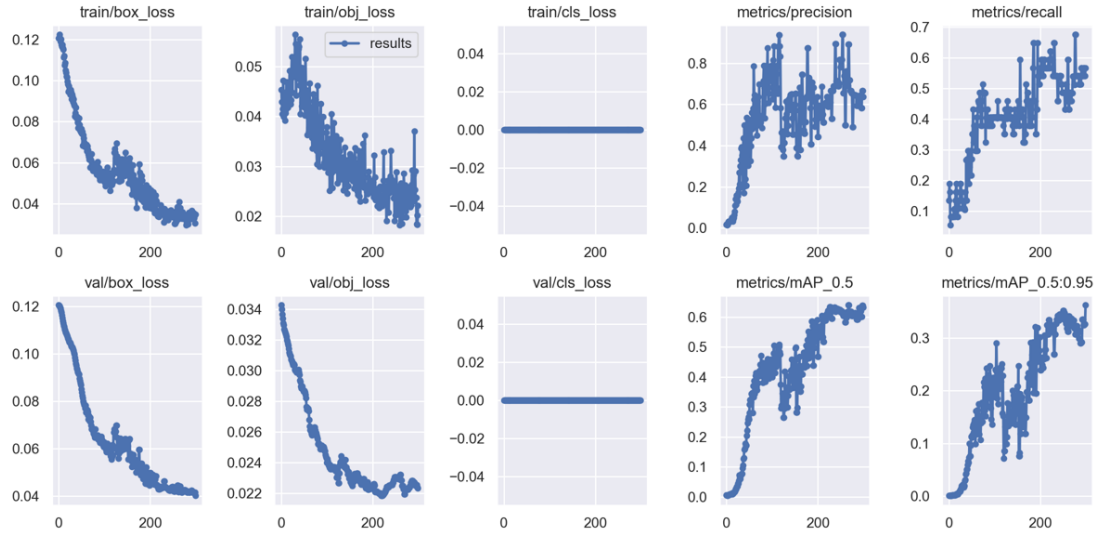

Figure 0.7: Metric results for a training dataset with 20 images of rocks.

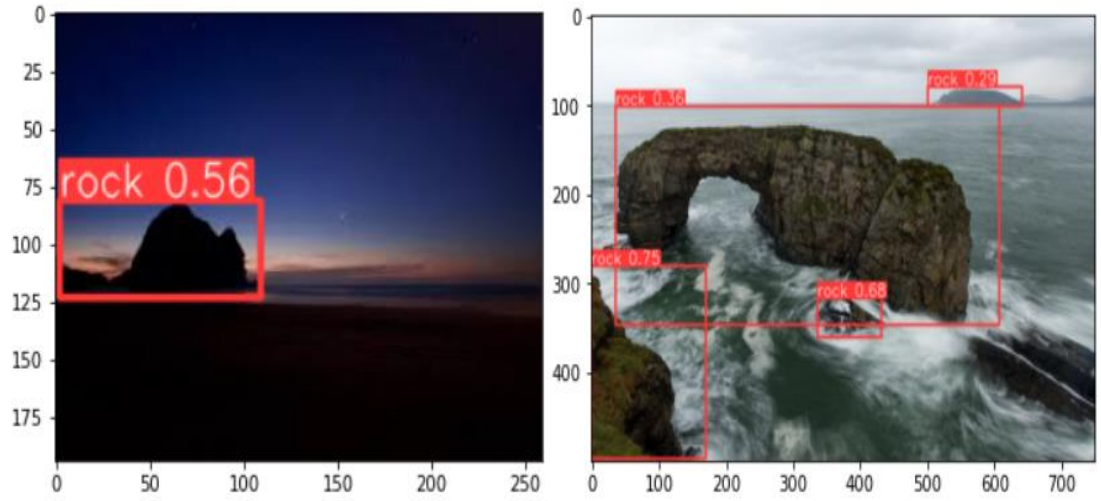

Figure 0.8: Results of the model detections, with the 20 images of rocks in the training dataset.

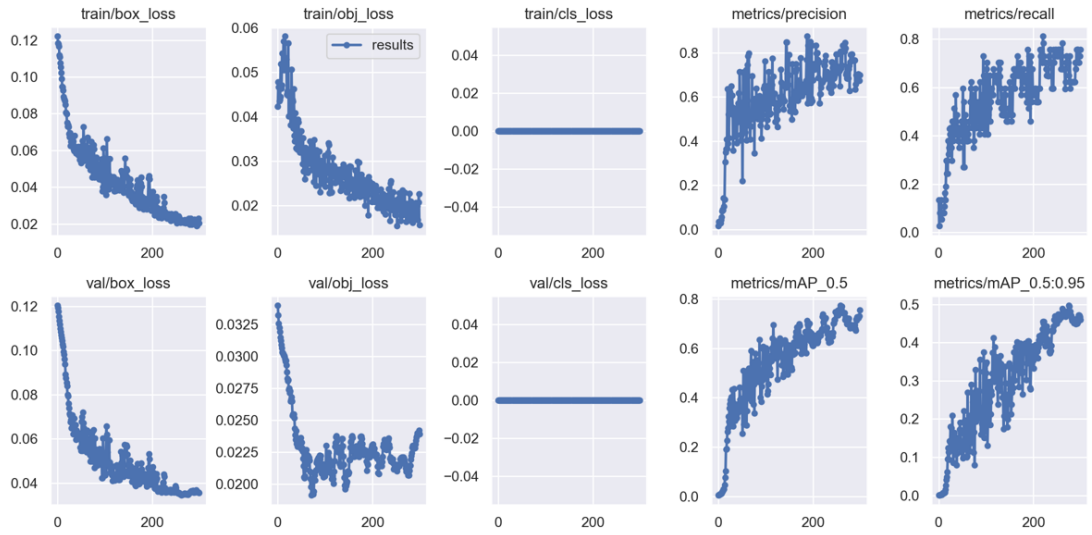

Figure 0.9: Metric results for a training dataset with 50 images of rocks.

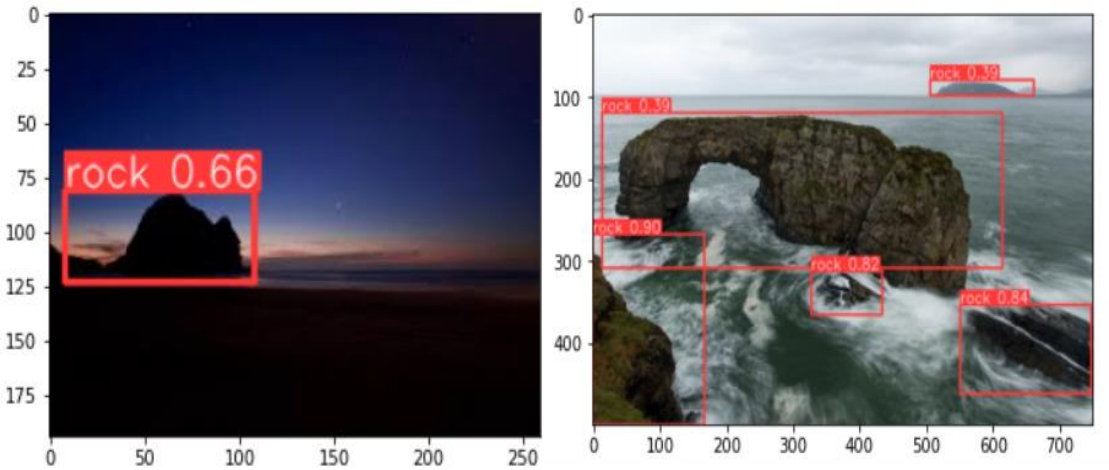

Figure 0.10: Results of the model detections, with the 50 images of rocks in the training dataset.

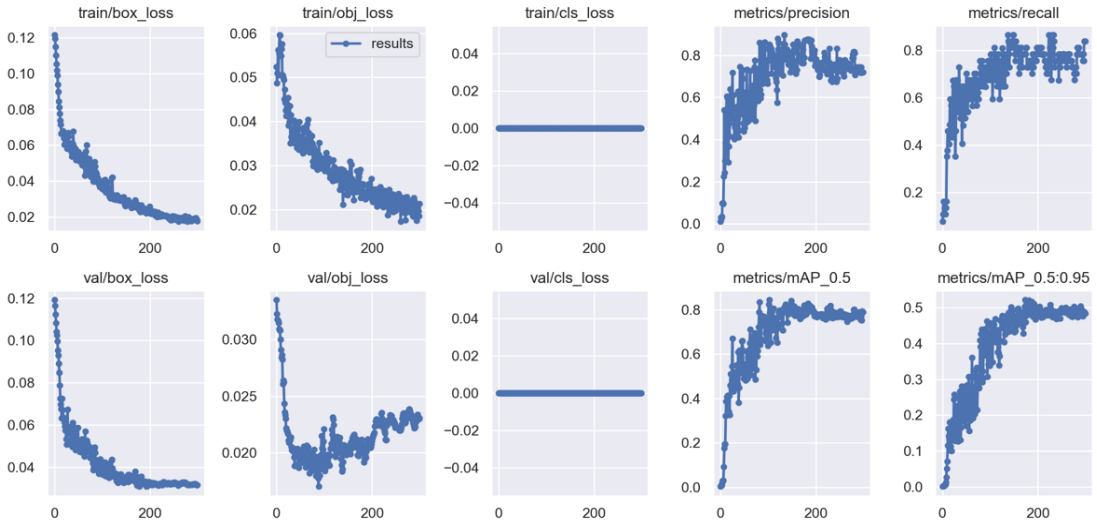

Figure 0.11: Metric results for a training dataset with 100 images of rocks.

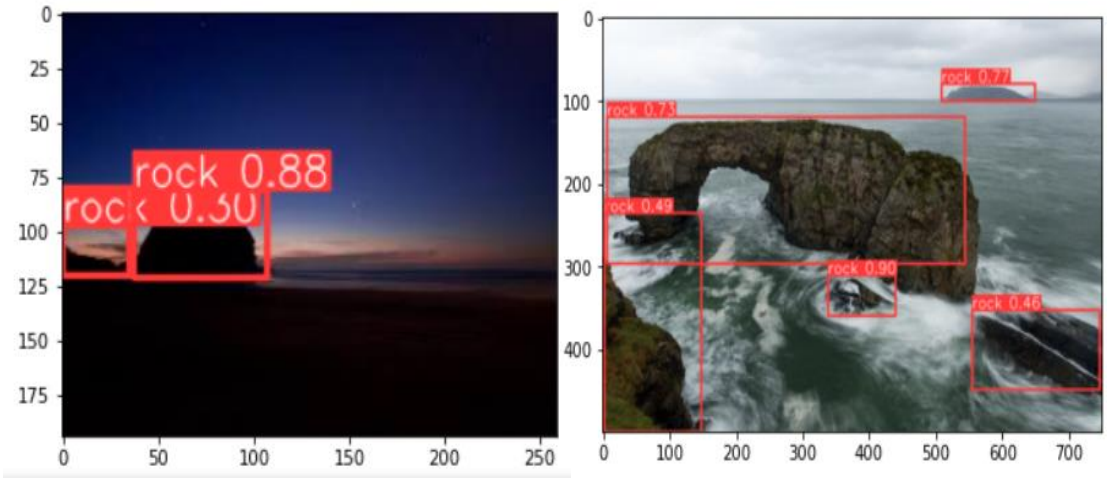

Figure 0.12: Results of the model detections, with the 100 images of rocks in the training dataset.

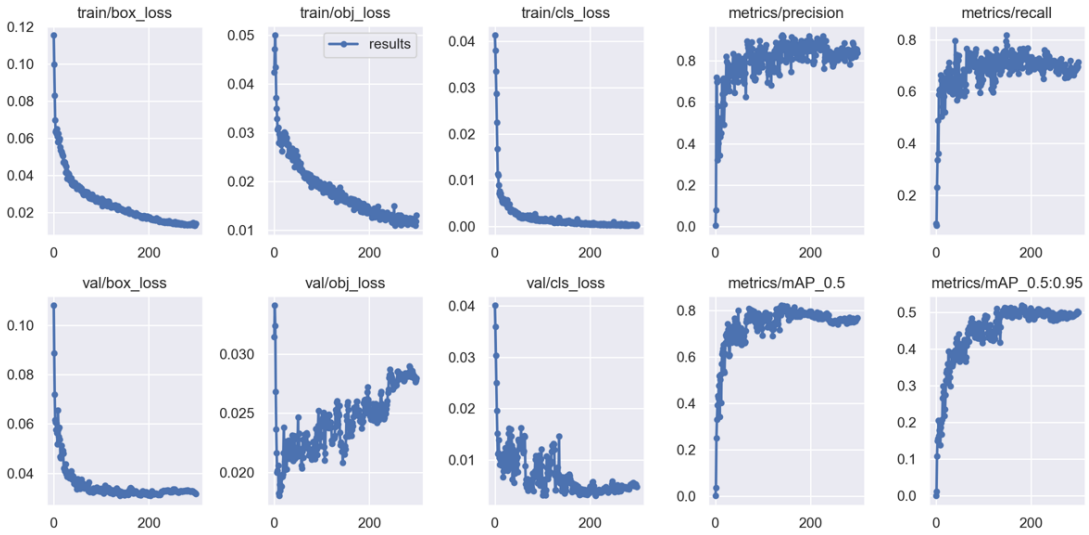

Figure 0.13: Metric results for a training dataset with 400 images trained on the medium-size network.