



**TECHNICAL UNIVERSITY OF CRETE
ELECTRICAL AND COMPUTER ENGINEERING DEPARTMENT**

**Selective extraction and identification of malicious traces in imaging
data of
endoscopic capsule, by using machine learning**

by

Christos Barpagiannis

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE
DIPLOMA OF

ELECTRICAL AND COMPUTER ENGINEERING

2019

THESIS COMMITTEE

Professor Dr. Michael Zervakis

Professor Dr. Georgios Stavrakakis

Dr. Eleutheria Sergaki

Χανιά, 2019

ACKNOWLEDGEMENTS

First of all I would like to thank my advisor from Technical University of Crete, Dr. Eleftheria Sergaki, for her guidance and support throughout this work and beyond that.

I also wish to extend my gratitude to my teacher Prof. Michael Zervakis for introducing me to the theory of Image analysis and supporting me in my Thesis and I would like to express gratitude to my committee member Prof. Georgios Stavrakakis. Both of these professors are excellent in their respective classes.

Also I would like to thank Prof. Andreas Polidorou, Professor of Surgery and Endoscopic Surgery, Director, Endoscopic Section B of Surgery Clinical Department of Health, School of Public Health Sciences (National and Kapodistrian University of Athens).

Prof. George Karamanolis, Associate Professor of Gastroenterology, University Gastroenterology Clinic, Department of Health, School of Public Health Sciences(National and Kapodistrian University of Athens), U.K. Athens "Laiko".

Spyridon Zouridaki, Gastroenterologist.

University Gastroenterology Clinic of the University of Athens School of Medicine Athens "Laiko".

University 2nd Surgery Clinic, Therapeutic Endoscopy Unit, University of Athens at "Aretaio" Hospital.

University Gastroenterology 2nd Department, Education Clinic, University of Athens,"Atticon" Hospital.

University Gastroenterology 2nd Department, Aristotle University of Thessaloniki, Thessaloniki, Greece, Hippocrates Hospital.

"EUROMEDICA" Thessaloniki General Clinic.

Additionally I would like to thank all my close friends who made my time studying in Chania easier and smoother.

Last but more importantly I would like to thank my family which supported me all this time.

TABLE OF CONTENTS

Abstract	6
1. Introduction	13
1.1. Objectives of the study	13
1.2. Outline of the dissertation	13
2. Endoscopic Capsule	15
2.1. Bowel Bleeding and Capsule Endoscopy	15
2.2. The Study Today on Endoscopic Capsule with CNN	17
3. Description of dataset (materials and methods)	19
3.1 Finding the image dataset	19
3.2. Preprocessing the data	20
3.3. Methods of processing	20
3.4. Methods we used	21
4. Building and Training Convolutional Neural Networks used in Thesis	27
4.1. Basic Knowledge: Introduction to Neural Networks	27
4.2. Basic Knowledge: Types Of Neural Networks: Multilayer Perceptrons MLPs, Convolutional CNN, Recurrent Neural RNN	30
4.2.1. Multilayer Perceptrons	31
4.2.2. Recurrent Neural Networks	32
4.2.3. Convolutional Neural Networks	33
4.3. Tools used: Tensorflow, Keras API, Google Colab	34
5. CNN Models Used in Thesis	39
5.1. Technics we used to improve CNN performance: Dropout, Augmentation, Transfer learning - Fine tuning	41
5.2. Our Implementation of the CNN Models	55
6. Results and Future Work	67
6.1. Compare The Results	67
6.2. Conclusions	71

7. Future Work	73
8. REFERENCES	75
Annex 1: Accuracy and Loss	78

Περίληψη

Σήμερα, η ενδοσκόπηση του λεπτού εντέρου με τη χρήση ενδοσκοπικής κάψουλας (CE), αποτελεί το πιο αξιόπιστο μέσον στη διάγνωση ασθενών που πάσχουν από αιμορραγία λεπτού εντέρου (Small bowel bleeding -SBB). Ο ασθενής καταπίνει την ειδικά σχεδιασμένη ηλεκτρονική κάψουλα, η οποία κινείται μέσω της γαστρεντερικής οδού (GI) φωτογραφίζοντας το περιβάλλον της. Ο χρόνος που χρειάζεται η κάψουλα για να διανύσει το λεπτό έντερο κυμαίνεται, από μια έως πέντε ώρες, ανάλογα τον ασθενή. Οι φωτογραφίες μεταφέρονται με τη βοήθεια λογισμικού και υπό μορφή βίντεο σε H/Y για την ανάγνωσή τους από τον ειδικευμένο ιατρό, προς αρωγή της διάγνωσης. Η μείωση του χρόνου ανάγνωσης του βίντεο κατά τη διαγνωστική φάση επιτυγχάνεται αυτόματα με την χρήση αλγορίθμων που βασίζονται κυρίως στην απαλοιφή όμοιων εικόνων, καθώς και εικόνων που απεικονίζουν με σιγουριά υγιείς ιστούς. Οι αλγόριθμοι Μηχανικής Μάθησης και ιδιαίτερα των Νευρωνικών Δικτύων μπορεί να αποτελέσουν σημαντικότατο υποστηρικτικό ιατρικό εργαλείο αυτόματης ιατρικής διάγνωσης. Στην παρούσα διπλωματική αναπτύξαμε τρία διαφορετικά Νευρωνικά μοντέλα Deep Learning Convolutional NN, τα οποία πετυχαίνουν αυτόματη διάγνωση αγγειεκτασίας και αιμορραγίας από δεδομένα εικόνων CE και τις ταξινομούν σε δύο κλάσεις, υγιείς και μη υγιείς. Τα τρία διαφορετικά μοντέλα CNN περιέχουν επίπεδα Conv2D, Max Pooling. Όλα τα μοντέλα στο τέλος έχουν τον classifier ο οποίος αποτελείται από flatten layer και fully connected layers. Στο τελευταίο fully connected layer χρησιμοποιήσαμε για activation την sigmoid function σε αντίθεση με όλα τα υπόλοιπα επίπεδα που χρησιμοποιήσαμε την συνάρτηση ReLU. Σε αυτά τα τρία μοντέλα έγινε σταδιακά εφαρμογή συνδυασμού διαφόρων τεχνικών προκειμένου να αξιολογήσουμε την αποτελεσματικότητά τους στην βελτίωση των CNN μοντέλων μας. Εφαρμόσαμε μεθόδους όπως την Augmentation που αυξάνει τα δεδομένα με εικόνες που παράγει από αυτές του σετ με το να τις περιστρέφει και να τις μετατοπίζει, και μεθόδους που αλλάζουν τη δομή των μοντέλων. Εφαρμόσαμε την Dropout που αφαιρεί προσωρινά μονάδες του Νευρωνικού μοντέλου μαζί με τις διασυνδέσεις τους και την Transfer learning χρησιμοποιώντας τα ήδη εκπαιδευμένα μοντέλα VGGNet σε δύο εκδοχές VGG16 και VGG19 και το ResNet. Τα οποία έχουν εκπαιδευτεί σε μεγάλο αριθμό διαφορετικής φύσης δεδομένων. Προκειμένου να εξοικονομηθεί υπολογιστικός χρόνος δοκιμάστηκαν και διαφορετικές τεχνικές τροφοδότησης των δεδομένων στα CNN μοντέλα. Στην παρούσα Διπλωματική εργασία χρησιμοποιούνται τα δεδομένα προηγούμενης Διπλωματικής εργασίας στην Σχολή ΗΜΜΥ του ΠΚ, (κ. Α. Πολυδώρου) που προέκυψαν από 171 βίντεο CE διαφορετικών ασθενών από πέντε ελληνικά νοσοκομεία. Σε συνεργασία με γιατρό χειρουργό γαστρεντερολόγο έγινε επιλογή συνολικά 3800 εικόνων από διαφορετικής μορφής αμαγγειοεκτασίες και διαφορετικής έκτασης αιμορραγιών, από διαφορετικά σημεία του GI, οι οποίες περιλαμβάνουν κυρίως δύσκολες περιπτώσεις διάγνωσης αλλοιώσεων. Δεν συμπεριλήφθηκαν περισσότερες από μια εικόνες από κάθε αλλοίωση. Μετά από διαδοχικές δοκιμές διαφορετικού πλήθους και συνδυασμού των εικόνων, τα καλύτερα αποτελέσματα εκπαίδευσης έγιναν χρησιμοποιώντας ένα ισορροπημένο σετ εκπαίδευσης ελάχιστου αριθμού 1000 εικόνων, αποτελούμενου από 500 υγιείς και 500 με αιμορραγία ή αμαγγειοεκτασίες. Ο

πειραματισμός αυτόματης διάγνωσης έγινε σε επιλεγμένες εικόνες από 33 βίντεο που δεν χρησιμοποιήθηκαν στην φάση της εκπαίδευσης. Αυτό ήταν ένα σετ δοκιμής 100 εικόνων, από 50 υγιείς και 50 με αιμορραγία ή αιμαγγειοεκτασίες. Τα δεδομένα μας αφορούν τιμές έντασης και στα τρία κανάλια του χρωματικού μοντέλου RGB της κάθε εικόνας. Οι συνολικές εκδοχές των τριών CNN μοντέλων μας ήταν 22. Αυτές οι εκδοχές μοντέλων συγκρίθηκαν με την μέτρηση μεταξύ άλλων και της ευαισθησίας και της ακρίβειας τους. Διαπιστώθηκε ότι στην περίπτωση μας δεν βοήθησε η τεχνική Transfer Learning χρησιμοποιώντας τα VGGNet , ResNet, ενώ η Dropout και Augmentation βοήθησαν αρκετά στην απόδοση των CNN μοντέλων μας. Η καλύτερη απόδοση αυτόματης διάγνωσης μοντέλου μας κατά τον έλεγχο του σε συλλογή 100 δύσκολα ανιχνεύσιμων εικόνων έδωσε Sensitivity 90%, Accuracy 91%, Specificity 92%, Precision 91.8%, FPR 8%, FNR 10%. Σε σύγκριση με άλλες ερευνητικές ομάδες που ανέπτυξαν CNN για το ίδιο πρόβλημα, έχουν ανακοινωθεί και καλύτερα αποτελέσματα χωρίς όμως αυτό να επιτρέπει την σύγκριση των αλγορίθμων, λόγω του ελέγχου τους σε διαφορετική συλλογή δεδομένων. Σε σύγκριση με την μέθοδο μέτρησης χρώματος HSV που αναπτύχθηκε στην Διπλωματική εργασία του κ. Α. Πολυδώρου και δοκιμάστηκε στα ίδια δεδομένα, τα αποτελέσματα της μεθόδου μέτρησης χρώματος ήταν καλύτερα από αυτά της μεθόδου CNN (Sensitivity 99%, Accuracy 96.1%, Specificity 93.2%, Precision 83.3%, FPR 20%, FNR 0%).

Abstract

Nowadays, the small bowel capsule endoscopy (SBCE) is the most reliable way to diagnose patients suffering from small bowel bleeding (SBB). The patient swallows a specially designed electronic capsule, which moves through the gastrointestinal (GI) tract, photographing the environment, as it moves on. The time the capsule needs to traverse the small intestine ranges from one to five hours, depending on the situation. The photos are transferred to a computer in order to be reviewed by the doctor, in a form of a video stream, for further diagnosis. Visualization of a massive number of images is a very time consuming and tedious task for gastroenterologists. Decreasing the time of reading the video during the diagnostic phase by the medical staff can be achieved automatically by using available software that employs algorithms, mainly based on the deletion of similar images and images that do not contain the red color (indicating blood). Machine Learning (ML) and Artificial Neural Networks (ANNs) have a crucial importance as supporting tools for doctors

In the present Thesis we have developed three different Deep Learning Convolutional NN models, which achieve an automatic diagnosis of vasculature and bleeding from CE image data and classify them into two classes, healthy and unhealthy. The three different CNN models contain Conv2D, Max Pooling layers. All models at the end have a classifier consisting of a flatten layer and fully connected layers. In the last fully connected layer we used the sigmoid function for activation as opposed to all other layers we used with the ReLU function. These three models were gradually applied to a combination of different techniques to evaluate their effectiveness in improving our CNN models. We have implemented methods like Augmentation that increase the data with images it produces from the set by rotating and shifting them, and methods that change the structure of the models. We implemented Dropout which temporarily removes neuronal model units along with their connections and the method Transfer Learning using the already pre-trained models versions VGG16 and VGG19 (Visual Geometry Group) from University of Oxford and the ResNet which are trained in a large set of images e.g. the ImageNet dataset. In order to save computational time, different data feed techniques were also tested on the CNN models.

This Thesis took in account data available in the Thesis of Mr. A. Polydorou, at the ECE Department of TUC in 2018. These data are extracted from 171 CE videos of different patients from five Greek hospitals. In collaboration with a gastrointestinal surgeon doctor, a total of 3800 images were selected from different forms of angioectasia and different haemorrhages, from different sites of the GI (gastrointestinal), which include mainly difficult cases of diagnosis of lesions.

No more than one image from each lesion was included. After successive trials of different numbers and combination of images, the best training results were achieved by using a balanced training set of at least 1000 images, consisting of 500 healthy and 500 bleeding or with angioectasia. The automatic diagnostic experiments were performed on selected images from 33

videos that were not used during the training/validation procedure. The testing set consists of 100 images, including 50 normal and 50 bleeding or angioectasia. Our data are the values of the RGB components for each pixel.

The total versions of our three CNN models are 22. Performance metrics such as accuracy (Acc), sensitivity (Sen), and specificity (Spe) were computed to evaluate the effectiveness of the proposed models. We find out that the methods Dropout and Augmentation improve the performance of our models but in the case of Transfer Learning the usage of VGGNet, ResNet we had the opposite results.

The best performance of the model based on unseen data achieved 90% Sen, 92% Spe, 91.8% Precision, 8% FPR, 10% FNR. Compared to other research teams that have developed CNN for the same problem, better results have been reported, but this does not allow the algorithms to be compared because of their control over different data collection. The results of the present work compared to the work developed by Mr. A. Polydorou Thesis (diagnosis based on handcrafted color metrics of the HSV color space), the color metrics results were better than those of our CNN method (99% Sen, 96.1% Acc, 93.2% Spe, 83.3% Precision, 20% FPR, 0% FNR).

Keywords:

Capsule Endoscopy (CE), Small bowel bleeding (SBB), gastrointestinal (GI) small bowel capsule endoscopy (SBCE), Machine Learning (ML), Artificial Neural Network ANN, Deep Learning, Convolutional Neural Network CNN, FPR False Positive Ratio, FNR False Negative Ratio, Convolutional Neural Network CNN, Dropout Transfer Learning Augmentation angioectasia Deep Learning VGGNet, ResNet, ReLU Sigmoid function.

Acronym:

ACC:	Accuracy
ANN:	Artificial Neural Networks
DL:	Deep Learning
FN:	False Negative
FNR:	False Negative Ratio
FP:	False Positive
FPR:	False Positive Ratio
FPS:	Frames per Second
GI:	Gastrointestinal
LED:	Light-Emitting Diode
ML:	Machine Learning
RGB:	Red Green Blue color model
SBB:	Small bowel bleeding
SBCE:	Small bowel Capsule Endoscopy
TN:	True Negative
TP:	True Positive
WCE:	Wireless Capsule Endoscopy

1. Introduction

The small bowel (or small intestine) is the longest portion of the gastrointestinal (GI) tract. It is called "small" because it is thin or narrow compared to the "large" bowel (also known as the colon), but it is much longer than the large bowel (6 m on average). The small intestine is involved in nutrient absorption from food. The Capsule Endoscopy is a method to detect Small Bowel Bleeding (or SBB) that has been used very frequently in recent years. The major problem with this method is the very large data of images as a capsule can take two images per second (2 fps) and the whole process can take many hours, as much as about 8 hours in total in the small and large bowel. It is known that the doctor in the end has 30.000 - 50.000 images to examine, however, there are technologies that make these tests easier but still it takes a lot of time for a doctor to examine the images. An experienced doctor can finish all the dataset in about 30 - 40 minutes and an inexperienced in a much longer time just for one patient. It takes a long time for a doctor to see all the images of every patient in a day and this process increases the percentage of false diagnosis. So an application is required to detect the bleeding images from these large datasets in order to make the job of a doctor easier and more accurate.

1.1. Objectives of the study

The main objective of this thesis is to examine different classification techniques for the problem of bleeding detection in Small Bowel from images taken from endoscopic capsule using Neural Networks and specific Convolutional Neural Networks.

- Processing the images in the right form and getting the datasets ready in order to pass them into the Convolutional Neural Network.
- Finding and creating CNN models modifying them and using various methods to increase the performance of these models.
- Comparing the performance of the models in field-test

1.2. Outline of the dissertation

- **Chapter 1** gives an introduction to the problem statement and the objectives of the study.
- **Chapter 2** gives some information about endoscopic capsule, the study and applications today on this method and the contribution of CNNs to this field.
- In **Chapter 3** we describe the processing and the techniques we use on the dataset in order for them to be ready for the CNNs.
- In **Chapter 4** we describe the Neural Networks and the CNNs and also the tools and libraries were used.
- In **Chapter 5** we will see the models we created and the methods which apply on them to achieve better performance.

- In **Chapter 6** we compare the results of our models and see which one has the best results, we also extract the conclusions on the CNNs and the methods.

2. Endoscopic Capsule

The images are taken from endoscopic capsule and especially from PillCam SB3.

The Pillcam™ SB is 11x26 mm, and weighs 3 gr. It has one camera with FOV 156 and led lights on one side. It takes 2 - 6 fps depending on the speed that moves inside the bowel. The camera in the entire process takes up to 50.000 images and transmits them wirelessly and finally converts them into a video. All this is inside a shell that is made of plastic, biocompatible and resistant to digestive fluids.[1]

- CMOS cam, 256x256 pixel.
- 6 white light LEDs.
- One short-focus lens.
- Two mercury free silver oxide batteries of 3V with a life span of 8-12 h.
- ASIC transmitter
- Ultra-high frequency band radio telemetry

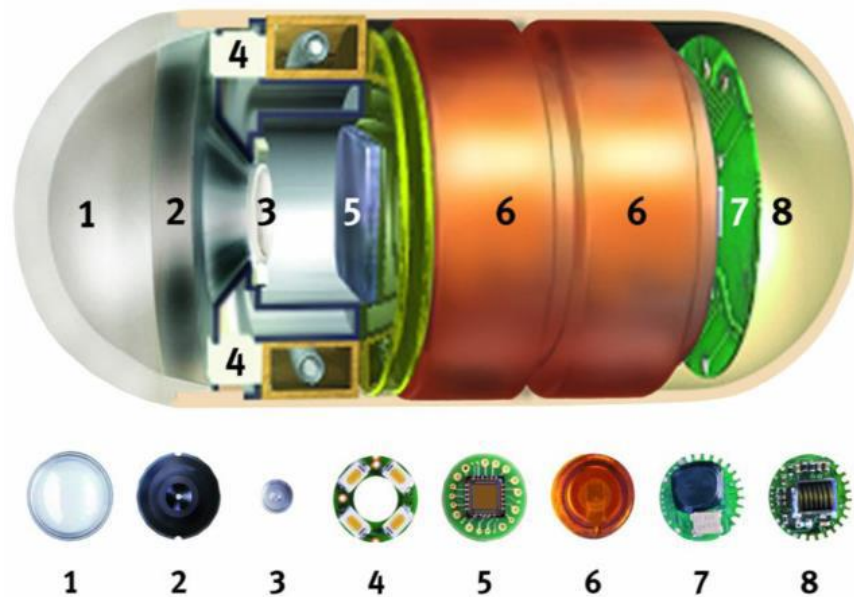


Figure 2.1 Pillcam™ SB2: (1) Optical dome, (2) Lens holder (3) short-focus lens, (4) four white light leds, (5) CMOS image sensor, (6) two batteries, (7) MCU (microcontrol unit), (8) RF (radio frequency) transmitter.

2.1. Bowel Bleeding and Capsule Endoscopy

GI (gastrointestinal) bleeding occurs when an abnormality on the inner lining begins to bleed. Approximately 5% of all GI bleeding comes from the small bowel. Abnormal blood vessels (arteriovenous malformations or AVMs) cause 30 to 40% of bleeding. AVMs are the main source

of bleeding in patients over the age of 50 years. Tumors (benign and malignant), polyps, Crohn's disease, and ulcers are some of the other causes of bleeding.

Multiple tests can be used to diagnose and treat the source of small bowel bleeding, including: endoscopy, enteroscopy, x-ray studies, capsule endoscopy, deep small bowel enteroscopy, and intraoperative enteroscopy. AVMs can typically be treated with cautery delivered through an endoscope or enteroscope. Tumors (benign and malignant) can be biopsied and have their location marked using endoscopy, but surgery is typically required to take them out.

There some symptoms that we can see if someone has bleeding. Bleeding from the small bowel may be slow or fast. When the bleeding is slow, it may cause anemia (a low blood count). When the bleeding is slow it may not be visible in the stool. Anemia may cause symptoms such as tiredness and shortness of breath, but many people have no symptoms. If the bleeding is fast it is called a hemorrhage. People with hemorrhage may notice blood when they move their bowels, or their bowel movements may be black and tarry.

The causes of bleeding in the small bowel are different from those in the colon or the stomach. The most common causes of bleeding from the colon are polyps, diverticulosis (small outpouchings in the wall of the colon), or cancer. Upper GI (esophagus, stomach, or duodenum) bleeding is most often due to ulcers.

In the small bowel, 30 to 40% of bleeding is caused by abnormal blood vessels in the wall of the small bowel. These abnormal blood vessels have many names, including angioectasias, angiodysplasias, or arteriovenous malformations (AVMs). In people over the age of 50 years, AVMs are the most common cause of small bowel bleeding. Other causes of small bowel bleeding include benign (non-cancerous) and malignant (cancerous) tumors, polyps, Crohn's disease (a type of inflammatory bowel disease), and ulcers.[27]

So for the diagnosis there are multiple tests for evaluating the small bowel. In most cases, the first step is endoscopy and/or enteroscopy. If that fails to find the source of bleeding, a common next step is capsule endoscopy. X-ray options include a small bowel follow-through or a computed tomographic scan (also known as a CT or CAT scan) of the small bowel. Deep small bowel enteroscopy can now be performed using special scopes with inflatable balloons and/or overtubes. The final option is intraoperative enteroscopy. Intraoperative enteroscopy requires surgery and is usually only done if the other tests are negative.[27]

As we said the purpose of this thesis is to detect the bleeding images extracted from endoscopic capsule.

Capsule endoscopy uses a device that is about the size of a large pill. It is 1-1/8 inches long and 3/8 inches wide (26 mm x 11 mm). It is made up of a battery with an 8-hour lifespan, a strong light source, a camera, and a small transmitter. Once swallowed, the capsule begins transmitting images of the inside of the esophagus, stomach, and small bowel to a receiver worn by the patient. The capsule takes two pictures per second, for a total of about 55,000 images. After 8 hours, the patient returns the receiver to the doctor who downloads the information to a

computer and then can review in detail the 8 hours of pictures of the capsule passing through the intestine. The patient passes the capsule through the colon and it is eliminated in the stool and discarded.

The capsule is generally safe and easy to take, however, rarely the capsule can get stuck in the small intestine. This may happen if there has been prior abdominal surgery causing scarring or other conditions that cause narrowing of the small intestine. If the capsule becomes stuck, endoscopic or surgical removal is necessary. However, if the capsule becomes stuck, there is a good chance that it is stuck at the place where the bleeding is coming from. So the procedure to get the capsule (endoscopy or surgery) may be able to treat the bleeding site at the same time. In about 15% of exams, the capsule does not view the entire small bowel prior to the battery running out and may need to be repeated.

Like x-rays, the capsule cannot be used to take biopsies, apply therapy, or mark abnormalities for surgery. Moreover, the capsule cannot be controlled once it has been swallowed, so once it has passed a suspicious area, it cannot be slowed to better look at the area. Despite these limitations, capsule endoscopy is frequently the test of choice for finding a source of small bowel bleeding if standard endoscopy has failed to do so because it is able to look at the whole small bowel and is an easy test for most people to do.

Overall, in patients with occult bleeding (blood is microscopically present in the stool, but the stool looks normal), capsule endoscopy finds a cause of bleeding in up to 67% of patients. In cases of overt bleeding (blood is seen in the stool or the stool is black and tarry), the results are highly variable. If the bleed happened in the past, the yield may be as low as 6%. If, however, the doctor believes that there is active bleeding occurring at the time of the test, the yield is >90%.[27]

2.2. The Study Today on Endoscopic Capsule with CNN

As we said with capsule endoscopy the gathering data for diagnosis is huge 30.000 - 50.000 images just for one patient, so unlike wired endoscopy, capsule endoscopy requires additional time for a clinical specialist to review the operation and examine the lesions. To reduce the tedious review time and increase the accuracy of medical examinations, various approaches have been reported based on artificial intelligence for computer-aided diagnosis. Recently, deep learning-based approaches have been applied to many possible areas, showing greatly improved performance, especially for image-based recognition and classification.[28]

Capsule endoscopy (CE) has been developed to obtain endoscopic imaging of the entire small bowel. Since its introduction, clinical practice guidelines have been established for several conditions, including unexplained obscure gastrointestinal (GI) bleeding, small bowel Crohn's disease, small bowel tumors, and other miscellaneous abnormalities. Because endoscopy images are acquired by imaging sensors, most computer vision technologies can be applied directly. Most of all, in the field of CE, various approaches based on artificial intelligence for computer-aided diagnosis have been undertaken to reduce the long review time.[28]

Based on artificial intelligence, specifically computer vision and machine learning methodologies, various computational methods including algorithms for detecting hemorrhage and lesions,

reducing review time, localizing capsules or lesions, and enhancing video quality have been proposed to improve efficiency and diagnostic accuracy. For detecting hemorrhages and lesions, color and texture information is usually used as a distinctive feature. Image features extracted from endoscopic images can be classified into target class using machine learning algorithms such as a support vector machine (SVM), neural network, or binary classifier. Although previous methods based on machine learning classifiers with invariant features showed concordant results for detecting various lesions, they have limitations related to insufficient training and testing databases and problems with specific feature design.[28]

Deep learning–based lesion detection and classification methods for flexible endoscopy have been presented. The ability of computer-assisted image analysis with a deep learning–based method, more specifically convolutional neural networks (CNN), has been tested to detect polyps, a surrogate for adenoma detection rate.[28] In table 2.1 below we can see the recent approaches with deep learning methods for Capsule Endoscopy (CE).

Table 1.

State-of-the-Art Deep Learning Based Methods for Capsule Endoscopy

Study	Class	No. of training/testing images	No. of patients or videos	Features	Accuracy	Sensitivity/Specificity
Zou et al. (2015) [27]	Localization ^{a)}	60K/15K	25 patients	Alexnet	95.5%	No info.
Segui et al. (2016) [28]	Scene classification ^{b)}	100K/20K	50 videos	CNN	96.0%	No info.
Jia et al. (2016) [29]	Bleeding	8.2K/1.8K	No info.	Alexnet	99.9%	99.2%/No info.
Li et al. (2017) [30]	Haemorrhage	9,672/2,418	No info.	LeNet	100%	98.7%/No info.
				AlexNet		
				GoogLeNet		
				VGG-Net		
Yuan et al. (2017) [32]	Polyp ^{c)}	4,000 (No info.)	35 patients	SSAE	98.0%	No info.
Iakovidis et al. (2018) [34]	Various lesions ^{c)}	465/233	1,063 volunteers	CNN	96.3%	90.7%/88.2%
		852/344	No info.			
He et al. (2018) [33]	Hookworm	400K/40K	11 patients	CNN	88.5%	84.6%/88.6%
Leenhardt et al. (2018) [31]	Angiectasia	600/600	200 videos	CNN	No info.	100%/96%

CNN, convolutional neural networks; SSAE, stacked sparse autoencoder.

^{a)}Localization, Localization of stomach, small intestine, colon.

^{b)}Scene classification, Scene classification of Bubble, wrinkle, turbid, wall, clear.

^{c)}Various lesions, Gastritis, Cancer, bleeding, ulcer.

Table 2.1 The various methods of deep learning for Capsule Endoscopy.[28]

3. Description of dataset (materials and methods)

3.1. Finding the image dataset

Our images have been provided by Alexios Polidorou Thesis (2018) and are a well balanced set [1], for bleeding detection. As we said the dataset is well balanced of 1000 images split into two sets of 500 images, healthy and unhealthy (bleeding).

Endoscopic capsules were used for these images such as Pillcam™ SB2 και Pillcam™ SB3. Altogether we had a set of 3.300 images with analysis of 576×576 pixel (2.800 unhealthy and 500 healthy) taken from 138 video clips. The final 500 bleeding images were taken randomly from the of 2.800 images and with the healthy we created the dataset. Also we created a test set of 100 images (50 healthy, 50 unhealthy), these images were extracted from 33 video clips. Video clips were taken from quite a number of patients male and female.

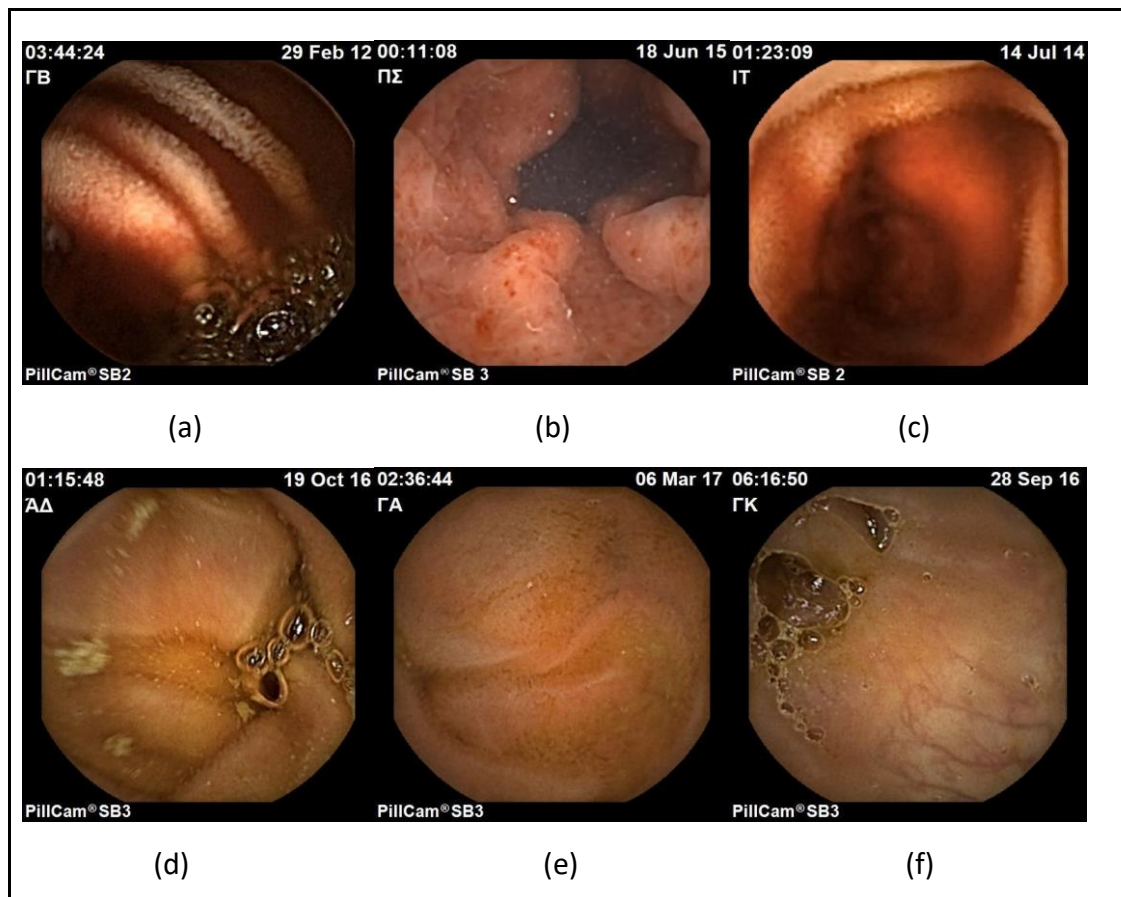


Figure 3.1 Examples of images, unhealthy (bleeding) (a), (b), (c) and healthy (d), (e), (f).

3.2. Preprocessing the data

The preparation and processing of data is the procedure that we handle and make the data ready with right form for the CNN.

Data preparation and preprocessing is very crucial work in CNN as we must be very careful how we feed our model because our results depend on how well we process our images.

3.3. Methods of processing

Neural network image recognition algorithms rely on the quality of the dataset – the images used to train and test the model. Here are a few important parameters and considerations for image data preparation.[3]

- **Image size**—higher quality images give the model more information but require more neural network nodes and more computing power to process.
- **The number of images**—the more data you feed to a model, the more accurate it will be, but ensure the training set represents the real population.
- **The number of channels**—grayscale images have 2 channels (black and white) and color images typically have 3 color channels (Red, Green, Blue / RGB), with colors represented in the range [0,255].
- **Aspect ratio**—ensure the images have the same aspect ratio and size. Typically, neural network models assume a square shape input image.
- **Image scaling**—once all images are squared you can scale each image. There are many upscaling and downscaling techniques, which are available as functions in deep learning libraries.
- **Normalizing image inputs**—ensures that all input parameters (pixels in this case) have a uniform data distribution. This makes convergence speedier when you train the network. You can conduct data normalization by subtracting the mean from each pixel and then dividing the outcome by the standard deviation.
- **Dimensionality reduction**—you can decide to collapse the RGB channels into a gray-scale channel. You may want to reduce other dimensions if you intend to make the neural network invariant to that dimension or to make training less computationally intensive.
- **Data augmentation**—involves augmenting the existing data-set, with perturbed types of current images, including scaling and rotating. You do this to expose the neural network

to a variety of variations. This way this neural network is less likely to identify unwanted characteristics in the data-set. (We discuss data augmentation with more details later).

- **Splitting**—the data set in train, validation and test set, also with the splitting we randomize the images in sets and put labels on them according to the condition that represent every image in that way we create classes, in our case (healthy, bleed).

3.4. Methods we used

As we mentioned before, we have a data set of 1000 images of 576×576 pixel split in two categories healthy and unhealthy (bleeding).

- First of all we renamed the healthy images to healthy.1, healthy.2 ... and the unhealthy to bleed.1, bleed.2 ... and see if any of the images are destroyed or corrupted.
- We noticed that all our images on each side had 60 pixels of black color that had no information for us to use to extract some values in order to help us with our problem. So the next step was to crop the images by 60 pixels on each side. Below in figure 3.2 is an example of an image that we cropped.

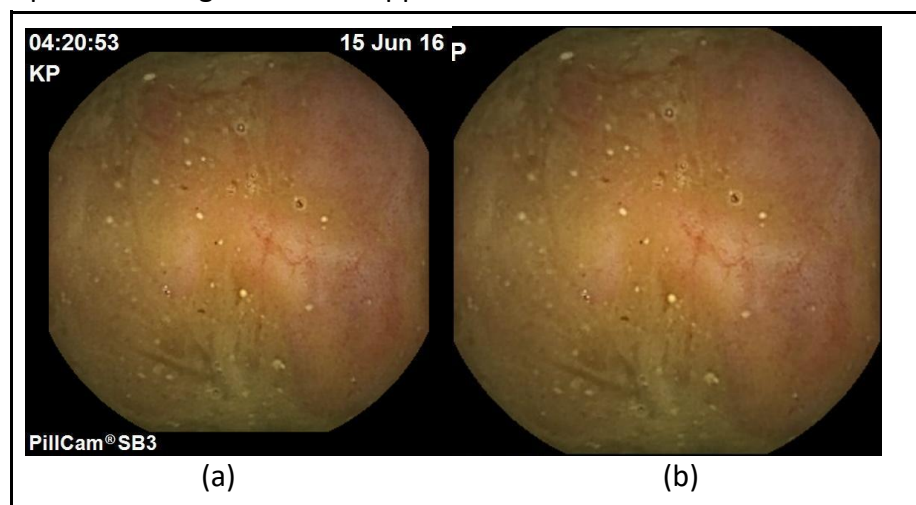


Figure 3.2 Example of healthy images, the original image 576×576 (a), the cropped image (b) 516×516.

We all know that CNN works well only when we have sufficiently large data. We have seen that CNN architecture failing when it is applied to small data. So it is very challenging how can we develop efficient CNN model on small data, because we have only 1000 images for training and validation. If someone considers that one common dataset for image classification of cats and dogs has 25.000 images we can understand the difference between this dataset and the one we have.

Because of large dataset a problem arises, for example if we want to load all of the images of a large dataset(cats vs dogs) into memory, we can estimate that it would require about 12

gigabytes of RAM. In our case we don't have a problem but we follow the processes methods as if we had large data.

We used two slightly different types of pre-processing the data to see if we had some difference in our models but the only difference was the speed of models.

In the first approach[5] we create a file path to our train and test data, next we grab all healthy and bleeding images of the train set and create a list of all data, next we randomly shuffle the dataset because right now our data is at this point all healthy, then all bleeding. This will usually wind up causing trouble too, as, initially, the classifier will learn to always predict healthy. Then it will shift over to bleeding, to always predict bleeding! Going back and forth like this is no good either.[6]

For the shape we definitely don't want the images that big (576x576), and also want the exact same shape for all of them. We use 224 by 224 for height and width and 3 channels. We don't use shorter image size because we don't want blurry images and the choice of 224x224 is for the process of transfer learning that we apply in our model to achieve better accuracy and this process requires height and width of 224x224. A colored Image is made up of 3 channels, i.e 3 arrays of red, green and blue pixel values. We could use 1 channel which would read our images in gray-scale format (black and white) but the difference between healthy and bleeding images is the value of red color so we stick to colored images.[5] Also we use RGB color space and not some others like HSV because in some researches different color space like RGB,HSV,CIE Lab, CIE XYZ which have been used in CNN models, show that RGB has slightly better results in accuracy[2]. Also in our thesis we emphasize on CNN models, how they work and what we can do to achieve better results.

The next step is to create two lists X, y. In X we hold the new training set with all resized images and y which will hold our training labels, we append for every healthy image 0 and for every bleeding 1 in y. After that X is now an array of image pixel values and y is a list of labels. Let's preview the first ten labels of y in figure 3.3 and the first image in X in figure 3.4.

```
▶ y[:10]
[0, 0, 1, 1, 0, 0, 1, 1, 0, 0]
```

Figure 3.3 The first 10 labels in **y**. We have six healthy images ("0") and four bleeding images ("1").

```
▶ x[0]
array([[1, 1, 1],
       [2, 2, 2],
       [1, 1, 1],
       ...,
       [3, 2, 2],
       [4, 2, 2],
       [1, 0, 2]],

      [[0, 0, 0],
       [1, 1, 1],
       [0, 0, 0],
       ...,
       [0, 0, 0],
       [0, 0, 0],
       [0, 0, 0]],

      [[0, 0, 0],
       [0, 0, 0],
       [0, 0, 0],
       ...,
       [0, 0, 0],
       [0, 0, 0],
       [0, 0, 0]])
```

Figure 3.4 The first image of **X** array is a healthy image according to the label list **y**.

So now we have **X** and **y** which are currently of type list (list of python array), we will convert these to **numpy** array so we can feed it into our model. In the figure 3.5 below we see our dataset.

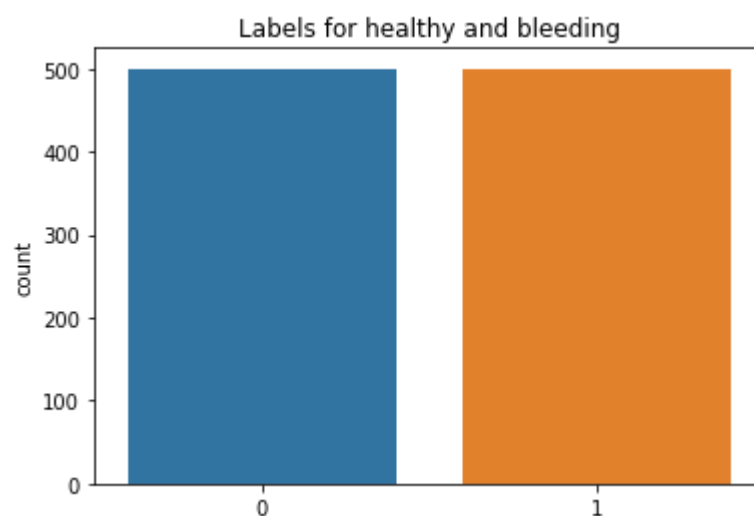


Figure 3.5 Our dataset 500 healthy images ("0") and 500 bleeding images ("1").

Let's see the shape of our dataset in Figure 3.6

```
▶ print("Shape of train images is:", X.shape)
  print("Shape of labels is:", y.shape)

📄 Shape of train images is: (1000, 224, 224, 3)
  Shape of labels is: (1000,)
```

Figure 3.6 Shape of our data.

The shape of our image array is important for the **keras** model we're going to build. The model takes as input an array of (height, width, channels). We can see that our image is a tensor of rank 4, or we could say a 4 dimensional array with dimensions 1000 x 224 x 224 x 3 which correspond to the batch size, height, width and channels respectively.

One more important thing we have to do is to split our data into train and validation set. This is one of the most important things to do before starting training the model. We are going to split 20% of the data to be assigned to the validation set and the other 80% to the train set. Figure 3.7

```
▶ print("Shape of train images is:", X_train.shape)
  print("Shape of validation images is:", X_val.shape)
  print("Shape of labels is:", y_train.shape)
  print("Shape of labels is:", y_val.shape)

📄 Shape of train images is: (800, 224, 224, 3)
  Shape of validation images is: (200, 224, 224, 3)
  Shape of labels is: (800,)
  Shape of labels is: (200,)
```

Figure 3.7 Shape of train and validation set for **X** and **y**.

Finally, before we build and start training our model we need to perform some Normalization of our data. We'll use an important module in keras called ***ImageDataGenerator*** which performs some important functions when we're feeding Images into our model during training. Rescale the pixel values (between 0 and 255) to the [0, 1] interval (neural networks perform better with normalize data).[5]

In figure 3.8 we can see the image after normalization and the values are between 0, 1.


```
train_generator [0]
(array([[[[0.00392157, 0.00392157, 0.00392157],
          [0.00392157, 0.00392157, 0.00392157],
          [0.00392157, 0.00392157, 0.00392157],
          ...,
          [0.01176471, 0.01176471, 0.01176471],
          [0.01960784, 0.00784314, 0.00784314],
          [0.01176471, 0.00392157, 0.00392157]],
        [[0., 0., 0.],
         [0., 0., 0.],
         [0., 0., 0.],
         ...,
         [0., 0., 0.],
         [0., 0., 0.],
         [0., 0., 0.]],
        [[0., 0., 0.],
         [0., 0., 0.],
         [0., 0., 0.],
         ...,
         [0., 0., 0.],
         [0., 0., 0.],
         [0., 0., 0.]]],
      ...])
```

Figure 3.8 Image after normalization all the values are between 0, 1.

Another way is to pre-processes photos into standard directories. We can load the images progressively using the Keras ImageDataGenerator class and *flow_from_directory()* API. This will be slower to execute but will run on more machines.[4] According to the creator of keras **Francois Chollet**, Keras ImageDataGenerator() lets us quickly set-up python generators that automatically turn image files into preprocessed tensors that can be fed directly into models during training. It performs the following functions for us easily[5]:

1. Decode the JPEG content to RGB grids of pixels.
2. Convert these into floating-point tensors.
3. Rescale the pixel values (between 0 and 255) to the [0, 1] interval (neural networks perform better with normalize data).

4. Helps us easily augment images. (An important feature we'll be using since we're training on a small data set).

This API prefers data to be divided into separate *train/* and *test/* directories, and under each directory to have a subdirectory for each class, e.g. a *train/bleed/* and a *train/healthy/* subdirectories and the same for test. Images are then organized under the subdirectories. So what we have to do is to create directories in Python using the *makedirs()* function and use a loop to create the *bleed/* and *healthy/* subdirectories for both the *train/* and *test/* directories. Next, we can enumerate all image files in the dataset and copy them into the *bleed/* or *healthy/* subdirectory based on their filename.

Additionally, we can randomly decide to hold back 20% of the images into the test dataset. This is done consistently by fixing the seed for the pseudorandom number generator so that we get the same split of data each time. After all these we can see in figure 3.9 the data structure into folders.

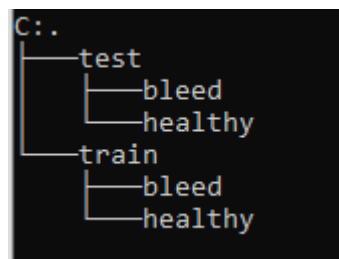


Figure 3.9 The structure of our dataset. Each directory has two subdirectories one for bleed images and one for the healthy.

Now that our dataset is in the right form we can continue to build our CNN models and feed them with our images.

In the next chapter we will discuss about CNN models and explain the structure of our models.

4. Building and Training Convolutional Neural Networks used in Thesis

4.1. Basic Knowledge: Introduction to Neural Networks

Artificial neural networks (ANN) or **connectionist systems** are computing systems that are inspired by, but not identical to, biological neural networks that constitute animal brains.[7]

A large ANN might have hundreds or thousands of processor units, whereas a mammalian brain has billions of neurons with a corresponding increase in magnitude of their overall interaction and emergent behavior.[8]

Such systems "learn" to perform tasks by considering examples, generally without being programmed with task-specific rules. For example, in image recognition, they might learn to identify images that contain cats by analyzing example images that have been manually labeled as "cat" or "no cat" and using the results to identify cats in other images. They do this without any prior knowledge of cats, for example, that they have fur, tails, whiskers and cat-like faces. Instead, they automatically generate identifying characteristics from the examples that they process.[7]

We have to mention that results getting better with more data and larger models, that in turn require more computation to train.[9]

An ANN is based on a collection of connected units or nodes called artificial neurons, which loosely model the neurons in a biological brain. Each connection, like the synapses in a biological brain, can transmit a signal to other neurons. An artificial neuron that receives a signal then processes it and can signal neurons connected to it.

In ANN implementations, the "signal" at a connection is a real number, and the output of each neuron is computed by some non-linear function of the sum of its inputs. The connections are called edges. Neurons and edges typically have a weight that adjusts as learning proceeds. The weight increases or decreases the strength of the signal at a connection. Neurons may have a threshold such that a signal is sent only if the aggregate signal crosses that threshold. Typically, neurons are aggregated into layers. Different layers may perform different transformations on their inputs. Signals travel from the first layer (the input layer), to the last layer (the output layer), possibly after traversing the layers multiple times.[7]

So a basic neural network is typically organized in layers. Layers are made up of a number of interconnected "nodes" which contain an "activation function". Patterns are presented to the network via the "input layer", which communicates to one or more "hidden layers" where the actual processing is done via a system of weighted "connections". The hidden layers then link to an "output layer" where the answer is output as shown in the figure 4.1 below.[8]

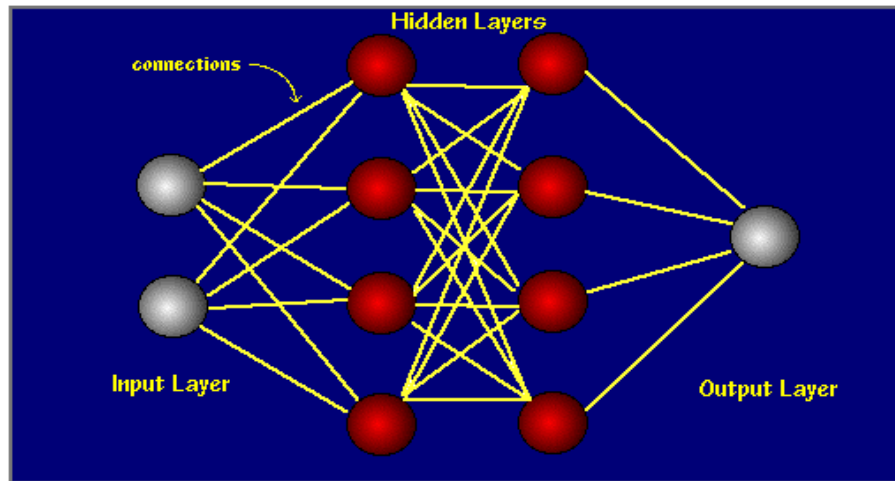


Figure 4.1 The basic Neural Network with one Input Layer one Output Layer and two Hidden Layers.

In Neural Network with one hidden layer means you just have a Neural Network. When you have two or more hidden layers you have got a deep Neural Network. The difference between a Neural Network and a deep Neural Network is if you have a single hidden layer, the model is going to only learn linear relationships. If you have many hidden layers, you can begin to learn non-linear relationships between your input and output layers.[10]

The original goal of the ANN approach was to solve problems in the same way that a human brain would. However, over time, attention moved to performing specific tasks, leading to deviations from biology. ANNs have been used on a variety of tasks, including:

- Computer Vision
- Speech Recognition
- Machine Translation
- Social Network Filtering
- Playing board and video games
- Medical Diagnosis

Even in activities that have traditionally been considered as reserved to humans, like painting.[7]

Now we have to dig a little further on how the Neural Networks works as we think that it may help to have a better understanding what the Hidden Layers might be doing, also will help us when we explain the models that we created.

To begin, we need to find some balance between treating neural networks like a total black box, and understanding every single detail with them.

For the sake of simplicity, we'll be using the most common "hello world" example for deep learning, which is the MNIST dataset (LeCoon, 1990) of 70,000 handwritten digits, 0 through 9. In figure 4.2 we can see an example of a handwritten digit image, 28x28 pixels, handwritten digits 0 through 9. It's 28x28 images of samples of handwritten digits. In figure 4.2 we can see an example of handwritten digit image.

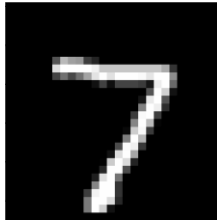


Figure 4.2 Sample of handwritten digit image of 7, is $28 \times 28 = 784$ pixels fit 784 neurons of the 1st layer of a NN, e.g. of 4 layers, with 10 neurons as output. The 784 input neurons can be connected with e.g. total 784×16 weights, means 13,002 weights and biases.

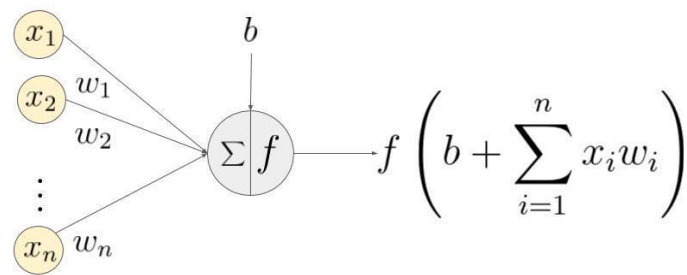
We start referring to the model that has 4 layers, one input layer, two hidden layers and output layer. The first layer is the input layer that has $28 \times 28 = 784$ neurons each one of these holds a number that represents the grayscale value of the corresponding pixel ranging to 0 for black pixels up to 1 for white pixels. This number inside the neuron is called activation and the neuron is lit up when its activation is a high number. The last layer has 10 neurons each one representing the digits, the activation in these neurons is again some number between 0 and 1 and presents how much the system thinks that the given image correspond to the even digit. There are also the hidden layers that have 16 neurons, which can't really say what they are doing maybe they recognize some patterns like loops or edges or part of these.

The way the network operates is the activations in on layer determine the activations of the next layer, so if you feed an image in the model lighting up all 784 neurons of the input layer according to the brightness of each pixel on the image, that pattern of activations causes some very specific pattern in the next layer which causes some pattern in the one after it, which finally gives some pattern in the output layer. And the brightest Neuron of that output layer is the network's choice for what digit this image represents.

Every neuron in one layer is connected with all neurons of the next layer, if we zoom in one neuron in the second layer we will see that the neuron is connected with all 784 neurons of the input layer. They are connected with weights, so the neuron that we are looking at has 784 weights, one for each connection. So the neuron holds a number that comes from the sum of the first layer activations multiplied by the weights that correspond to each connection.

When you compute the results it may be any number but for the network what we want is a number between 0 and 1, for this purpose we insert the "**Sigmoid Function**" which squeezes the results in range between 0 and 1.

The activation is basically the measure of how positive the relevant weighted sum is but we want to light up when some is slightly bigger than 0, so we add a number, called "**Bias**", to this weighted sum before plugging to the Sigmoid Function.



An example of a neuron showing the input ($x_1 - x_n$), their corresponding weights ($w_1 - w_n$), a bias (b) and the activation function f applied to the weighted sum of the inputs.

Figure from [32]

If we put all weights and biases for all the neural network we have $784 \times 16 + 16 \times 16 + 16 \times 10$ weights and $16 + 16 + 10$ biases that means we have 13,002 weights and biases. When we talk about learning it is to get the computer to find the right weights and biases so as to solve the problem.[11]

In the learning process most ANNs contain some form of "**learning rule**" which modifies the weights of the connections according to the input patterns that it is presented with. In a sense, ANNs learn by example as do their biological counterparts, as a child learns to recognize dogs from examples of dogs.

There are many different kinds of learning rules used by neural networks, most of "**learnings**" are supervised processes that occur with each cycle or "**epoch**" (i.e. each time the network is presented with a new input pattern) through a forward activation flow of outputs, and the backwards error propagation of weight adjustments. More simply, when a neural network is initially presented with a pattern it makes a random 'guess' as to what it might be. It then sees how far its answer was from the actual one and makes an appropriate adjustment to its connection weights. Finally once a neural network is 'trained' to a satisfactory level it may be used as an analytical tool on other data.

One last thing that we will see and in our models, is the possibility to over-train a neural network, which means that the network has been trained exactly to respond to only one type of input, which is much like rote memorization. If this should happen then learning can no longer occur and the network is referred to as having been "**grandmothered**" in neural network jargon terminology.[8]

4.2. Basic Knowledge: Types Of Neural Networks: Multilayer Perceptrons MLPs, Convolutional CNN, Recurrent Neural RNN

Now that we have seen some things about Neural Networks is time to see some types of them. There are many types of Neural Networks and there are thousands of types of specific neural

networks proposed by researchers as modifications or tweaks to existing models. Sometimes wholly new approaches. There are three classes of artificial neural networks that are very common

- Multilayer Perceptrons (MLPs)
- Recurrent Neural Networks (RNNs)
- Convolutional Neural Networks (CNNs)

These three classes of networks provide a lot of flexibility and have proven themselves over decades to be useful and reliable in a wide range of problems. They also have many subtypes to help specialize them to the quirks of different framings of prediction problems and different datasets. We will see when we can use each class of neural network and also we must say that most neural networks are flexible enough that they work (make a prediction) even when used with the wrong type of data or prediction problem.[12]

4.2.1. *Multilayer Perceptrons*

Multilayer Perceptrons, or MLPs for short, are the classical type of neural network. They are comprised of one or more layers of neurons. Data is fed to the input layer, there may be one or more hidden layers providing levels of abstraction, and predictions are made on the output layer, also called the visible layer.

MLPs are suitable for classification prediction problems where inputs are assigned a class or label.

They are also suitable for regression prediction problems where a real-valued quantity is predicted given a set of inputs. Data is often provided in a tabular format, such as you would see in a CSV file or a spreadsheet.

MLPs Used For:

- Tabular datasets
- Classification prediction problems
- Regression prediction problems

They are very flexible and can be used generally to learn a mapping from inputs to outputs. This flexibility allows them to be applied to other types of data. For example, the pixels of an image can be reduced down to one long row of data and fed into a MLP. The words of a document can also be reduced to one long row of data and fed to a MLP. Even the lag observations for a time series prediction problem can be reduced to a long row of data and fed to a MLP.

As such, if the data is in a form other than a tabular dataset, such as an image, document, or time series and testing an MLP on the problem. The results can be used as a baseline point of comparison to confirm that other models that may appear better suited add value.[12]

MLPs suggested Used On:

- Image data
- Text Data
- Time series data
- Other types of data

4.2.2. Recurrent Neural Networks

Recurrent Neural Networks, or RNNs, were designed to work with sequence prediction problems.

Sequence prediction problems come in many forms and are best described by the types of inputs and outputs supported.

Some examples of sequence prediction problems include:

- **One-to-Many:** An observation as input mapped to a sequence with multiple steps as an output.
- **Many-to-One:** A sequence of multiple steps as input mapped to class or quantity prediction.
- **Many-to-Many:** A sequence of multiple steps as input mapped to a sequence with multiple steps as output.

The Many-to-Many problem is often referred to as sequence-to-sequence, or seq2seq for short.

Recurrent neural networks were traditionally difficult to train.

The Long Short-Term Memory, or LSTM, network is perhaps the most successful RNN because it overcomes the problems of training a recurrent network and in turn has been used on a wide range of applications.

RNNs in general and LSTMs in particular have received the most success when working with sequences of words and paragraphs, generally called natural language processing.

This includes both sequences of text and sequences of spoken language represented as a time series. They are also used as generative models that require a sequence output, not only with text, but on applications such as generating handwriting.

RNNs Used For:

- Text data
- Speech data
- Classification prediction problems
- Regression prediction problems
- Generative models

Recurrent neural networks are not appropriate for tabular datasets as you would see in a CSV file or spreadsheet. They are also not appropriate for image data input.

Suggested Don't Use RNNs For:

- Tabular data
- Image data

RNNs and LSTMs have been tested on time series forecasting problems, but the results have been poor, to say the least. Autoregression methods, even linear methods often perform much better. LSTMs are often outperformed by simple MLPs applied on the same data.[12]

Perhaps Try RNNs on:

- Time series data

4.2.3. Convolutional Neural Networks

Convolutional Neural Networks, or CNNs, were designed to map image data to an output variable.

They have proven so effective that they are the go-to method for any type of prediction problem involving image data as an input.

The benefit of using CNNs is their ability to develop an internal representation of a two-dimensional image. This allows the model to learn position and scale invariant structures in the data, which is important when working with images.

CNNs Used For:

- Image data
- Classification prediction problems
- Regression prediction problems

More generally, CNNs work well with data that has a spatial relationship.

The CNN input is traditionally two-dimensional, a field or matrix, but can also be changed to be one-dimensional, allowing it to develop an internal representation of a one-dimensional sequence.

This allows the CNN to be used more generally on other types of data that has a spatial relationship. For example, there is an order relationship between words in a document of text. There is an ordered relationship in the time steps of a time series.

Although not specifically developed for non-image data, CNNs achieve state-of-the-art results on problems such as document classification used in sentiment analysis and related problems.[12]

Suggested Used CNNs On:

- Text data
- Time series data
- Sequence input data

4.3. Tools used: Tensorflow, Keras API, Google Colab

The software of this Thesis for building and training the NN is implemented in Python 3 by using the google Colaboratory, and the libraries of Keras API and Tensorflow.

Google Colaboratory

When we are training a Neural Network a major problem is how to process all of these data. It is well known for Neural Network GPU is needed, even in the case that we have small dataset of images without GPU the data processing will take much longer.

Colaboratory (Colab) is a Google research project created at 2011, for the implementation of large scale DL models on thousands CPUs, to help disseminate machine learning education and research. It's a Jupyter notebook environment that requires no setup to use and runs entirely in the cloud. This means that as long as we have a Google account, we can freely train our models on a K80 GPU. When we log in and connect to the Colaboratory runtime, we get our own virtual machine with a K80 GPU (not always entirely ours) and a Jupyter notebook environment. We could use it to our heart's content for up to 12 hours. Or until we closed our browser window. Be warned, sometimes the runtime can disconnect randomly.[13]

Colab is suggested as ideal for everything from improving our Python coding skills to working with deep learning libraries, like PyTorch, Keras, TensorFlow, and OpenCV. We can create notebooks in Colab, upload notebooks, store notebooks, share notebooks, mount our Google Drive and use whatever we've got stored in there, import most of our favorite directories, upload our personal Jupyter Notebooks, upload notebooks directly from GitHub, upload Kaggle files, download our notebooks, and do just about everything else that we might want to be able to do.[14]

Tensorflow

Two of the top numerical platforms in Python that provide the basis for Deep Learning research and development are Theano and TensorFlow. Both are free and open-source software libraries very powerful libraries, but both can be difficult to use directly for creating deep learning models. TensorFlow is for dataflow and differentiable programming across a range of tasks. It is a symbolic math library, and is also used for machine learning applications such as neural networks. [15]

The DL algorithms consist of computations and states that include both the input data that is needed to be preprocessed and the parameters with their update rules which are necessary for the mathematical calculations. TensorFlow calculates the above by using a dataflow graph. The key feature of a dataflow graph is that the autonomous computations of a NN can be performed in parallel. It is also possible to fragment computations across different devices in order to speed-up the whole procedure. This can be achieved from the dataflow graph because the links between the calculations are organized exceptionally.[29][30]

It allows developers to create large-scale neural networks with many layers. TensorFlow is mainly used for[16]:

- Classification
- Perception
- Understanding
- Discovering
- Prediction and Creation.

The typical structure of a TensorFlow graph comprises of vertices and edges. Vertices depict the processing elements of an NN which are responsible for the computations. Edges illustrate the inputs and outputs to vertices and are responsible for transferring the values. The computations performed in vertices are declared as operations while the values along edges are declared as tensors. Both the inputs and the outputs of operations are tensors. Tensors can be considered as n-dimensional arrays, and their primitive type can be ranging between float, int, and string. TensorFlow is a flexible and robust library for implementing DL models because it enables the user to choose between partial and concurrent execution. This is achieved as the API grants the permission to determine which subgraph will be carried out each time. Moreover, in many DL models, the training parameters are adjusted by using the stochastic gradient descent (SGD) method. Given a loss function, SGD calculates the gradients of it and updates the parameters in order to minimize the error rate. This procedure is optimized and automated in TensorFlow as it automatically generates the backpropagation code to reduce the loss function by using an appropriate library. Typically, the training phase of a DNN may take hours or even days. Consequently, it is crucial to be able to monitor the training process. TensorFlow supports a dashboard where users can follow the training process through graphs that illustrate useful metrics. The user can also easily see the connections within the neural network with a tool called graph visualizer. In this way, the client knows if the neural network he implemented has the correct structure (Abadi et al.2016,2015). [29][30]

TensorFlow is used for both research and production at Google.[15]

Main Use Cases of TensorFlow:

- **Voice/Sound Recognition:** One of the most well-known uses of TensorFlow are Sound based applications. With the proper data feed, neural networks are capable of understanding audio signals (Voice recognition, Voice search).
- **Text Based Applications:** Further popular uses of TensorFlow are, text based applications such as sentimental analysis (CRM, Social Media), Threat Detection (Social Media, Government) and Fraud Detection (Insurance, Finance) Language Detection is one of the most popular uses of text based applications. We all know Google Translate, which supports over 100 languages translating from one to another.
- **Image Recognition:** Mostly used by Social Media, Telecom and Handset Manufacturers; Face Recognition, Image Search, Motion Detection, Machine Vision and Photo Clustering can be used also in Automotive, Aviation and Healthcare Industries. Image Recognition aims to recognize and identify people and objects in images as well as understanding the content and context.

TensorFlow object recognition algorithms classify and identify arbitrary objects within larger images. This is usually used in engineering applications to identify shapes for modeling purposes (3D space construction from 2D images) and by social networks for photo tagging (Facebook's Deep Face). By analyzing thousands of photos of trees for example, the technology can learn to identify a tree it has never seen before.

Image Recognition is starting to expand in the Healthcare Industry, too where TensorFlow algorithms can process more information and spot more patterns than their human counterparts. Computers are now able to review scans and spot more illnesses than humans.[16]

By using Tensorflow things getting very easy, because with Tensorflow we have:

- **Easy model building:** Build and train ML models easily using intuitive high-level APIs like Keras with eager execution, which makes for immediate model iteration and easy debugging.
- **Robust ML production anywhere:** Easily train and deploy models in the cloud, on-prem, in the browser, or on-device no matter what language you use.
- **Powerful experimentation for research:** A simple and flexible architecture to take new ideas from concept to code, to state-of-the-art models, and to publication faster.[17]

Keras API

Keras is a minimalist Python library for deep learning that can run on top of Theano or TensorFlow. It supports any DNN ranging from CNNs and RNNs (Keras2018a). It was developed to make implementing deep learning models as fast and easy as possible for research and development. It runs on Python 2.7 or 3.5 and can seamlessly execute on GPUs and CPUs given the underlying frameworks. It is released under the permissive MIT license.

Keras was developed and maintained by François Chollet, a Google engineer using four guiding principles[18]:

- **Modularity:** A model can be understood as a sequence or a graph alone. All the concerns of a deep learning model are discrete components that can be combined in arbitrary ways.
- **Minimalism:** The library provides just enough to achieve an outcome, no frills and maximizing readability.
- **Extensibility:** New components are intentionally easy to add and use within the framework, intended for researchers to trial and explore new ideas.
- **Python:** No separate model files with custom file formats. Everything is native Python.

5. CNN Models Used in Thesis

Generally we used for our purpose 3 different models of Convolutional Neural Network:

- Model 1
- Model 2
- Model 3

All models are inspired from the many tutorials on CNN for image recognition and with the right adjustments and tweaks on the layers (Conv2d, Activations...) and on the hyperparameters we manage to achieve some promising results for our case study.

We are going to see some basic parts of models, from what parts and functions the layers are made and analyze each layer in the Convolutional Neural Network.

First of all, CNN processes images using matrixes of weights called filters (features) that detect specific attributes such as vertical edges, horizontal edges, etc. Moreover, as the image progresses through each layer, the filters are able to recognize more complex attributes. Ultimate goal of the CNN is to detect what is going on in the scene.[20]

Input

A Matrix of pixel values in the shape of [WIDTH, HEIGHT, CHANNELS]. Let's assume for this purpose that our input is [32x32x3]. Basically our input are images of [224x224x3].

Convolution

The ultimate purpose of this layer is to receive a feature map. Usually, we start with low number of filters for low-level feature detection. The deeper we go into the CNN, the more filters (usually they are also smaller) we use to detect high-level features.[20] Figure 4.3

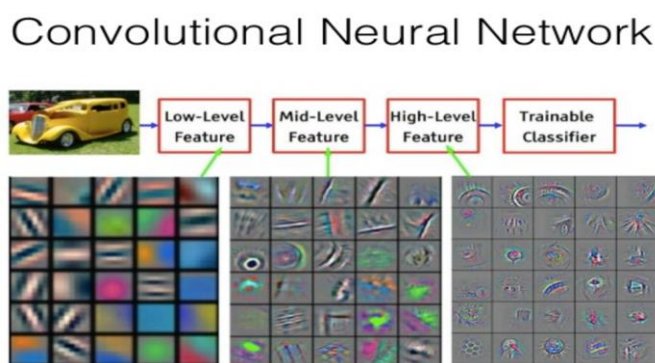


Figure 4.3 Example of feature maps of CNN.

Feature detection is based on “**scanning**” the input with the filter of a given size and applying matrix computations in order to derive a feature map. Figure 4.4

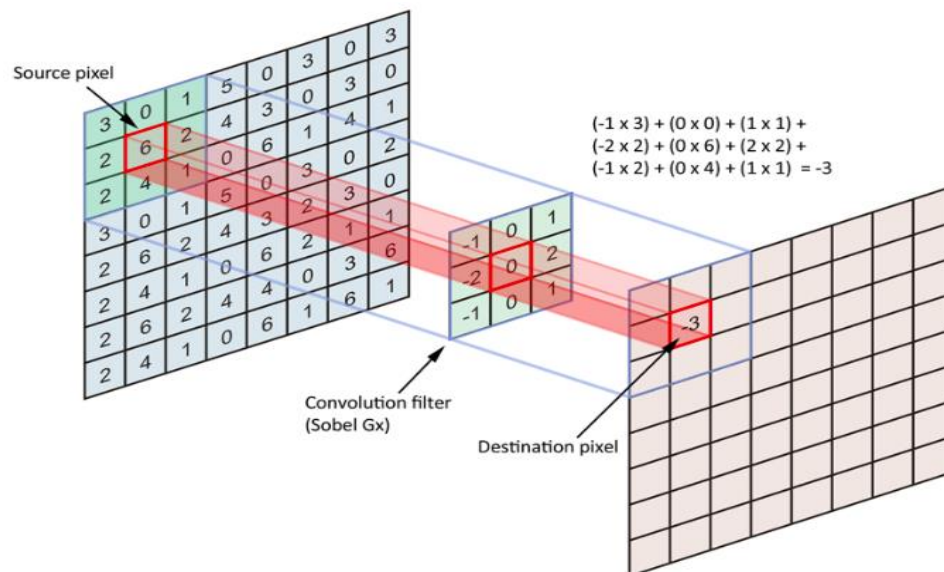


Figure 4.4 The scanning of image with 3x3 filter.

Let's take a look at the following 3x3 filter with stride 1. Figure 4.5 Assuming that we use 12 filters, we will end up with a [32x32x12] output.

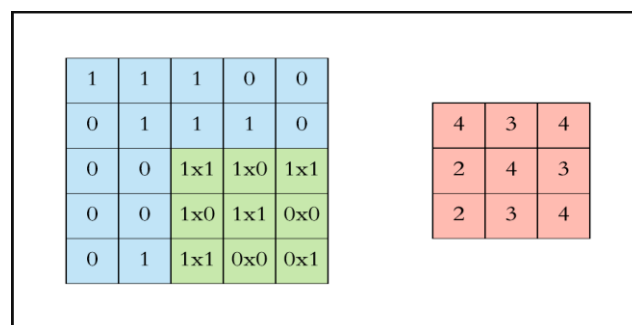


Figure 4.5 Applied 3x3 filter with stride 1.

Activation

Without going into further details, we will use ReLU activation function that returns 0 for every negative value in the input image while it returns the same value for every positive value. Figure 4.6 Shape remains unchanged, it's still [32x32x12]. [20]

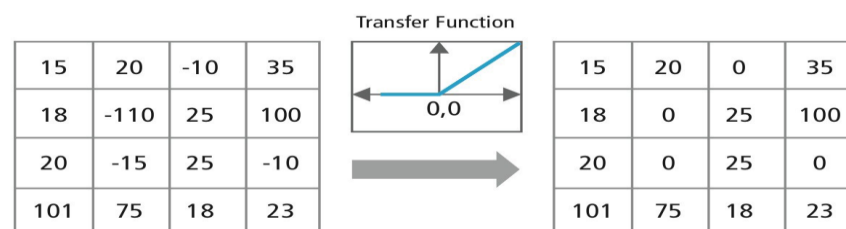


Figure 4.6 Applied Activation ReLU, we hold the positive values and negatives becomes 0.

Pooling

The goal of this layer is to provide spatial variance, which simply means that the system will be capable of recognizing an object as an object even when its appearance varies in some way.

Pooling layer will perform a downsampling operation along the spatial dimensions (width, height), resulting in output such as [16x16x12] for pooling_size=(2, 2) [20]. Figure 4.7

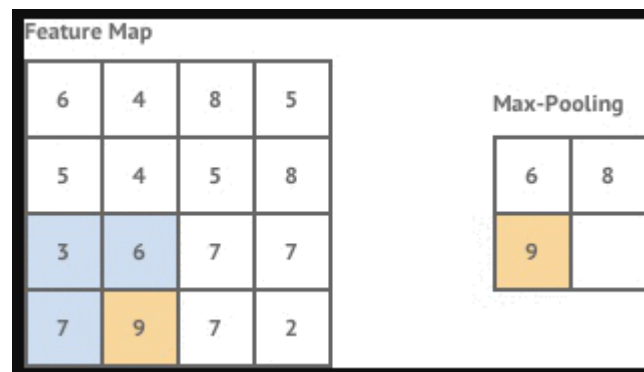


Figure 4.7 Maxpooling2d (2,2).

Fully Connected

In a fully connected layer, we flatten the output of the last convolution layer and connect every node of the current layer with the other node of the next layer. Neurons in a fully connected layer have full connections to all activations in the previous layer, as seen in regular Neural Networks and work in a similar way.

The last layer of our CNN will compute the class probability scores, resulting in a volume of size [1x1xNUMBER_OF_CLASSES]. [20] Figure 4.8

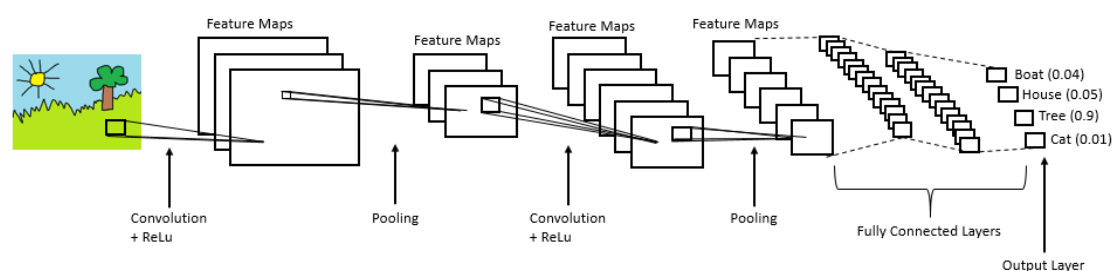


Figure 4.8 Fully connected layer (flatten).

5.1. Technics we used to improve CNN performance: Dropout, Augmentation, Transfer learning - Fine tuning

Training a deep neural network that can generalize well to new data is a challenging problem. A model with too little capacity cannot learn the problem, whereas a model with too much capacity can learn it too well and overfit the training dataset. Both cases result in a model that does not generalize well. A modern approach to reducing generalization error is to use a larger model that may be required to use regularization during training that keeps the weights of the model small.

These techniques not only reduce overfitting, but they can also lead to faster optimization of the model and better overall performance. [21]

The objective of a neural network is to have a final model that performs well both on the data that we used to train it (e.g. the training dataset) and the new data on which the model will be used to make predictions.

We require that the model learn from known examples and generalize from those known examples to new examples in the future. We use methods like a train/validation split or k-fold cross-validation only to estimate the ability of the model to generalize to new data.[21]

Too little learning and the model will perform poorly on the training dataset and on new data. The model will underfit the problem. Too much learning and the model will perform well on the training dataset and poorly on new data, the model will overfit the problem. In both cases, the model has not generalized.

- **Underfit Model.** A model that fails to sufficiently learn the problem and performs poorly on a training dataset and does not perform well on a holdout sample.
- **Overfit Model.** A model that learns the training dataset too well, performing well on the training dataset but does not perform well on a hold out sample.
- **Good Fit Model.** A model that suitably learns the training dataset and generalizes well to the old out dataset.

We can address underfitting by increasing the **capacity of the model**. Capacity refers to the ability of a model to fit a variety of functions; more capacity, means that a model can fit more types of functions for mapping inputs to outputs. Increasing the capacity of a model is easily achieved by changing the structure of the model, such as adding more layers and/or more nodes to layers. Because an underfit model is so easily addressed, it is more common to have an overfit model.

An overfit model is easily diagnosed by monitoring the performance of the model during training by evaluating it on both a training dataset and on a holdout validation dataset. Graphing line plots of the performance of the model during training, called learning curves, will show a familiar pattern. For example, line plots of the loss (that we seek to minimize) of the model on train and validation datasets will show a line for the training dataset that drops and may plateau and a line for the validation dataset that drops at first, then at some point begins to rise again. [21] Figure 4.9

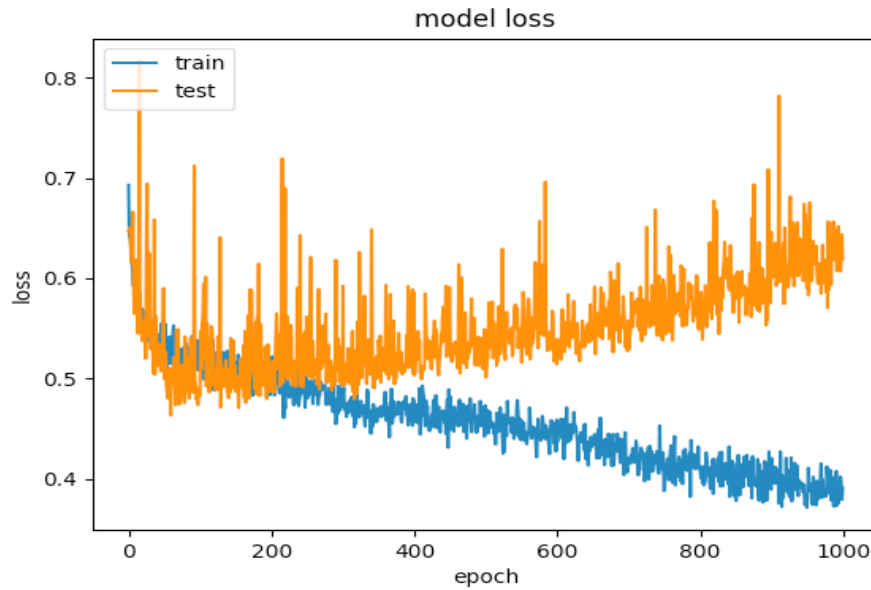


Figure 4.9 Example of overfit model diagnosed by monitoring the performance of the model during training [22]

Dropout

As we said deep neural nets with a large number of parameters are very powerful machine learning systems. However, overfitting is a serious problem in such networks. Large networks are also slow to use, making it difficult to deal with overfitting by combining the predictions of many different large neural nets at test time. Dropout is a technique for addressing this problem.

The key idea is to randomly drop units (along with their connections) from the neural network during training. This prevents units from co-adapting too much. During training, dropout samples from an exponential number of different “thinned” networks. At test time, it is easy to approximate the effect of averaging the predictions of all these thinned networks by simply using a single unthinned network that has smaller weights. This significantly reduces overfitting and gives major improvements over other regularization methods. [23]

The term “**dropout**” refers to dropping out units (hidden and visible) in a neural network. By dropping a unit out, we mean temporarily removing it from the network, along with all its incoming and outgoing connections. The choice of which units to drop is random. In the simplest case, each unit is retained with a fixed probability p independent of other units, where p can be chosen using a validation set or can simply be set at 0.5, which seems to be close to optimal for a wide range of networks and tasks.[23] Figure 4.10

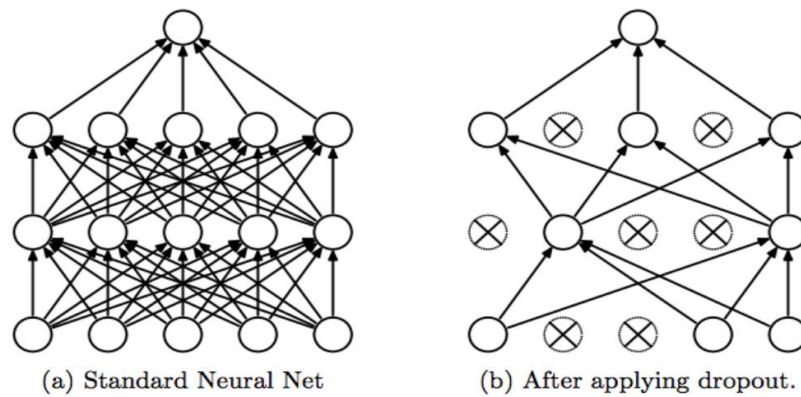


Figure 4.10 Dropout Neural Net Model. Left: A standard neural net with 2 hidden layers. Right: An example of a thinned net produced by applying dropout to the network on the left. Crossed units have been dropped.[23]

Augmentation

Image data augmentation is a technique that can be used to artificially expand the size of a training dataset by creating modified versions of images in the dataset. Training deep learning neural network models on more data can result in more skillful models, and the augmentation techniques can create variations of the images that can improve the ability of the fit models to generalize what they have learned to new images. The Keras deep learning neural network library provides the capability to fit models using image data augmentation via the **ImageDataGenerator** class. The performance of deep learning neural networks often improves with the amount of available data. Data augmentation is a technique to artificially create new training data from existing training data. This is done by applying domain-specific techniques to examples from the training data that create new and different training examples. Image data augmentation is perhaps the most well-known type of data augmentation and involves creating transformed versions of images in the training dataset that belong to the same class as the original image. Transforms include a range of operations from the field of image manipulation, such as shifts, flips, zooms, and much more. The intent is to expand the training dataset with new, plausible examples. This means, variations of the training set images that are likely to be seen by the model. Image data augmentation is typically only applied to the training dataset, and not to the validation or test dataset. This is different from data preparation such as image resizing and pixel scaling, which they must be performed consistently across all datasets that interact with the model.[24]

Next will see some of the techniques that we used with examples. We take two images from our dataset one healthy and one bleeding. Figure 4.11

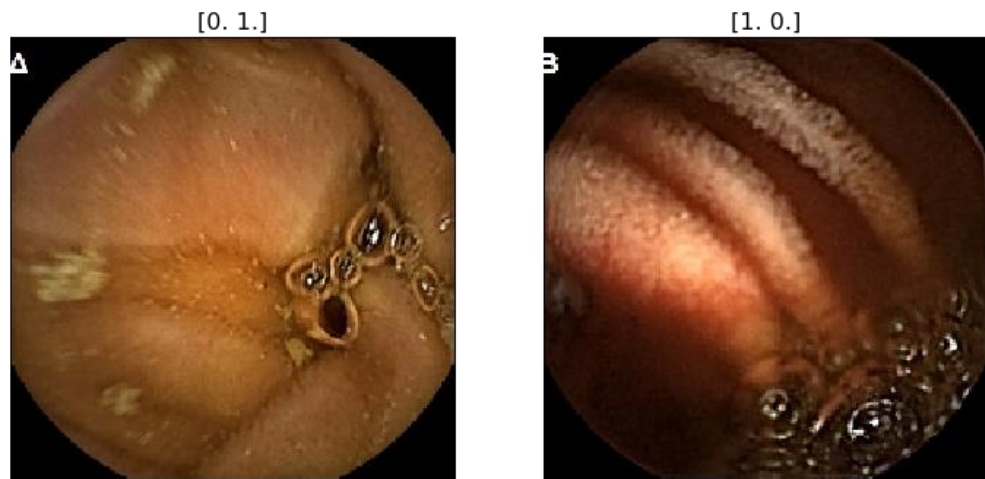


Figure 4.11 Two images with their labels, left a healthy image with label [0.1] and right a bleeding one with label [1.0]

Rotation

By specifying the `rotation_range`, the data generated is randomly rotated by an angle in the range of `+rotation_range` to `-rotation_range` (in degrees). Figure 4.12

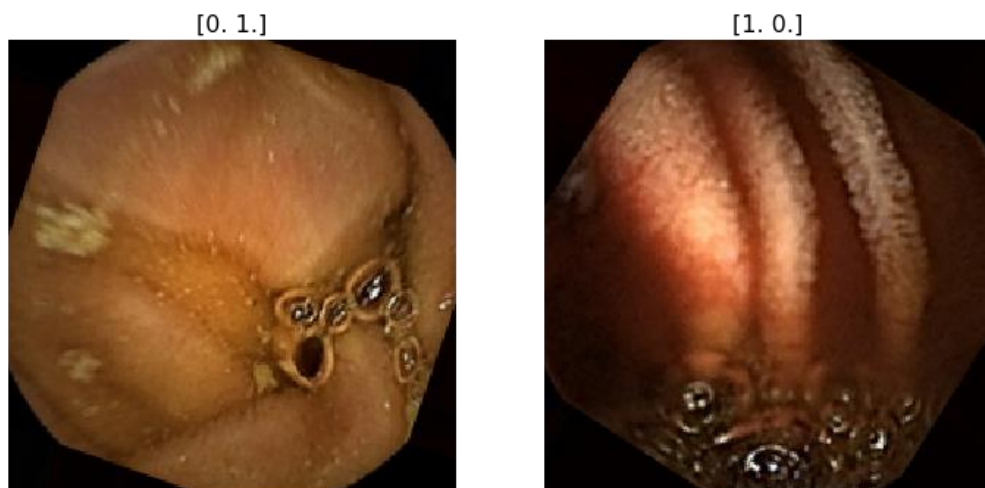


Figure 4.12 Rotated images generated from Image data augmentation technique.

Width Shifting

The `width_shift_range` is a floating point number between 0.0 and 1.0 which specifies the upper bound of the fraction of the total width by which the image is to be randomly shifted, either towards the left or right. Figure 4.13

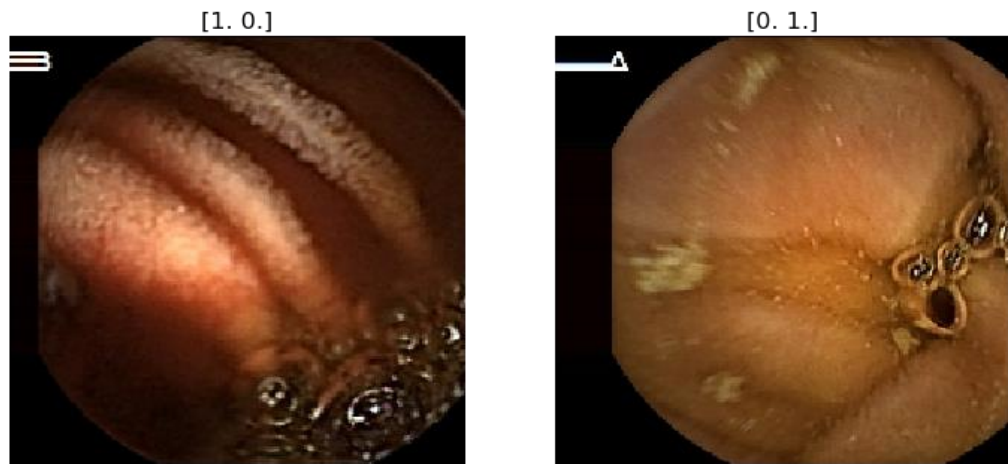


Figure 4.13 Example of Images generated with Width Shift technique.

Height Shifting

Exactly like width shifting, except that the image is shifted vertically instead of horizontally. Figure 4.14



Figure 4.14 Example of Images generated with Height Shift.

Shear Intensity

Shear transformation slants the shape of the image. This is different from rotation in the sense that in shear transformation, we fix one axis and stretch the image at a certain angle known as the shear angle. This creates a sort of 'stretch' in the image, which is not seen in rotation. `shear_range` specifies the angle of the slant in degrees. Figure 4.15. As we said before, the intent is to expand the training dataset with new, plausible examples. This means, variations of the training set images that are likely to be seen by the model, we don't expect from endoscopic capsule to take images like in Figure 4.15 so the shear intensity is more smaller than the example.

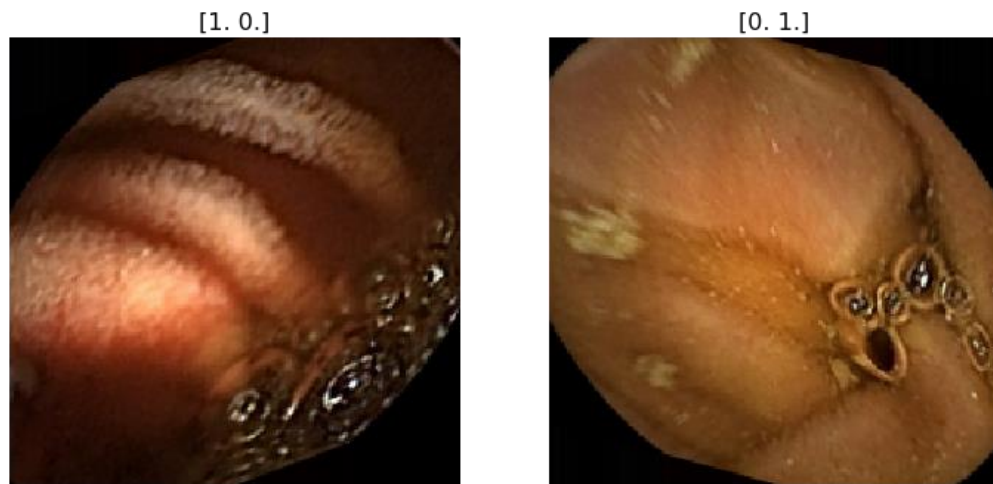


Figure 4.15 Example of Images generated with Shear Intensity with larger value.

Zoom

A random zoom is obtained by the zoom_range argument. A zoom less than 1.0 magnifies the image, while a zoom greater than 1.0 zooms out of the image. Figure 4.16

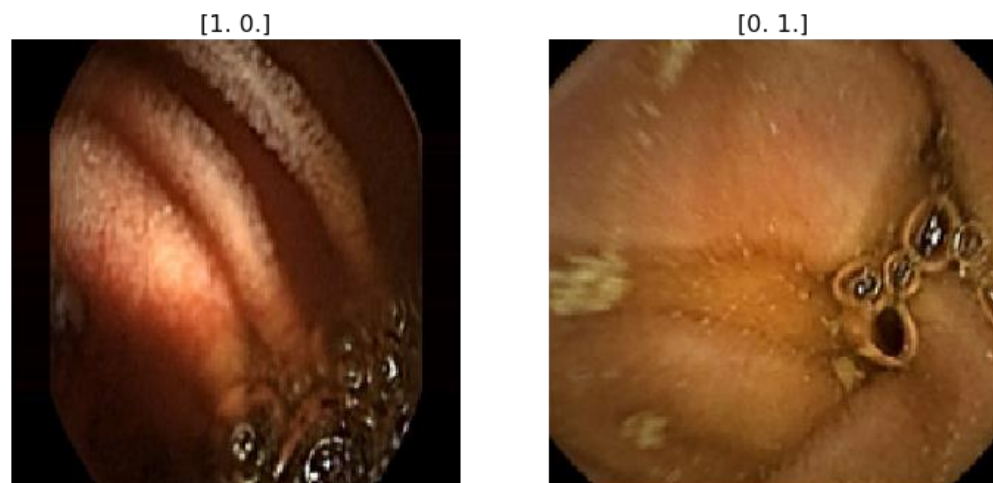


Figure 4.16 Example of Images generated with Zoom_range = 0.2

Horizontal Flip

The generator will generate images, which on a random basis, will be horizontally flipped. Figure 4.17

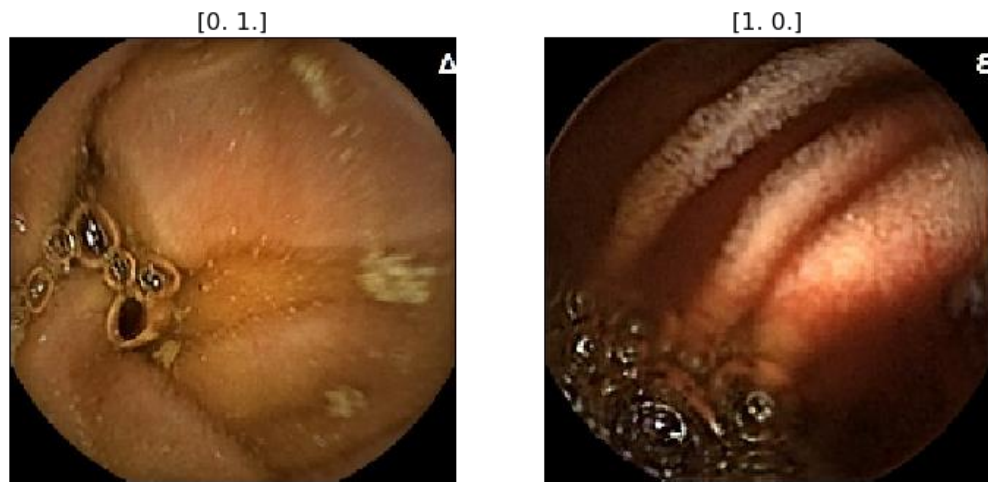


Figure 4.17 Example of Images generated with Images with Horizontal Flip.

There are and some others augmentation techniques like Vertical Flip, Reflect, Wrap but in our case they wasn't necessary and after all when we tested them they didn't work very well, so we used these who makes the difference.

Background Knowledge: Transfer Learning - Fine Tuning (VGGNet, ResNet V2)

We tried to improve the performance of our models with “**Transfer Learning**”, which is a popular method for cases that the dataset is small. With transfer learning, instead of starting the learning process from scratch, we start from patterns that have been learned when solving a different problem. For that purpose we use pre-trained models that was trained on a large benchmark dataset to solve a problem similar to the one that we want to solve. There are some pre-trained models from published literature and Keras comes prepackaged with many of these. Famous CNN architectures are VGG and ResNet, are the models that we tried in the present Thesis. [31]

In the table below four of the well known CNNs are sorted their top-5 accuracy on Imagenet dataset. The number of trainable parameters and the Floating Point Operations (FLOP) required for a forward pass can also be seen.

Comparison					
Network	Year	Salient Feature	top5 accuracy	Parameters	FLOP
AlexNet	2012	Deeper	84.70%	62M	1.5B
VGGNet	2014	Fixed-size kernels	92.30%	138M	19.6B
Inception	2014	Wider - Parallel kernels	93.30%	6.4M	2B
ResNet-152	2015	Shortcut connections	95.51%	60.3M	11B

- AlexNet and ResNet-152, both have about 60M parameters but there is about 10% difference in their top-5 accuracy. But training a ResNet-152 requires a lot of computations (about 10 times more than that of AlexNet) which means more training time and energy required.

- VGGNet not only has a higher number of parameters and FLOP as compared to ResNet-152, but also has a decreased accuracy. It takes more time to train a VGGNet with a reduced accuracy.
- Training an AlexNet takes about the same time as training Inception. The memory requirements is 10 times less with an improved accuracy (about 9%)

Background Knowledge: VGGNet was born out of the need to reduce the # of parameters in the CONV layers and improve on training time. There are multiple variants of VGGNet (VGG16, VGG19 etc.) which differ only in the total number of layers in the network. The structural details of a VGG16 network has been shown below.

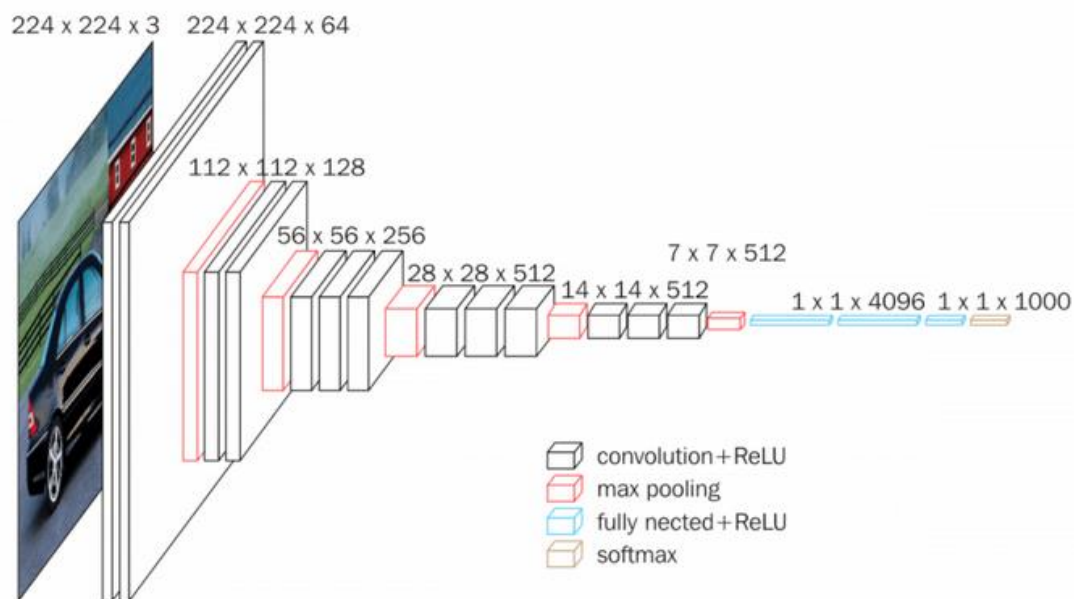


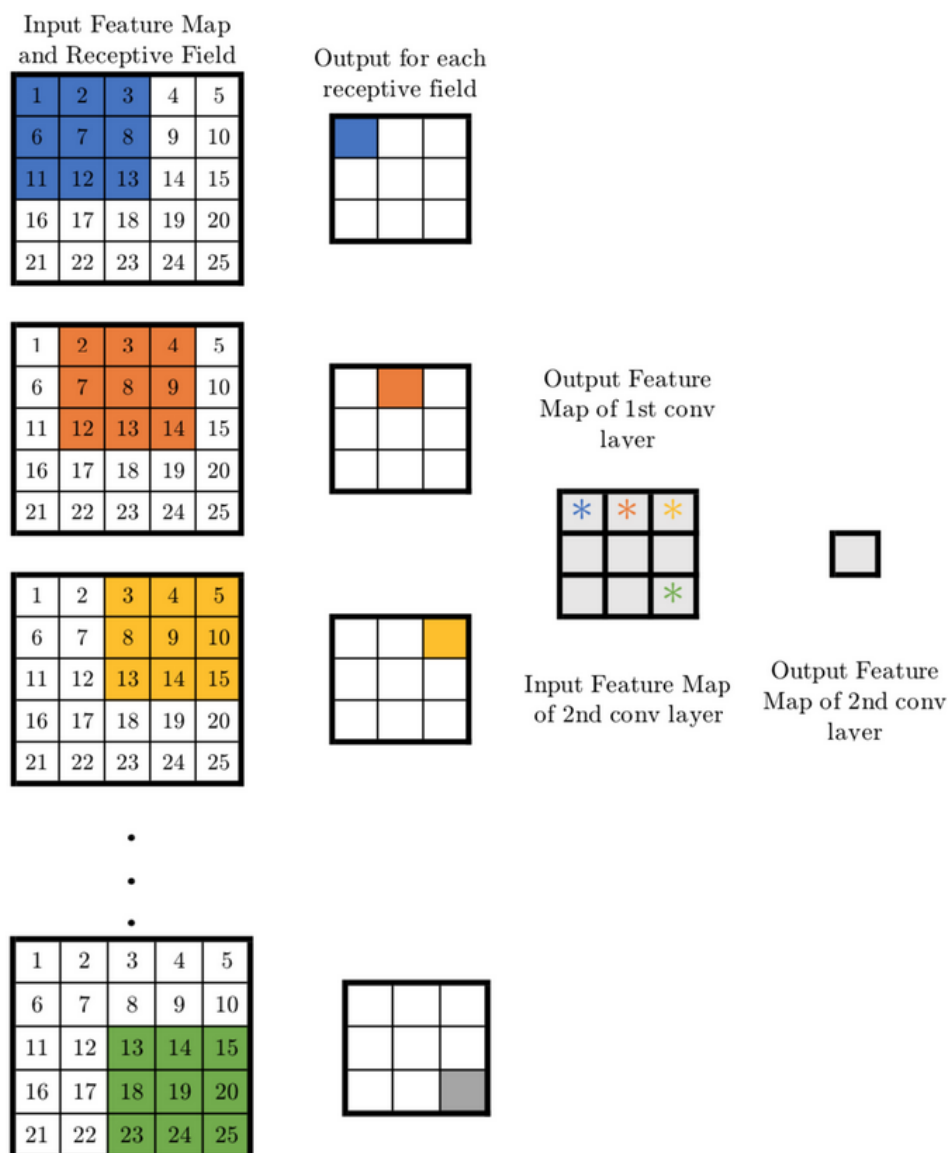
Fig VGG16 Block Diagram (source: neurohive.io)

Below we present the structural details of VGG16.

VGG16 - Structural Details													
#	Input Image			output			Layer	Stride	Kernel		in	out	Param
1	224	224	3	224	224	64	conv3-64	1	3	3	3	64	1792
2	224	224	64	224	224	64	conv3064	1	3	3	64	64	36928
	224	224	64	112	112	64	maxpool	2	2	2	64	64	0
3	112	112	64	112	112	128	conv3-128	1	3	3	64	128	73856
4	112	112	128	112	112	128	conv3-128	1	3	3	128	128	147584
	112	112	128	56	56	128	maxpool	2	2	2	128	128	65664
5	56	56	128	56	56	256	conv3-256	1	3	3	128	256	295168
6	56	56	256	56	56	256	conv3-256	1	3	3	256	256	590080
7	56	56	256	56	56	256	conv3-256	1	3	3	256	256	590080
	56	56	256	28	28	256	maxpool	2	2	2	256	256	0
8	28	28	256	28	28	512	conv3-512	1	3	3	256	512	1180160
9	28	28	512	28	28	512	conv3-512	1	3	3	512	512	2359808
10	28	28	512	28	28	512	conv3-512	1	3	3	512	512	2359808
	28	28	512	14	14	512	maxpool	2	2	2	512	512	0
11	14	14	512	14	14	512	conv3-512	1	3	3	512	512	2359808
12	14	14	512	14	14	512	conv3-512	1	3	3	512	512	2359808
13	14	14	512	14	14	512	conv3-512	1	3	3	512	512	2359808
	14	14	512	7	7	512	maxpool	2	2	2	512	512	0
14	1	1	25088	1	1	4096	fc		1	1	25088	4096	102764544
15	1	1	4096	1	1	4096	fc		1	1	4096	4096	16781312
16	1	1	4096	1	1	1000	fc		1	1	4096	1000	4097000
Total													138,423,208

VGG16 has a total of 138 million parameters. The important point to note here is that all the conv kernels are of size 3x3 and maxpool kernels are of size 2x2 with stride of two.

The idea behind having fixed size kernels is that all the variable size convolutional kernels used in the famous CNN model Alexnet (11x11, 5x5, 3x3) can be replicated by making use of multiple 3x3 kernels as building blocks. The replication is in terms of the receptive field covered by the kernels. Let's consider the following example. Say we have an input layer of size 5x5x1. Implementing a conv layer with kernel size of 5x5 and stride one will result and output feature map of 1x1. The same output feature map can be obtained by implementing two 3x3 conv layers with stride of 1 as shown below



Now let's look at the number of variables needed to be trained. For a 5x5 conv layer filter the number of variables is 25. On the other hand, two conv layers of kernel size 3x3 have a total of $3 \times 3 \times 2 = 18$ variables (a reduction of 28%).

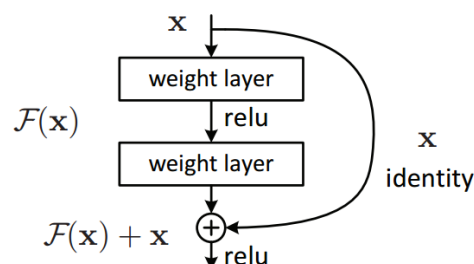
Similarly, the effect of one 7x7 (11x11) conv layer can be achieved by implementing three (five) 3x3 conv layer with stride of one. This reduces the number of trainable variables by 44.9% (62.8%). Reduced number of trainable variables means faster learning and more robust to over-fitting.[31]

Background Knowledge: ResNetXX model architecture (where 'XX' denotes the number of layers)

Most commonly used ones are ResNet50 and ResNet101. Below we present the structural details of ResNet18.

ResNet18 - Structural Details														
#	Input Image			output			Layer	Stride	Pad	Kernel		in	out	Param
1	227	227	3	112	112	64	conv1	2	1	7	7	3	64	9472
	112	112	64	56	56	64	maxpool	2	0.5	3	3	64	64	0
2	56	56	64	56	56	64	conv2-1	1	1	3	3	64	64	36928
3	56	56	64	56	56	64	conv2-2	1	1	3	3	64	64	36928
4	56	56	64	56	56	64	conv2-3	1	1	3	3	64	64	36928
5	56	56	64	56	56	64	conv2-4	1	1	3	3	64	64	36928
6	56	56	64	28	28	128	conv3-1	2	0.5	3	3	64	128	73856
7	28	28	128	28	28	128	conv3-2	1	1	3	3	128	128	147584
8	28	28	128	28	28	128	conv3-3	1	1	3	3	128	128	147584
9	28	28	128	28	28	128	conv3-4	1	1	3	3	128	128	147584
10	28	28	128	14	14	256	conv4-1	2	0.5	3	3	128	256	295168
11	14	14	256	14	14	256	conv4-2	1	1	3	3	256	256	590080
12	14	14	256	14	14	256	conv4-3	1	1	3	3	256	256	590080
13	14	14	256	14	14	256	conv4-4	1	1	3	3	256	256	590080
14	14	14	256	7	7	512	conv5-1	2	0.5	3	3	256	512	1180160
15	7	7	512	7	7	512	conv5-2	1	1	3	3	512	512	2359808
16	7	7	512	7	7	512	conv5-3	1	1	3	3	512	512	2359808
17	7	7	512	7	7	512	conv5-4	1	1	3	3	512	512	2359808
	7	7	512	1	1	512	avg pool	7	0	7	7	512	512	0
18	1	1	512	1	1	1000	fc					512	1000	513000
Total														11,511,784

Since the vanishing gradient problem was taken care (more about it in the How part) of, the CNN started to get deeper and deeper. Resnet18 has around 11 million trainable parameters. It consists of CONV layers with filters of size 3x3 (just like VGGNet). Only two pooling layers are used throughout the network one at the beginning and other at the end of the network. Identity connections are between every two CONV layers. The solid arrows show identity shortcuts where the dimension of the input and output is same, while the dotted ones present the projection connections where the dimensions differ. ResNet architecture makes use of shortcut connections do solve the vanishing gradient problem. The basic building block of ResNet is a Residual block which is repeated throughout the network.



Residual Block — Image taken from original paper.

Instead of learning the mapping from $x \rightarrow F(x)$, the network learns the mapping from $x \rightarrow F(x)+G(x)$. When the dimension of the input x and output $F(x)$ is same, the function $G(x) = x$ is an identity function and the shortcut connection is called Identity connection. The identical mapping is learned by zeroing out the weights in the intermediate layer during training, since it's easier to zero out the weights than push them to one.[31]

For the case when the dimensions of $F(x)$ differ from x (due to stride length >1 in the CONV layers in between), Projection connection is implemented rather than Identity connection. The function $G(x)$ changes the dimensions of input x to that of output $F(x)$. Two kinds of mapping were considered in the original paper.

- **Non-trainable Mapping** (Padding): The input x is simply padded with zeros to make the dimension match to that of $F(x)$
- **Trainable Mapping** (Conv Layer): 1×1 Conv layer is used to map x to $G(x)$. It can be seen from the table above that across the network the spatial dimensions are either kept the same or halved, and the depth is either kept the same or doubled and the product of Width and Depth after each conv layer remains same i.e. 3584. 1×1 conv layers are used to half the spatial dimension and double the depth by using stride length of 2 and multiple of such filters respectively. The number of 1×1 conv layers is equal to the depth of $F(x)$.

Our Methodology for transfer Learning

For us now that we use CNN there are some strategies for transfer learning, first we must show the parts of CNN that we are interested in. A typical CNN has two parts, the Convolutional base and the Classifier. Figure 4.18

Convolutional base, is composed of a stack of convolutional and pooling layers. The main goal of the convolutional base is to generate features from the image.

Classifier, is usually composed of fully connected layers. The main goal of the classifier is to classify the image based on the detected features. A fully connected layer is a layer whose neurons have full connections to all activation in the previous layer.



Figure 4.18 The basic architecture of CNN with the two parts Convolutional base and Classifier.[25]

There are three main strategies for the transfer learning-fine tuning Figure 4.19 :

- **Train the entire model.** In this case, you use the architecture of the pre-trained model and train it according to your dataset. You're learning the model from scratch, so you'll need a large dataset (and a lot of computational power).[25]
- **Train some layers and leave the others frozen.** As you remember, lower layers refer to general features (problem independent), while higher layers refer to specific features (problem dependent). Here, we play with that dichotomy by choosing how much we want to adjust the weights of the network (a frozen layer does not change during training). The layers that we keep frozen depends of the dataset we have and how similar the problem is with that of the pre-trained model.[25]
- **Freeze the convolutional base.** Here we freeze the entire convolutional base and we keep all the weights as are. So we only train the classifier.[25]

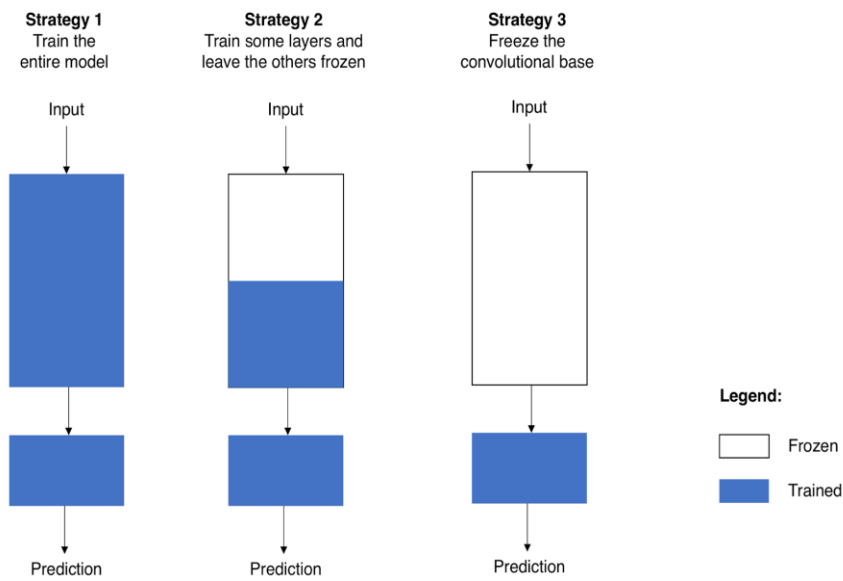


Figure 4.19 The three strategies of transfer learning- fine tuning.[25]

Also in the second strategy we can choose how many layers we can keep frozen and that depends on our problem, for example if we have large or small dataset and how similar our dataset is to that of the pre-trained model the number of layers that we keep frozen are different.

We must say that the transfer learning in the end didn't work very well for our problem and the factors are basically two, the first and the most important is that the dataset is not similar to the dataset of the pre-trained model. All the pre-trained models we used are trained on ImageNet dataset that has no similarity to our dataset, in Figure 4.20 we can see some categories of the ImageNet.[26]

Statistics of high level categories

High level category	# synset (subcategories)	Avg # images per synset	Total # images
amphibian	94	591	56K
animal	3822	732	2799K
appliance	51	1164	59K
bird	856	949	812K
covering	946	819	774K
device	2385	675	1610K
fabric	262	690	181K
fish	566	494	280K
flower	462	735	339K
food	1495	670	1001K
fruit	309	607	188K
fungus	303	453	137K
furniture	187	1043	195K

Figure 4.20 Some categories of ImageNet

Because the pre-trained models are well trained on many categories and many images of each category and if we see our images the difference between healthy and bleeding images is the values of color red, that lead our models to be overtrained and predict the same for all the images.

5.2. Our Implementation of the CNN Models

As we said, we used three different models for our problems

- **Model 1**
- **Model 2**
- **Model 3**

In these models except for the basic structure with the convolutional, activation and pooling layers, we used the techniques we said before, like dropout and the results of each model are represented in tables in the next chapter for comparing and seeing which of these models is better. So will see our models one at each time and the different techniques we applied on them.

CNN: Model 1

First, we will see Model 1, in this model we also used different blocks of convolutional layers and see what results they give us each time. So first we applied one block, next two and finally three and then we applied augmentation, dropout and transfer learning.

One Block Model 1. In Figure 4.21 we can see the architecture of the model.

```
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(224, 224, 3)),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(128, activation='relu'),
    Dense(2, activation='sigmoid'),
])
```

Figure 4.21 Model 1 with one convolutional block.

First we create a **Sequential** model. The Sequential model API is a way of creating deep learning models where an instance of the Sequential class is created and model layers are created and added to it. This tells keras to stack all layers sequentially.

As we said in this model we have one block, so the first layer is a **Conv2D** layer and there is also the input layer which has some important parameters. We have filter size (32) this is the size of the output dimension, the number of output filters in the convolution and we specify the height and width of the 2D convolutional window to [3,3]. Next we select the activation function and we use the **ReLU**, which is the most common activation function, we saw previously what this function does. Finally, one more important thing to do is to specify the input shape, we pass the shape of our images 224 by 224 and the number of channels 3 because we have colored RGB images.

After that we add a **MaxPool2D (2, 2)** layer whose function is to reduce the spatial size of the incoming features and therefore helps reduce the number of parameters and computation in the network, thereby helping to reduce overfitting.[5]

In the end, we have the classifier where we add a **Flatten layer**. A Conv2D layer extracts and learns spatial features which is then passed to a dense layer after it has been flattened. The last layer has output of 2 and a Sigmoid activation function, this is because we have two states healthy and bleeding, so we want the model to output a probability between 0 and 1. The closest to 0 if the image is healthy and to 1 if the image has a bleed.

Two block Mode 1. Figure 4.22


```

model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(224, 224, 3)),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(128, activation='relu'),
    Dense(2, activation='sigmoid'),
])

```

Figure 4.22 Two block model.

Here the model is the same, we only add one more convolutional layer with filter size (64) and we specify the convolutional window to [3,3]. Also after that we add one more Max Pool layer and we set it to (2,2).

Three block Model 1 in this model we did the same as previous we add a convolutional layer and a Max Pool layer, the only difference is the filter size which we set to (128). Figure 4.23

```

model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(224, 224, 3)),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(128, activation='relu'),
    Dense(2, activation='sigmoid'),
])

```

Figure 4.23 Three block model.

CNN: Model 2

In this model we don't separate the blocks of convolutional layers to see how the model performs because we did it in the previous model (Model 1). Let's see in Figure 4.24 the structure of our model.

```

model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(224, 224, 3)),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(512, activation='relu'),
    Dense(2, activation='sigmoid'),
])

```

Figure 4.24 The structure of the model Model 2.

Again we create a **Sequential** model in order to stack all layers sequentially. In the first layer, the input layer we use again a Conv2D layer with filter size (32) and the width, height of 2D convolutional window at [3,3]. Next we call the ReLU activation function, the input shape remains 224 by 224 with 3 channels. After that we add a MaxPool2D (2,2) layer. This was the first block, we have three more blocks of Conv2D, and MaxPool2D (2,2), in all blocks we use the ReLU activation and the same width and height of convolutional window. The difference is in the filter size because in the second block we use size filter of (64) and in the next two blocks we use size filter (128).

In the end, we have the classifier, where we add a Flatten layer. A Conv2D layers extracts and learns spatial features which is then passed to a dense layer after it has been flattened. The last layer has output of 2 and Sigmoid activation function, this is because we have two states healthy and bleeding, so we want the model to output a probability between 0 and 1. The closest to 0 if the image is healthy and to 1 if the image has a bleed.

CNN: Model 3

This is the final model that we used for our problem. In this case we had a slightly different approach on the basic model. In Figure 4.25 we can see the structure of the model.

```

model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(224, 224, 3)),
    Conv2D(32, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Conv2D(128, (3, 3), activation='relu'),
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Conv2D(256, (3, 3), activation='relu'),
    Conv2D(256, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Flatten(input_shape=(6, 6, 256)),
    Dense(256, activation='relu'),
    Dense(256, activation='relu'),
    Dense(2, activation='sigmoid'),
])

```

Figure 4.25 The structure of the model Model 3.

As we can see we begin with the creation of **Sequential** model in order to stack all layers sequentially. The input shape of our images is still the same 224 by 224 and 3 channels. For the activation we use the ReLU function except in the last layer of the classifier that we use the sigmoid function. Also here we have 4 blocks each one has two Conv2D layers, the ReLU function for activation and a MaxPool2D of (2, 2), the width and height of the convolutional window is [3, 3], the only change is the the size of Conv2D filters. In the first block we have size of (32), in second block we have (64), in block three we have size of (128) and in last one we have (256).

Implementation of Our Methods versions applying Augmentation, Dropout and Transfer Learning - Fine Tuning

Now it is time to apply the methods like Augmentation, Dropout and Transfer Learning - Fine Tuning in order to increase the performance of models and see if we really get better results than the basic models.

Dropout

As we mentioned earlier the term “**dropout**” refers to dropping out units (hidden and visible) in a neural network. By dropping a unit out, we mean temporarily removing it from the network, along with all its incoming and outgoing connections. The choice of which units to drop is random. In our case we use dropout “0.2” and “0.5” that means that we drop out randomly the 20% and 50% of the units along with their connections.

Model 1

For model 1 in figure 4.26

```

model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(224, 224, 3)),
    MaxPooling2D((2, 2)),
    Dropout(0.2),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Dropout(0.2),
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Dropout(0.2),
    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(2, activation='sigmoid'),
])

```

Figure 4.26 Our model with added with the Dropout layers.

As we can see here after the Max Pool layers we add a Dropout layer of (0.2) and in the Classifier we add a Dropout layer of (0.5).

Model 2

For model 2 we use two different approaches. First we applied dropout only in the classifier and secondly in both parts of CNN, in the convolutional base and in the classifier.

In figure 4.27 we can see the model with the dropout used in the classifier.

```

model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(224, 224, 3)),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dropout(0.5),
    Dense(512, activation='relu'),
    Dense(2, activation='sigmoid'),
])

```

Figure 4.27 Our model with added the Dropout layer only in classifier.

In figure 4.28 the model 2 with dropout layers in convolutional base and in classifier.

```

▶ model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(224, 224, 3)),
    MaxPooling2D((2, 2)),
    Dropout(0.2),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Dropout(0.2),
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Dropout(0.2),
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Dropout(0.2),
    Flatten(),
    Dropout(0.5),
    Dense(512, activation='relu'),
    Dense(2, activation='sigmoid'),
])

```

Figure 4.28 Our model with added the Dropout layers in convolutional base and in classifier.

After each block of Conv2D and Maxpooling2D we use a dropout layer of “0.2”.

Model 3

And for the third model we use the same technique as we used in the second model. We first insert a dropout layer in classifier and then in this case we use dropout layers and in the convolution base.

In figure 4.29 we can see the model 3 with the dropout layer added in the classifier, in this case we dropout “0.5”.

```

▶ model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(224, 224, 3)),
    Conv2D(32, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Conv2D(128, (3, 3), activation='relu'),
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Conv2D(256, (3, 3), activation='relu'),
    Conv2D(256, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Flatten(input_shape=(6,6,256)),
    Dense(256, activation='relu'),
    Dropout(0.5),
    Dense(256, activation='relu'),
    Dropout(0.5),
    Dense(2, activation='sigmoid'),
])

```

Figure 4.29 Our model with added the Dropout layer in classifier.

In figure 4.30 we can see the model 3 with the dropout layer added and in the convolutional base. Again here the dropout layers in the convolutional base are after each block of two Conv2D and one Maxpooling2D layers and we drop out “0.5”

```
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(224, 224, 3)),
    Conv2D(32, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Dropout(0.2),
    Conv2D(64, (3, 3), activation='relu'),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Dropout(0.2),
    Conv2D(128, (3, 3), activation='relu'),
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Dropout(0.2),
    Conv2D(256, (3, 3), activation='relu'),
    Conv2D(256, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Dropout(0.2),
    Flatten(input_shape=(6,6,256)),
    Dense(256, activation='relu'),
    Dropout(0.5),
    Dense(256, activation='relu'),
    Dropout(0.5),
    Dense(2, activation='sigmoid'),
])
```

Figure 4.30 Our model with added the Dropout layers in the entire model.

Augmentation

One more improvement we want to do in our models is to increase the performance and most importantly to smooth them to avoid the overfitting since we use small dataset, this is why we add the **Augmentation** method with the help of the **ImageDataGenerator** in our data that we feed in the model. The changes that we use are:

- the rotation of the images
- the shift of the width and height
- the change of shear
- zoom in and out of the images
- and the horizontal flip of the images

In Figure 4.31 we can see the changes we made. We must mention that we used the same Augmentation in all our models. The Augmentation applied in our models after the Dropout method.

```

▶ datagen = ImageDataGenerator(rescale=1.0/255.0,
                               rotation_range=40,
                               width_shift_range=0.2,
                               height_shift_range=0.2,
                               shear_range=0.2,
                               zoom_range=0.2,
                               horizontal_flip=True,)

```

Figure 4.31 The Augmentation changes in our model.

Transfer Learning

The final method we are going to apply is Transfer Learning and Fine Tune our model. We tried to apply three different models that were available in Keras API but all three of them have the same results so we use only one, the VGG16. Also, the other two are Inception ResNet V2 and VGG19. We used two different types of Transfer Learning but first of all we have to remove the classifier of the model and after that was to freeze all the convolutional base and add the classifier of our models. The second was to freeze some of the convolutional base and let the other available to train together with the classifier, in our case we unfreeze the “block4” and “block5” in the VGG16. We followed the same process for Transfer Learning in every model we will see. In Figure 4.32 we can see the classifier that we remove and in Figure 4.33 we can see the structure of the VGG16 and how many blocks it has without the classifier.

flatten (Flatten)	(None, 25088)	0
fc1 (Dense)	(None, 4096)	102764544
fc2 (Dense)	(None, 4096)	16781312
predictions (Dense)	(None, 1000)	4097000
=====		

Figure 4.32 The Classifier that we remove, he has a Flatten layer with to Dense layers.

input_2 (InputLayer)	(None, 224, 224, 3)	0
block1_conv1 (Conv2D)	(None, 224, 224, 64)	1792
block1_conv2 (Conv2D)	(None, 224, 224, 64)	36928
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0
block2_conv1 (Conv2D)	(None, 112, 112, 128)	73856
block2_conv2 (Conv2D)	(None, 112, 112, 128)	147584
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0
block3_conv1 (Conv2D)	(None, 56, 56, 256)	295168
block3_conv2 (Conv2D)	(None, 56, 56, 256)	590080
block3_conv3 (Conv2D)	(None, 56, 56, 256)	590080
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0
block4_conv1 (Conv2D)	(None, 28, 28, 512)	1180160
block4_conv2 (Conv2D)	(None, 28, 28, 512)	2359808
block4_conv3 (Conv2D)	(None, 28, 28, 512)	2359808
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0
block5_conv1 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv2 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv3 (Conv2D)	(None, 14, 14, 512)	2359808
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0
=====		

Figure 4.33 The structure of VGG16 without the classifier.

Model 1

Below in Figure 4.34 we can see the the added classifier from model 1 to VGG16 model.

```

▶ new_model.add(Flatten())
  new_model.add(Dropout(0.5))
  new_model.add(Dense(128, activation='relu'))
  new_model.add(Dense(2, activation='sigmoid'))

```

Figure 4.34 The classifier of model 1 that we add to VGG16.

As we said in first case we only train the classifier and leave the rest freeze with their own weights and in the second case we train along with the classifier and the block 4 and 5 of the convolutional base.

Model 2

Again we used the same process as previously. First we only change the classifier and in the second we unfreeze the blocks 4 and 5 in order to train them with our dataset. In Figure 4.35 is the classifier we used.

```
new_model.add(Flatten())
new_model.add(Dropout(0.5))
new_model.add(Dense(512, activation='relu'))
new_model.add(Dense(2, activation='sigmoid'))
```

Figure 4.35 The classifier of model 2 that we used in the VGG16.

Model 3

Finally and in third model we apply transfer learning same as before in figure 4.36 is the classifier of our model.

```
new_model.add(Flatten())
new_model.add(Dense(256, activation='relu'))
new_model.add(Dropout(0.5))
new_model.add(Dense(256, activation='relu'))
new_model.add(Dropout(0.5))
new_model.add(Dense(2, activation='sigmoid'))
```

Figure 4.36 The classifier of model 3 that we add to VGG16.

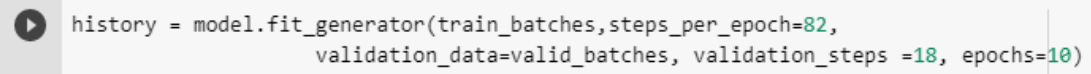
Finally in the next step we must compile our models. In Figure 4.37 we can see the parameters. The hyperparameters are the same for all models and we choose them after many tries and changes and these give us the best results.

```
model.compile(Adam(lr=.0001), loss='binary_crossentropy', metrics=['acc'])
```

Figure 4.37 Compile with the 3 parameters.

We use the **Adam** optimizer and we set a low learning rate (0.0001), a module that contains different types of back propagation algorithm for training our model. (Adam is one of the best optimization algorithms for deep learning). For the loss we used **binary cross entropy** which is good for machine learning problems especially to classify observations into two possible classes (often simply labelled 0 and 1). And for the metics we use **Accuracy**.

The next step is to train the model. Here we use fit generator and we train for 10 epochs, only in the transfer learning did we train the models for 5 epochs. Figure 4.38

A code snippet showing the training of a model using the fit_generator method. The code is as follows:

```
history = model.fit_generator(train_batches, steps_per_epoch=82,  
                             validation_data=valid_batches, validation_steps =18, epochs=10)
```

Figure 4.38 Fit the model.

Now that we have train the models, in the next chapter will see the results of each model and discuss the difference between them.

6. Results and Future Work

In this chapter we will see and compare the results of the models that we have trained and also we will discuss the practices and techniques we used and extract our conclusions. Finally, we suggest future work for our case and in which other fields CNN might be doing well.

6.1. Compare The Results

In this section we will see the results of the different techniques we applied on each model and compare them. First we are going to subjoin the results of the validation set, Accuracy and Loss. After that we are going to subjoin the results of the test set and extract the **Sensitivity**, **Specificity**, **Precision**, **FPR** (False Positive Ratio), **FNR** (False Negative Ratio). For the test set we have not split the images into healthy and bleeding but we have put labels on the them, which is going to help us in predictions, to compare the predicted labels with the true labels and so we can see if the model predicted well. The label for the bleeding images is [1, 0] and for the healthy [0, 1].

Model 1

We will present the Accuracy and Loss for all cases in this model in tables.

Model	Accuracy	Loss	Val Accuracy	Val Loss
Model 1 block 1	100%	0.54%	99.4%	2.3%
Model 1 block 2	99.87%	1.8%	99.4%	2.8%
Model 1 block 3	99.6%	2%	96.1%	9.5%
Model 1_Dp	98.4%	6.1%	98.8%	9.9%
Model 1_Ag	92.2%	20%	94.9%	24%
Model 1_Dp_AG	92.2%	20.8%	94.9%	24.8%
Model 1_TL	96.09%	10%	96.1%	12.3%
Model 1_FT	98%	7.9%	97.7%	6.8%

Table 5.1 All the cases of Model 1 model with Accuracy, Loss, Validation Accuracy, Validation Loss.

Model 2

Model	Accuracy	Loss	Val Accuracy	Val Loss
Model 2	99.8%	0.9%	99.4%	1.4%
Model 2_DPcl	99.8%	1%	98.8%	2.6%
Model 2_DP	97.2%	8.9%	98.3%	17.1%
Model 2_DP_Ag	94.2%	15.9%	95.5%	20.6%
Model 2_DPcl_Ag	95.5%	11.7%	97.4%	7.1%
Model 2_TL	91.7%	20.3%	92.4%	22.5%
Model 2_FT	95.9%	13.9%	96.6%	20.5%

Table 5.2 All the cases of Model 2 model with Accuracy, Loss, Validation Accuracy, Validation Loss.

Model 3

Model	Accuracy	Loss	Val Accuracy	Val Loss
Model 3	98.5%	7.4%	96.3%	13.2%
Model 3_DP_cl	98.8%	6%	97.7%	4.7%
Model 3_DP	97.6%	7.1%	96.6%	9%
Model 3_DPcl_Ag	94.8%	18.4%	93.3%	22.3%
Model 3_Dp_Ag	96.2%	14.3%	97.2%	10.3%
Model 3_TL	82.3%	41.2%	86.9%	31.4%
Model 3_FT	95.4%	12.5%	97.7%	9.1%

Table 5.3 All the cases of Model 3 model with Accuracy, Loss, Validation Accuracy, Validation Loss.

The above results are from the Training and Validation set which is not very accurate, but helps us greatly to see if we have overfitting when we are train the models. But the real deal is to test the models in unseen data and see the results of the predictions, that's why we created the testset, a dataset from images that the models have never seen. This testset have 100 images (50 healthy and 50 bleeding). So the next step is to evaluate our models based on the testset.

First we must mention confusion matrix and his attributes, TP (True Positive), FP (False Positive), TN (True Negative), FN (False Negative).

To begin we must declare the Positives and Negatives. In our case we have the problem to classify the images between healthy and bleed, which are the two states we are working on. So it doesn't really matter which state we are going to choose for positive and negative. In our case we declare the bleeding images as positive and the healthy as negative. So after the prediction we have the predicted labels and the actual labels for the test set, based on that we can see the following definitions.

1. TP: Is the state that a bleeding image was correctly predicted as bleeding.
2. TN: Is the state that a healthy image was correctly predicted as healthy.
3. FP: Is the state that a healthy image was falsely predicted as bleeding.
4. FN: Is the state that a bleeding image was falsely predicted as healthy.

Below is an example of confusion matrix with the TP, TN, FP, FN for our case of problem.

	Predicted 0	Predicted 1
Actual 0	TN	FP
Actual 1	FN	TP

Table 5.4 Example of confusion matrix with TP, TN, FP, FN. picture from [33]

Accuracy is the proportion of true results (both true positive and true negative) among the total number of cases examined.

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

Sensitivity (also called the **true positive rate**, the **recall**, or **probability of detection** in some fields) measures the proportion of actual positives that are correctly identified as such (e.g., the percentage of sick people who are correctly identified as having the condition).

$$\text{Sensitivity} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

Specificity (also called the **true negative rate**) measures the proportion of actual negatives that are correctly identified as such (e.g., the percentage of healthy people who are correctly identified as not having the condition).

$$\text{Specificity} = \frac{\text{TN}}{\text{TN} + \text{FP}}$$

Precision measures the proportion of all positive (TP + FP) identifications was actually correct.

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

Also we have the **FPR** (False Negative Ratio) or (Type I Error) which is the error of proportion to recognize as bleeding (positives) the images that are actually healthy (negatives). We need that to have very low values.

$$\text{FPR} = \frac{\text{FP}}{\text{TN} + \text{FP}}$$

FNR (False Negative Ratio) or (Type II Error) is the error of proportion to recognize as healthy (negatives) the images that are actually bleeding (positives). we must say that it is very important in our case to be very low because people that have bleed may not get the proper healthcare because of the results.

$$\text{FNR} = \frac{\text{FN}}{\text{TP} + \text{FN}}$$

Model 1

Model	TP	TN	FP	FN
Model 1 block 1	43	41	9	7
Model 1 block 2	32	38	12	18
Model 1 block 3	12	45	5	38
Model 1_Dp	28	40	10	22
Model 1_Ag	45	46	4	5
Model 1_Dp_AG	37	50	0	13
Model 1_TL	50	0	50	0
Model 1_FT	50	0	50	0

Table 5.5 The TP, TN, FP, FN for all the cases for model JB_Beed.

Model	Accuracy	Sensitivity	Specificity
Model 1 block 1	84%	86%	82%
Model 1 block 2	70%	64%	76%
Model 1 block 3	57%	24%	90%
Model 1_Dp	68%	56%	80%
Model 1_Ag	91%	90%	92%
Model 1_Dp_AG	97%	74%	100%
Model 1_TL	50%	100%	0%
Model 1_FT	50%	100%	0%

Table 5.6 The Accuracy, Sensitivity, Specificity for model Model 1.

Model	FPR	FNR	Precision
Model 1 block 1	18%	14%	82.6%
Model 1 block 2	24%	36%	72.7%
Model 1 block 3	10%	76%	70.5%
Model 1_Dp	20%	44%	73.6%
Model 1_Ag	8%	10%	91.8%
Model 1_Dp_AG	0%	26%	100%
Model 1_TL	100%	0%	50%
Model 1_FT	100%	0%	50%

Table 5.7 The FPR, FNR, Precision for the model Model 1.

Model 2

Model	TP	TN	FP	FN
Model 2	39	50	0	11
Model 2_DPcl	42	40	10	8
Model 2_DP	40	24	26	10
Model 2_DP_Ag	37	50	0	13
Model 2_DPcl_Ag	41	50	0	9
Model 2_TL	50	0	50	0
Model 2_FT	50	0	50	0

Table 5.8 The TP, TN, FP, FN for all the cases for model Model 2.

Model	Accuracy	Sensitivity	Specificity
Model 2	89%	78%	100%
Model 2_DPcl	82%	84%	80%
Model 2_DP	64%	80%	48%
Model 2_DP_Ag	87%	74%	100%
Model 2_DPcl_Ag	91%	82%	100%
Model 2_TL	50%	100%	0%
Model 2_FT	50%	100%	0%

Table 5.9 The Accuracy, Sensitivity, Specificity for model Model 2.

Model	FPR	FNR	Precision
Model 2	0%	22%	100%
Model 2_DPcl	20%	16%	80.7%
Model 2_DP	52%	20%	60.6%
Model 2_DP_Ag	0%	26%	100%
Model 2_DPcl_Ag	0%	18%	100%
Model 2_TL	100%	0%	50%
Model 2_FT	100%	0%	50%

Table 5.10 The FPR, FNR, Precision for the model Model 2.

Model 3

Model	TP	TN	FP	FN
Model 3	42	48	2	8
Model 3_DPcl	45	42	8	5
Model 3_DP	42	44	6	8
Model 3_DPcl_Ag	45	41	9	5
Model 3_Dp_Ag	46	36	14	4
Model 3_TL	48	2	48	2
Model 3_FT	6	44	6	44

Table 5.11 The TP, TN, FP, FN for all the cases for model Model 3.

Model	Accuracy	Sensitivity	Specificity
Model 3	90%	84%	96%
Model 3_DPcl	87%	90%	84%
Model 3_DP	86%	84%	88%
Model 3_DPcl_Ag	86%	90%	82%
Model 3_Dp_Ag	82%	92%	72%
Model 3_TL	50%	96%	4%
Model 3_FT	50%	12%	88%

Table 5.12 The Accuracy, Sensitivity, Specificity for model Model 3.

Model	FPR	FNR	Precision
Model 3	4%	16%	95.4%
Model 3_DPcl	16%	10%	84.9%
Model 3_DP	12%	16%	87.5%
Model 3_DPcl_Ag	18%	10%	83.3%
Model 3_Dp_Ag	28%	8%	76.6%
Model 3_TL	96%	4%	50%
Model 3_FT	12%	88%	50%

Table 5.13 The FPR, FNR, Precision for the model Model 3.

6.2. Conclusions

Here we will see the best cases of each model and compare the results of each case to see which is the best model. The first we are looking for is the Sensitivity and Precision must be high and balanced and after that all the other metrics and of course we want them to be balanced. For example if we have Sensitivity 99% and Precision 51% or FPR 80% and FNR 0% this model is not balanced.

Model	Accuracy	Sensitivity	Specificity	Precision	FPR	FNR
Model 1_Ag	91%	90%	92%	91.8%	8%	10%
Model 2_DPcl	82%	84%	80%	80.7%	20%	16%
Model 3_DPcl	87%	90%	84%	84.9%	16%	10%

Table 5.14 The best versions of the models.

As we can see all the models are very close. If we can dropdown one model this will be the Model 2_DPcl because it has the biggest gap between Sensitivity and Precision. Also the FNR is higher than the others and as we said before that means there are images with bleeding that the model predicted the opposite. And between and of the two remaining models the Model 1_Ag is better, because we have the same Sensitivity but the other metrics are slightly better. So the best model is the Model 1_Ag.

For the methods and techniques we used on the models in this process we extract the conclusion that both Augmentation and Dropout helped greatly in predictions. In some cases one model with one method had better results than the other and in another model the opposite happened and the compiend methods had good results. For the Dropout we notice that the models performs better when the method applied only in the classifier.

Also we figure out that the numbers of layers in a CNN don't secure better results, so the thought to put many layers in a model to get better results is false, as we mentioned the hidden layers of a CNN is like a black box, we don't know what really happened in these layers. The same happens for the numbers of epochs we train our models. Many epochs doesn't mean that our models are well trained even though the Accuracy is high and the Loss very low.

We can accomplish better performance of a model only with trial and error of the number of layers and with the methods we used. It is very important to understand our problem to use the correct type of model (CNN, RNN...) and also the classification we want to do. For example in our case the difference between healthy and unhealthy images is the values of color red so we don't care about shapes and edges.

The Transfer Learning we used in our models is an example of overtrain. As we said the model (VGG16) we used for this method was trained on imagenet dataset that is not at all similar to our dataset. So the weights of the model is the results of very complex images compared to ours, that had as a result the model to be overtrained and classify the images only as healthy or only as bleeding. Also with VGG16 we ran an example and unfreezed all the layers and trained them only with our dataset and the results aren't very good. Although the VGG16 has 5 blocks which every block has 3 or 4 layers. This proves what we said about the number of layers we use on a model. The same thing happened with the other models if we train them for many epochs. For example, we trained the Model 2 for 60 and 100 epochs and the model overtrained again. That's why we used only 10 epochs and for the transfer learning only 5 but still we didn't get better results. Of course the Accuracy and Loss of the validation set was fine as we had high Accuracy and low Loss and also their curves didn't show overfit. That's why it is important to test the predictions of the models with different images of the dataset than we trained them.

6.3. Future Work

The purpose of the thesis was to classify images between healthy and bleeding from endoscopy capsule in order to reduce the time of examination from doctors. For this purpose we used Neural Networks, a field that the last decade has made huge steps in classification problems. Although we managed to get good results for our classification problem there is still plenty of room for improvements.

The major part of Neural Network and specifically of CNN is the data. It is well known in CNN that a not too good algorithm with enough data would do better than a good algorithm with less data. We had this results with only 1000 images for training, so gathering more images only improves the models. The next step after getting more data it is good idea to go further and more deeply in CNN with more different ways to add layers (like ResNet) to solve problems. Examine other parts (e.g. different optimizers, loss functions, learning rates etc.) and even try different color space like HSV. Also the part of Transfer Learning has plenty of room for improvements in order to give good results. Further on our problem, with enough data we can classify bleeding images and see the scale of bleeding.

As we said Neural Networks and especially CNN have many applications in image classification. The application in healthcare is vast for example with CNN and we can classify brain images from CAT scans for tumor or blood tests.

7. REFERENCES

- [1] Ανίχνευση κακώσεων σε βίντεο ενδοσκοπικών καψουλών με την χρήση ηλεκτρονικού υπολογιστή - Computer aided detection of lesions for endoscopic capsule videos, Alexios Polydorou
- [2] Vehicle Color Recognition using Convolutional Neural Network
<https://arxiv.org/pdf/1510.07391.pdf>
- [3] Neural Networks for Image Recognition: Methods, Best Practices, Applications
<https://missinglink.ai/guides/computer-vision/neural-networks-image-recognition-methods-best-practices-applications/>
- [4] Machine Learning Mastery by Jason Brownlee How to Classify Photos of Dogs and Cats (with 97% accuracy <https://machinelearningmastery.com/how-to-develop-a-convolutional-neural-network-to-classify-photos-of-dogs-and-cats/>)
- [5] Image classification from scratch in keras. Beginner friendly, intermediate exciting and expert refreshing by Rising Odegua <https://towardsdatascience.com/image-detection-from-scratch-in-keras-f314872006c9>
- [6] Loading in your own data - Deep Learning basics with Python, TensorFlow and Keras p.2 <https://pythonprogramming.net/loading-custom-data-deep-learning-python-tensorflow-keras/?completed=/introduction-deep-learning-python-tensorflow-keras/>
- [7] Artificial neural network From Wikipedia, the free encyclopedia.
https://en.wikipedia.org/wiki/Artificial_neural_network
- [8] What Is A Neural Network?
<http://pages.cs.wisc.edu/~bolo/shipyard/neural/local.html>
- [9] What is Deep Learning? <https://machinelearningmastery.com/what-is-deep-learning/>
- [10] Introduction to Deep Learning - Deep Learning basics with Python, TensorFlow and Keras p.1 <https://pythonprogramming.net/introduction-deep-learning-python-tensorflow-keras/>
- [11] Animated math <https://www.3blue1brown.com/neural-networks>
- [12] When to Use MLP, CNN, and RNN Neural Networks
<https://machinelearningmastery.com/when-to-use-mlp-cnn-and-rnn-neural-networks/>
- [13] Google Colaboratory https://tzeny.ddns.net/index.php/Google_Colaboratory
- [14] Getting Started With Google Colab <https://towardsdatascience.com/getting-started-with-google-colab-f2fff97f594c>
- [15] Tensorflow <https://en.wikipedia.org/wiki/TensorFlow>

- [16] TOP FIVE USE CASES OF TENSORFLOW DEEP LEARNING
<https://www.exastax.com/deep-learning/top-five-use-cases-of-tensorflow/>
- [17] An end-to-end open source machine learning platform
<https://www.tensorflow.org/>
- [18] Introduction to Python Deep Learning with Keras
<https://machinelearningmastery.com/introduction-python-deep-learning-library-keras/>
- [19] How to Use the Keras Functional API for Deep Learning
<https://machinelearningmastery.com/keras-functional-api-deep-learning/>
- [20] Image Classifier - Cats vs Dogs <https://towardsdatascience.com/image-classifier-cats-vs-dogs-with-convolutional-neural-networks-cnns-and-google-colabs-4e9af21ae7a8>
- [21] How to Avoid Overfitting in Deep Learning Neural Networks
<https://machinelearningmastery.com/introduction-to-regularization-to-reduce-overfitting-and-improve-generalization-error/>
- [22] How to avoid overfitting on a simple feed forward network
<https://stackoverflow.com/questions/44909134/how-to-avoid-overfitting-on-a-simple-feed-forward-network>
- [23] Dropout: A Simple Way to Prevent Neural Networks from Overfitting
<http://jmlr.org/papers/volume15/srivastava14a/srivastava14a.pdf>
- [24] How to Configure Image Data Augmentation in Keras
<https://machinelearningmastery.com/how-to-configure-image-data-augmentation-when-training-deep-learning-neural-networks/>
- [25] Transfer learning from pre-trained models
<https://towardsdatascience.com/transfer-learning-from-pre-trained-models-f2393f124751>
- [26] About ImageNet <http://www.image-net.org/about-stats>
- [27] American College of Gastroenterology Small Bowel Bleeding
<https://gi.org/topics/small-bowel-bleeding/>
- [28] Application of Artificial Intelligence in Capsule Endoscopy: Where Are We Now?
Youngbae Hwang,1,* Junseok Park,2,* Yun Jeong Lim,3 and Hoon Jai Chun4
<https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6283750/>
- [29] TensorFlow: A system for large-scale machine learning Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M. et al. (2016), Tensorflow: A system for large-scale machine learning, in '12th fUSENIXg Symposium on Operating Systems Design and Implementation (fOSDIg 16)', pp. 265–283.
<https://www.usenix.org/system/files/conference/osdi16/osdi16-abadi.pdf>

[30] TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y. & Zheng, X. (2015), 'TensorFlow: Large-scale machine learning on heterogeneous systems'. Software available from tensorflow.org.

URL: <https://www.tensorflow.org/> <https://arxiv.org/pdf/1603.04467.pdf>

[31] Difference between AlexNet, VGGNet, ResNet and Inception
<https://towardsdatascience.com/the-w3h-of-alexnet-vggnet-resnet-and-inception-7baaaecccc96>

[32] Understanding Activation Functions in Deep Learning
<https://www.learnopencv.com/understanding-activation-functions-in-deep-learning/>

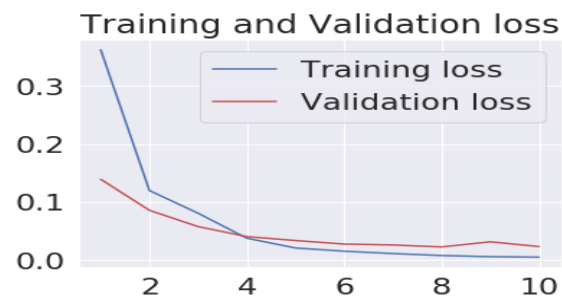
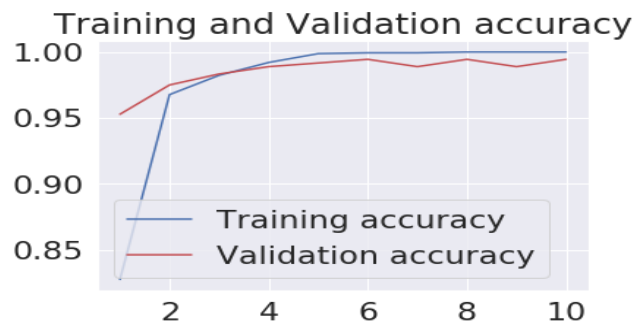
[33]
https://subscription.packtpub.com/book/big_data_and_business_intelligence/9781838555078/6/ch06lvl1sec34/confusion-matrix

Annex 1: Accuracy and Loss

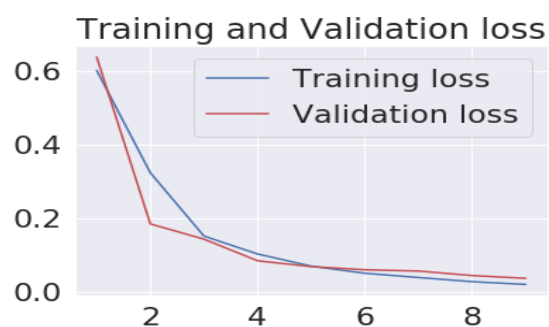
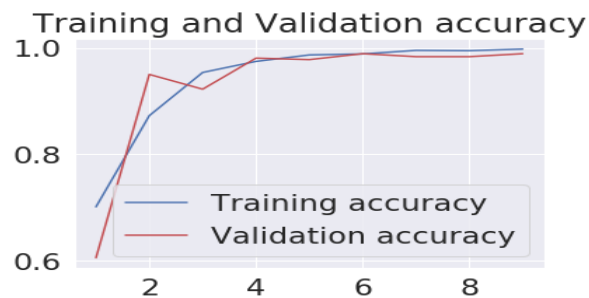
Here we can see the graphs of models during training, in graphs represent the Accuracy and Loss from the training dataset and the Validation Accuracy and Loss from the validation dataset.

Model 1

One block model 1

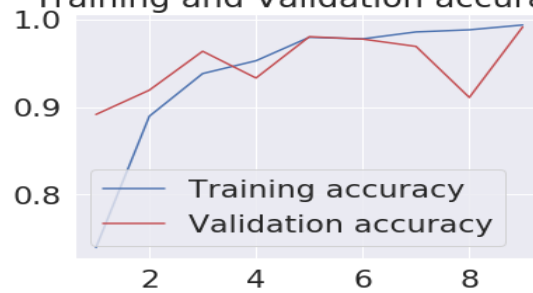


Two block model 1



Three block model 1

Training and Validation accuracy

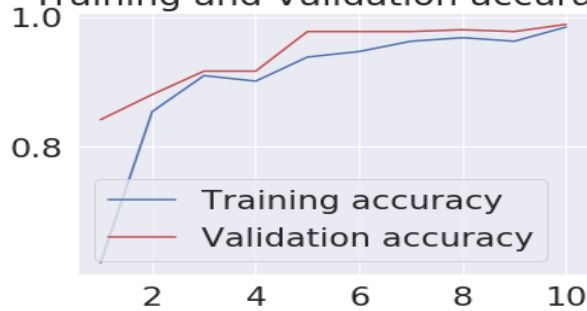


Training and Validation loss



Model 1 DP

Training and Validation accuracy

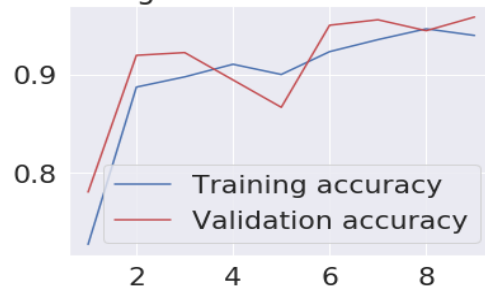


Training and Validation loss



Model 1 Augmentation

Training and Validation accuracy

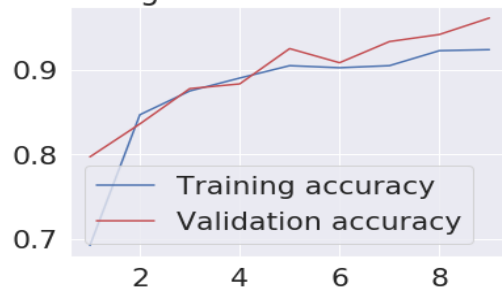


Training and Validation loss



Model 1 DP_Ag

Training and Validation accuracy

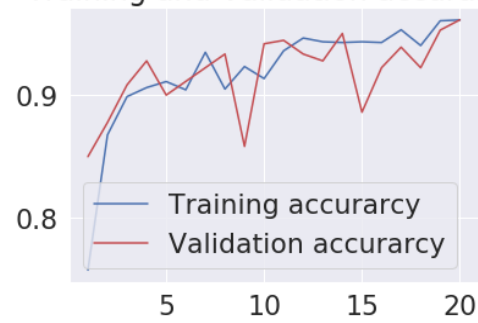


Training and Validation loss



Model 1 TF

Training and Validation accuracy

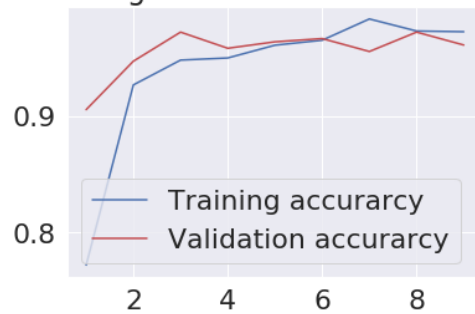


Training and Validation loss

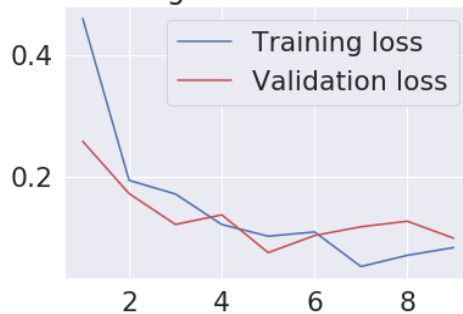


Model 1 TF_FT

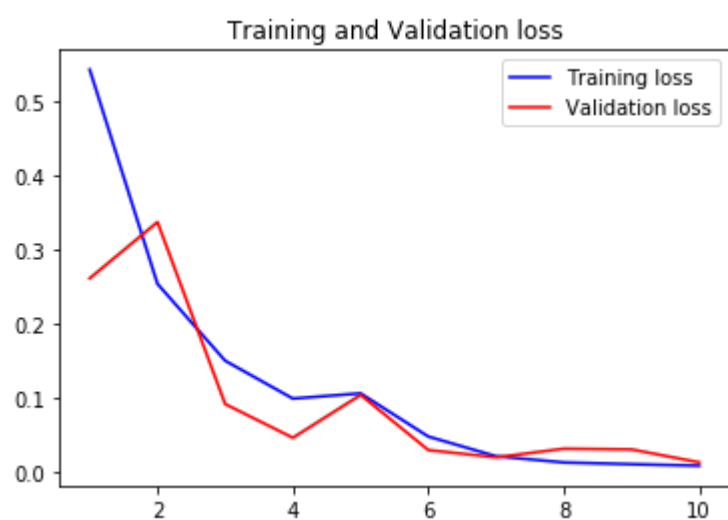
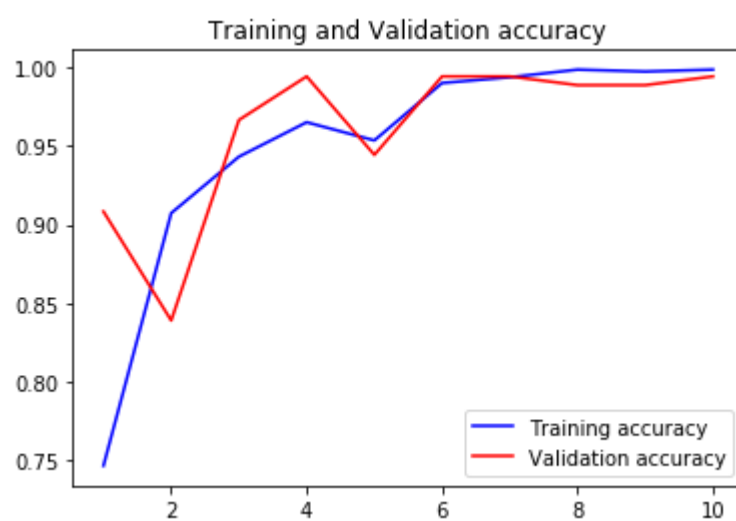
Training and Validation accuracy



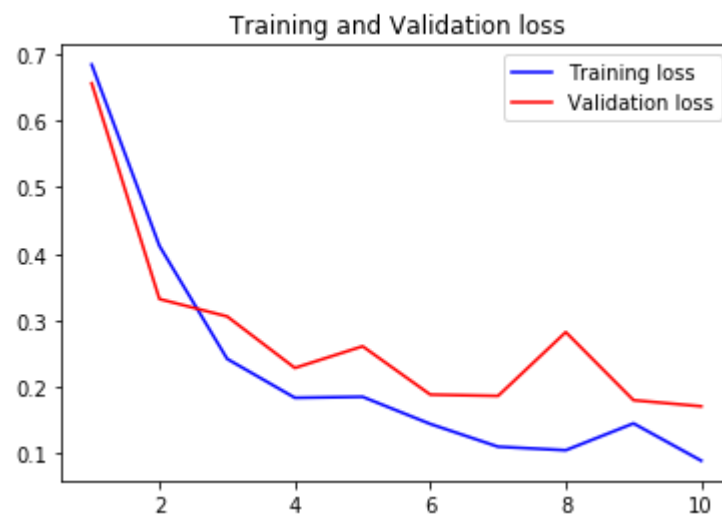
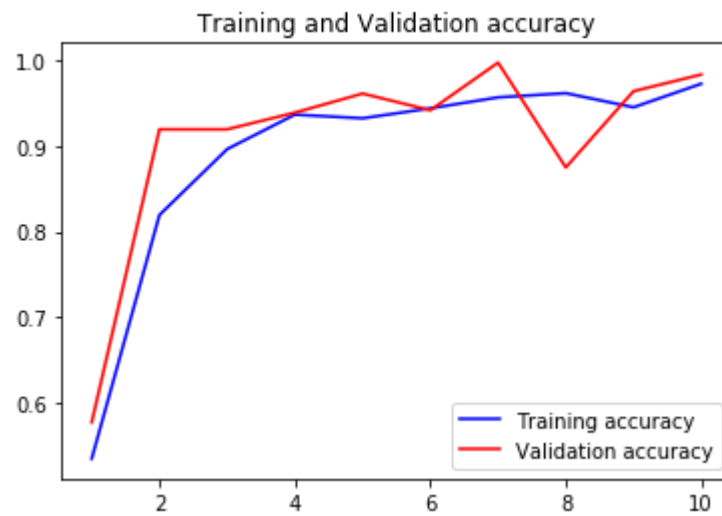
Training and Validation loss



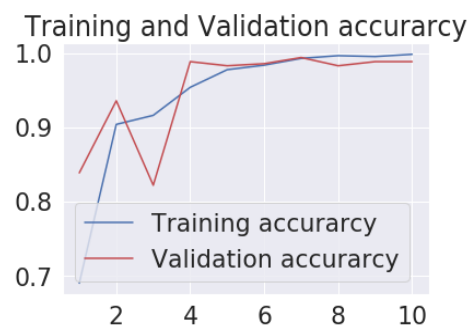
Model 2

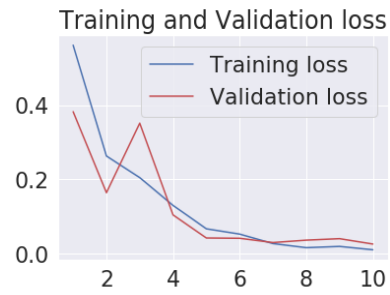


Model 2 DP

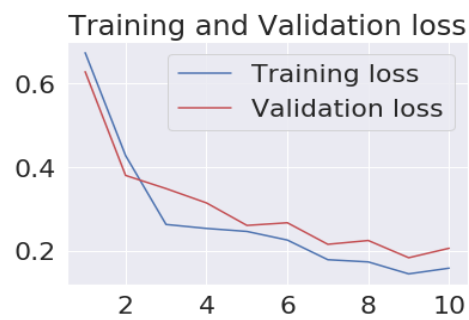
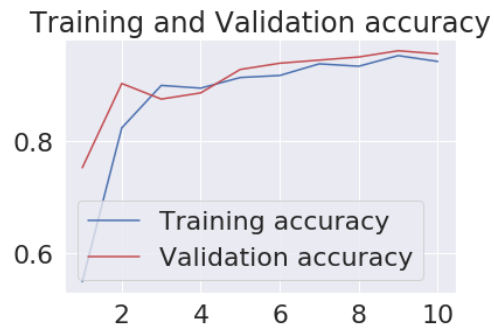


Model 2 DPcl

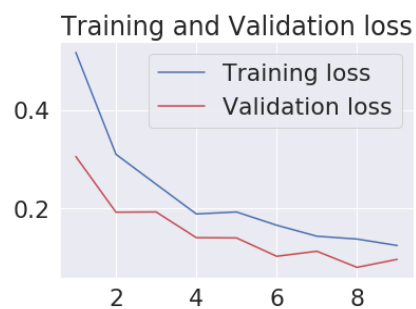
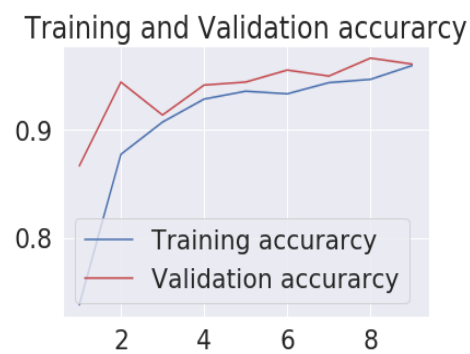




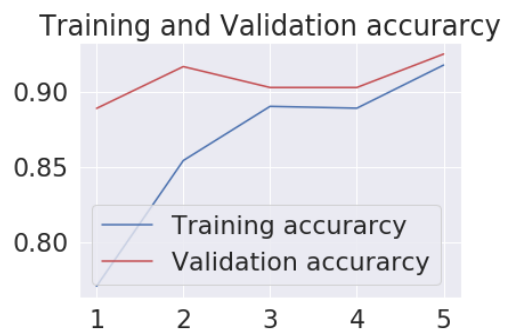
Model 2 DP_Ag



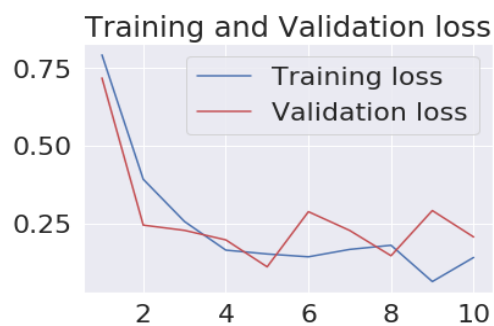
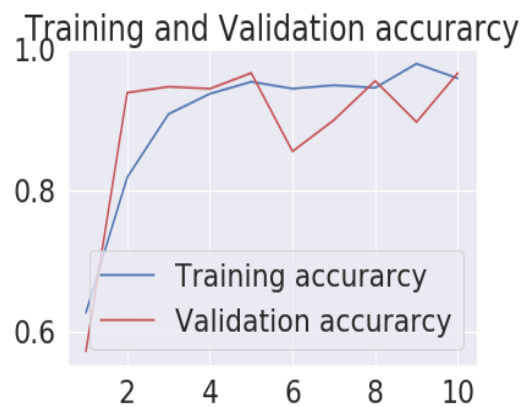
Model 2 DPcl_Ag



Model 2 TL

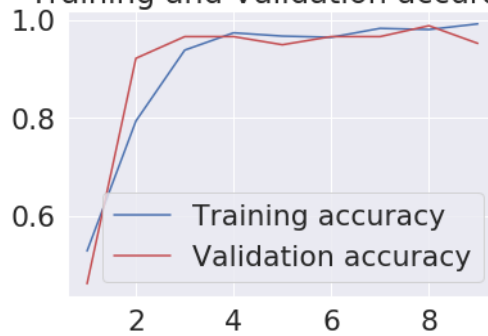


Model 2 TL_FT

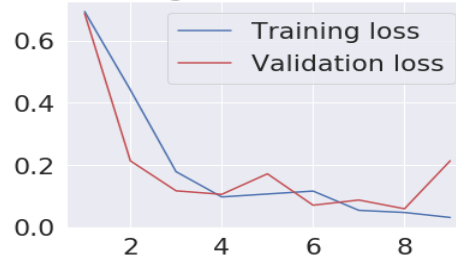


Model 3

Training and Validation accuracy

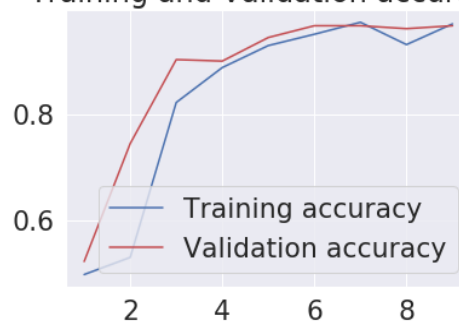


Training and Validation loss

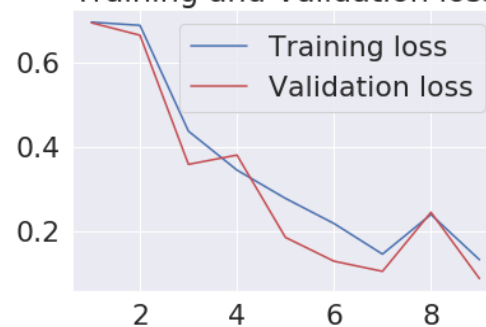


Model 3 DP

Training and Validation accuracy

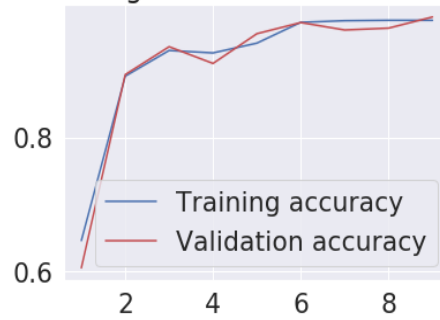


Training and Validation loss



Model 3 DPcl

Training and Validation accuracy

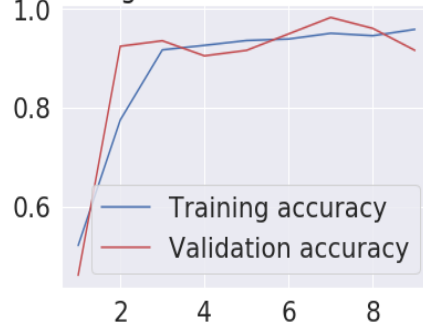


Training and Validation loss

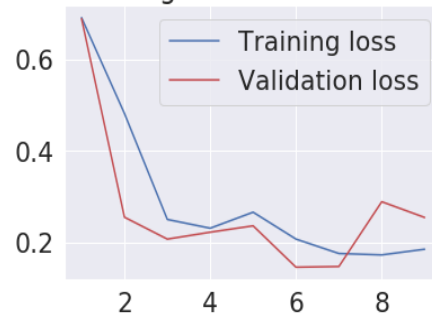


Model 3 DP_Ag

Training and Validation accuracy

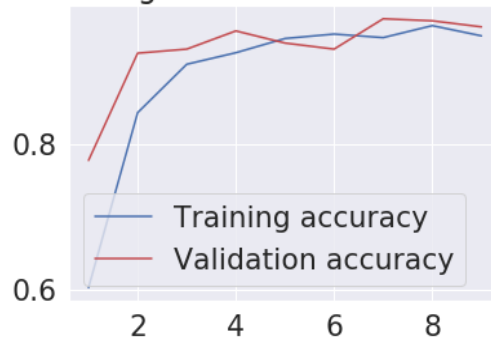


Training and Validation loss

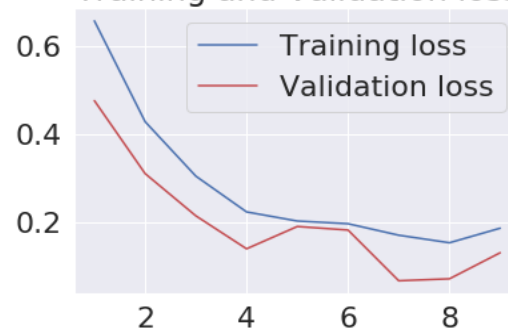


Model 3 DPcl_Ag

Training and Validation accuracy

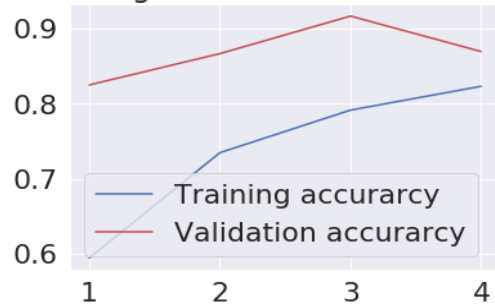


Training and Validation loss



Model 3 TL

Training and Validation accuracy



Training and Validation loss



Model 3 TL_FT

