IVAN FRANKO NATIONAL UNIVERSITY OF LVIV

INTERMATHS

UNIVERSITY OF L'AQUILA

Double-Degree Master's Programme "InterMaths"

Applied and Interdisciplinary Mathematics

MSc Applied Mathematics	MSc Mathematical Engineering
IVAN FRANKO NATIONAL UNIVERSITY OF LVIV	UNIVERSITY OF L'AQUILA

Master's Thesis

Applications of Neural Network Models in Kaggle Competitions

Supervisor

Prof. Vasyl Vavrychuk

Candidate

Grace Anulika Eze
Student ID (UAQ): 254872
Student ID (LVIV): 2718357C

ACADEMIC YEAR 2018/2019

UNIVERSITÀ STATALE
IVAN FRANKO DI LEOPOLI

INTERMATHS

UNIVERSITÀ DEGLI STUDI
DELL'AQUILA

Laurea Magistrale Doppio Titolo "InterMaths"

Applied and Interdisciplinary Mathematics

Laurea Magistrale in Matematica Applicata	Laurea Magistrale in Ingegneria Matematica
UNIVERSITÀ STATALE IVAN FRANKO DI LEOPOLI	UNIVERSITÀ DEGLI STUDI DELL'AQUILA

Tesi di Laurea Magistrale

Applications of Neural Network Models in Kaggle Competitions

Relatore

Prof. Vasyl Vavrychuk

Candidato

Grace Anulika Eze
Matricola (UAQ): 254872
Matricola (LVIV): 2718357C

Anno Accademico 2018/2019

Contents

Abstract

Medical images are not always readily available, it can often be tedious for medical practitioners to not just collect but annotate and analyze these images especially for multi-class image data. In a bid to reduce the workload, aid in diagnosis and treatment of diseases we try to implement and understand how artificial neural networks can be used in the automatic classification of these images. The results gotten from the implementation of the neural network models on our chosen datasets can help in determining how to improve the models used in training these images.

In this work, we implemented the Fully Connected Neural Network (FCN) and the Convolutional Neural Network (CNN) on the Malaria Cell Images dataset and and Kvasir dataset made available by National Institutes of Health (NIH), Vestre Viken Health trust and Cancer Registry of Norway (CRN). These medical image datasets were published on Kaggle for research purposes by data analyst and can also be uploaded by users.

We explore the relevant mathematical background for the various components of the implemented Neural Network (NN) architectures which are used in computer vision for image classification and the mathematics behind how the back propagation works in these networks. We also make comparisms of several optimization algorithms in the keras library and analysed how well they performed on the train and test sets of our chosen datasets as well as investigate how finetuning the hyperparameters of the can affect the models prediction accuracy.

Keywords: Neural Networks; Medical Imaging; Image Classification; Kvasir dataset; Endoscopic Images; Malaria; Optimization; BackPropagation; Hyperparameter Tuning; Regularization

Acknowledgments

Firstly, I want to appreciate God Almighty for His love, blessing steadfast and ever-helping hand upon me and for how He kept me throughout my academic year.

I would like give my sincere thanks to my supervisor, Prof. Vasyl Vavrychuk for his patience and time, for guiding me throughout this work. I would also like to thank Mr Hilary Okagbue and Dr. Mrs Eke. The whole student and lecturers of the Mathmod-Intermaths community. To the very good friends I made during my study under this platform who felt like family, Thank you

Finally, my special thanks goes to my parents Mr and Mrs Matthew Eze for your prayers, love and encouragement. I want to say how much I love my family and also thank my amazing only brother Mr Emmanuel Somadina Eze Matthew, his lovely wife Mrs Lotachukwu Eze, my cousin brother who is like a second brother to me Mr. Philip Eze, my beloved sisters Miss. Chinyenye Ruth Eze, Miss. Chiamaka Goodness Eze, Miss. Gift Chiwendu Eze and my favourite baby sister Miss. Favour Oluchi Eze for always being there and the unbreakable bond God has placed between us. To my loved ones and best friend Miss. Onome Ofoluwa, Miss. Onyinye Umeh, Miss. Stella Uche back home who have been my backbone and my support system. Thank you so much.

Chukwu Nna'm Nara Ekele Mo!

Abbreviation List

NN	Neural Networks
FCN	Fully Connected Neural Network
CNN	Convolutional Neural Network
GD	Gradient Descent
SGD	Stochastic Gradient Descent
NIH	National Institutes of Health
GI	Gastro-Intestinal
CRN	Cancer Registry of Norway
WSI	Whole Slide Images

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

Neural network architectures over the years have become the state-of-the-art method used in computer vision. Neural networks are a collection of precise rules which is used in recognizing patterns in data generated from various real life applications such as medicine, finance, time series analysis, security and much more. In computer vision, deep learning models aim to extract high dimensional image data from various sources such as pictures, videos or from medical imaging devices which are then analyzed and used to make informed decisions. The network tries to understand and make a correlation between the input features from the data and the specific target labels, it then uses a set of learning rules known as the backpropagation which adjusts the predictions made by the network.

Medical images are not always readily available, it can often be tedious for medical practitioners to not just collect but annotate and analyze these images especially for multi-class image data. In a bid to reduce the workload, aid in diagnosis and treatment of diseases we try to understand and implement how artificial neural networks can be used in the automatic classification of these images. The results gotten from the implementation of the neural network models on our chosen datasets can help in determining how to improve the models used in training these images.

In this work, the aim is to implement and analyze several neural network models on certain medical image datasets via the kaggle platform. This is in a bid to understand the mathematics behind these neural network models, try to construct neural networks which can actively recognize patterns from medical images, properly classify these images and investigate how the backpropagation algorithm iteratively learns.

1.2 Kaggle Datasets

Kaggle is the largest community of data scientists which challenges data scientists as well as machine learning enthusiasts to solve predictive analytic problems. It is a public data platform owned by Google, where several competitions dataset are published by companies and machine learners. Users can also upload external data for research competition and share with data analysts to work on producing modern machine learning models used in analyzing the data and also gain new ideas. These Kaggle competitions have had several impacts in various research and the predictive models implemented in this work are used for image classification, which involves trying to predict the content of an image and assigning the appropriate class labels to the image. An advantage of implementing the models on the Kaggle kernel is access to large GPU memory.

The datasets which was chosen from Kaggle are some medical imaging datasets namely

- The Malaria Cell Images Dataset and
- Kvasir Dataset

1.2.1 Malaria Cell Images Dataset

There are over 87 countries with a high estimate of malaria epidemic with a high percentage of this life threatening disease in the sub-Saharan African region. This disease is caused by the plasmodium parasites from

the bites of the female anopheles mosquitoes.

Most medical practitioners confirm the presence of the parasite via microscopic examination and this is done by collecting blood samples from patients. The sample is then spread as a thick or thin blood smear and the presence of the malaria parasite is determined if the blood smear slide image shows plasmodium falciparum parasites infecting some of the patients red blood cells.

In order to reduce the time taken and the strain for the microscopic examiners, the National Institute of Health (NIH) collected thin blood smear slides from 150 plasmodium falciparum infected and also blood smear slides of 50 healthy patients. The compiled dataset which is now published on Kaggle has a total of 27,558 cell image patches [1, 2, 3] from patients with the number of images equally divided int two distinct classes: parasitized and uninfected cells since there is no target label file.

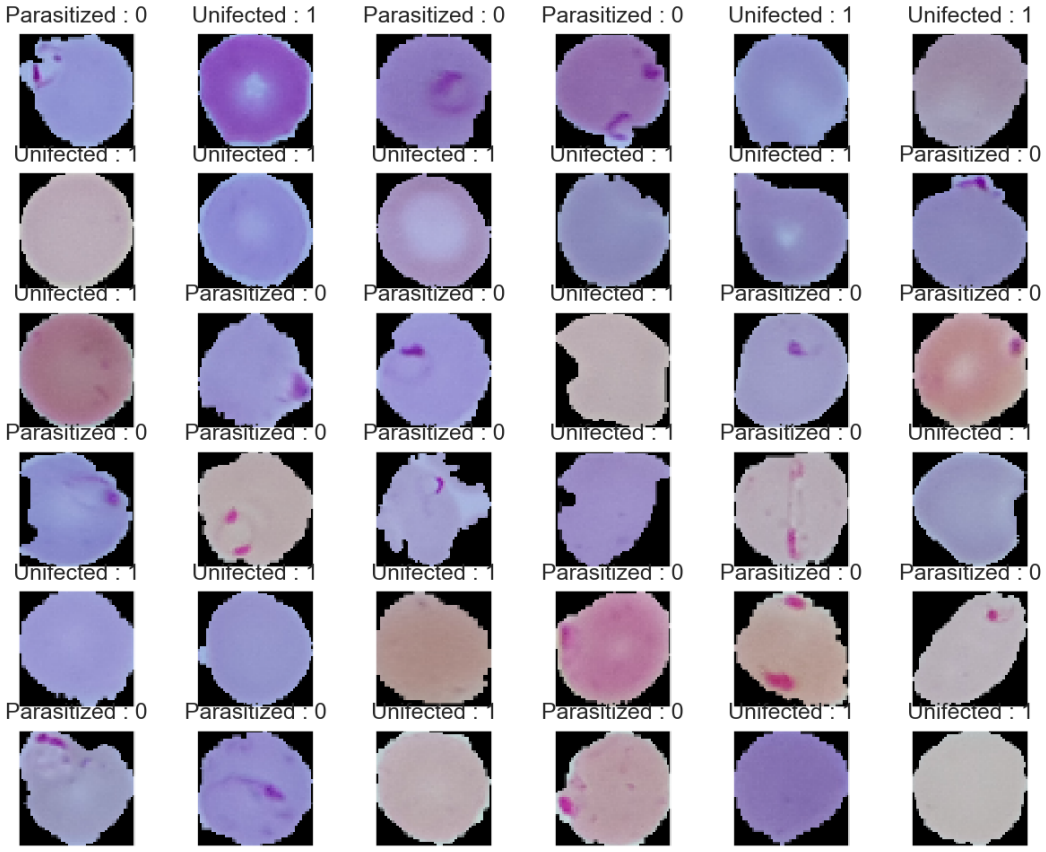


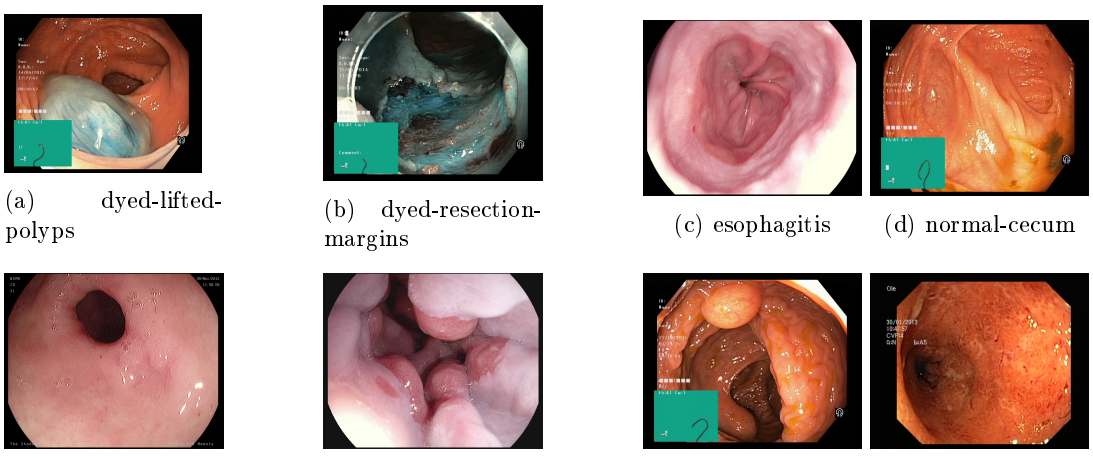
Figure 1.1: Cells from the Pre-processed Malaria Dataset

1.2.2 Kvasir Dataset

There are several types of disorders of the human digestive tract also known as the gastrointestinal (GI) tract [4, 5]. These digestive diseases such as gallstones, intestinal problems, esophageal , colon and stomach cancers have resulted in approximately 3 million cases with a record of 2 million deaths per year. The test for such disease in the digestive system are investigated through the endoscopic examinations which is a medical imaging technique where an endoscope in this case is used to examine the upper GI tract which includes the esophagus, stomach, first part of small bowel, the large bowel (colon) and rectum. The videos of the examinations are

taken to in order to properly detect colorectal cancer, polyps and removal of possible precancerous lesions are essential but the first stage is the detection [5, 6, 7] and classification of these endoscopic images. This may vary from one gastroenterologists to another and wrong diagnosis may increase the risk or the severity of a patient getting these digestive diseases. Proper classification the findings are of upmost importance as it will influence decision-making on treatment and follow-up[8, 9, 10].

The Kvasir dataset public multiclass image dataset which is collated from the Vestre Viken Health Trust (Norway) which can help in aiding research which can improve the health-care system and make it more economical. They collect the data with an endoscopic equipment, the images are carefully seperated by expert endoscopists at Vestre Viken Health Trust and the Cancer Registry of Norway (CRN). [4] The dataset contains eight (8) classes: dyed and lifted polyp, dyed resection margins, esophagitis, normal cecum, normal pylorus, normal z-line, polyps and ulcerative colitis. It consist of thousands of images per class which are organized in different folders according to their content. The normal cecum, normal pylorus and normal z-line are anatomical landmarks while esophagitis, polyps, ulcerative colitis are pathological findings and the dyed lifted polyp, the dyed resection margins are images related to removal of lesions. [8, 5]



(a) dyed-lifted-polyps (b) dyed-resection-margins (c) esophagitis (d) normal-cecum

(e) normal-pylorus (f) normal-z-line (g) polyps (h) ulcerative-colitis

Figure 1.2: Endoscope images of the 8 classes from the Kvasir dataset

The polyps as shown in figure 1.2g are clumps of cells (lesions) on the lining of the large bowels which might be indicators of cancer. They are either flat, elevated or pedunculated, and can be distinguished from normal mucosa by color and surface pattern. [9, 6] They can be removed via endoscopic mucosal resection (EMR) where some liquid is injected under the polyp, then the polyp is lifted from the underlying tissue,captured and removed by use of a snare. This reduces the risk of damaging the inner layers of the gastrointestinal walls. The polyps are stained with the diluted indigo carmine dye is added to help in accurately identifying the polyp margins.

The dyed and lifted polyps in figure 1.2a shows a light blue polyp margins.The resection margins are resection sites of polyp removal as shown in figure 1.2b and useful in evaluating whether the polyp has been completely removed or if there are any residual polyp left. Esophagitis is an inflammation of the tissues of the oesophagus usually caused by the reverse of acid flow from the stomach along the z-line where the oesophagus meets the stomach [5]. A sample is shown in the figure 1.2c caused by the gastroesophageal reflux disease. A sample of normal z-line as shown in figure 1.2c marks the demarcation border between the esophagus and the stomach where white mucosa from the esophagus meets the red gastric mucosa. The normal cecum is the anterior pouch like part of the large intestine which has the appendiceal orifice that is shaped like a crescent shaped slit and indicates the location of the appendix. This can be seen in figure 1.2d. The normal pylorus is the region around the opening which leads from the stomach into the duodenum. It regulates the movement of food from the stomach and prepares to digest chyme in the small intestine. Figure 1.2e shows an endoscopic image of a normal pylorus viewed from inside the stomach. A way to recognize it that it is like a smooth visibly dark circle with similar pink stomach mucosa. Ulcerative colitis as shown in figure 1.2h is an inflammatory bowel disease in the digestive tract which affects the lining of the colon and rectum. It may vary based on the severity of the inflammation which varies from none, mild, moderate and severe. The mucosa appears swollen and red for a mild ulcerative colitis while the ulcerations appear more noticable in moderate cases.[8, 4, 10]

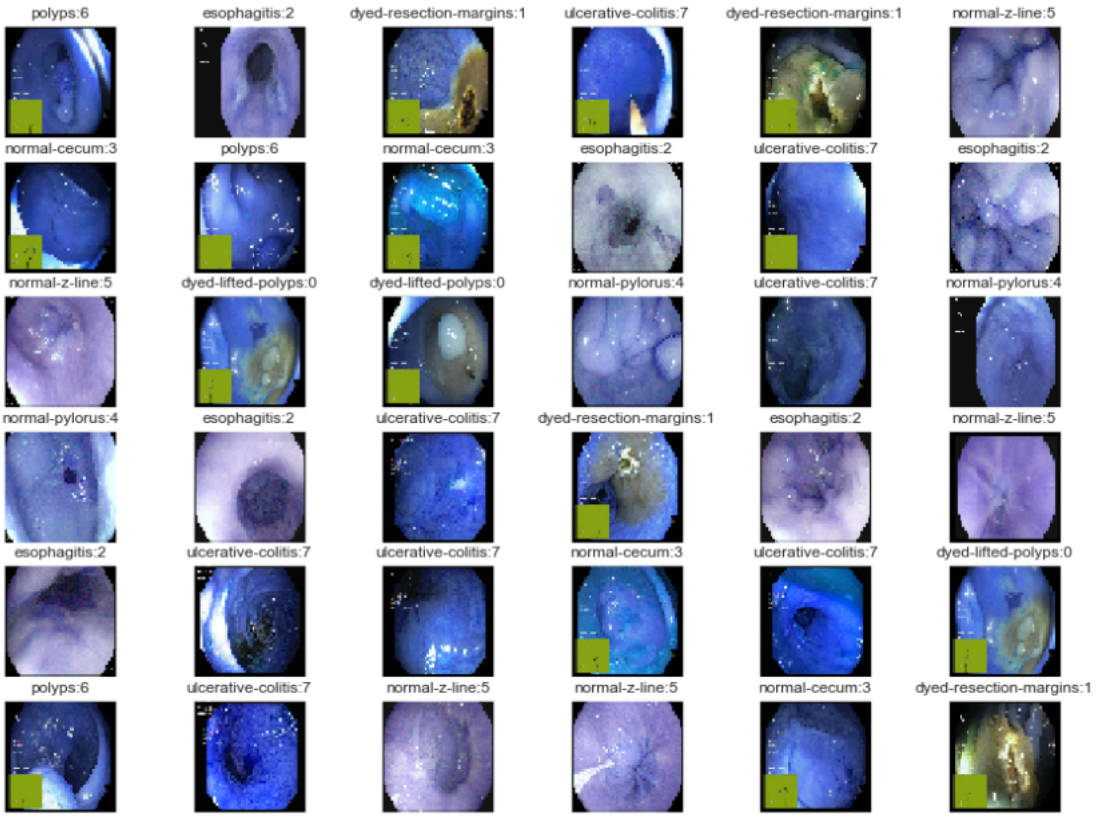


Figure 1.3: Cells from the Pre-processed Kvasir Endoscope Images Dataset

1.3 Overview of Project Work

This project is partitioned into four chapters. In chapter 1, we give a general introduction and motivation of the work. we also introduce the background information for the malaria cell images dataset and and open Kvasir dataset made available by NIH and CRN. These datasets was published on Kaggle for research purposes by data analyst and me respectively. The relevant literature's relating to the subject of study was also discussed in chapter 1. In chapter 2, we explore the mathematical background for various components of neural network architectures which are used in computer vision and specifically for image classification. In chapter 3, we discuss how these neural network models were applied on the chosen datasets. In chapter 4, we make comparisons of several optimizers and how they perform on our train and test sets. We also investigate how finetuning the hyperparameters can affect the models prediction accuracy.

1.4 Literature review

Thomas de Lange et al [5] discussed several methodologies in machine learning which can be used to improve the performance of endoscopy in the GI tract. They discussed useful tools for computer assisted diagnosis (CAD) systems in detection of various GI cancers such as CNN's , and also promising method for image analysis is generative adversarial networks (GANs). GANs consist of two neural networks competing with each other in a zero-sum game framework during the training phase. The generator network generates new data instances using an inverse convolutional network by upsampling random noise to an image. The other network, the discriminator, takes the generated image and the training set and checks for authenticity. This means that the discriminator decides whether the data belong to or are classified in the actual training dataset or not. GANs can also be defined as conditional GANs that have an image as input instead of random noise and that transform this image into another image [6, 7].

F. G. Zanjani et al [11] presented and evaluated automatic breast cancer metastases detection in lymph nodes Whole-Slide Images (WSI's). The detection is performed in slide-level and patient-level processing. The pN-stage for every patient is determined by the number of positive lymph nodes that consists of 5 categories. They used convolutional neural networks for slide level detection of tumor cells. They found that by using

test-time color augmentation and false positive samples bootstrapping, the prediction improves significantly. The pN-stage evaluation has been done as post processing stage for detected positive regions by using blob analysis and DBSCAN clustering. They evaluated their approach on our validation set and unrevealed test data of Camelyon17 dataset.

Alex Krizhevsky [12] trained a large, deep convolutional neural network to classify the 1.2 million high resolution images in the ImageNet LSVRC-2010 contest into the 1000 different classes. On the test data, they achieved top-1 and top-5 error rates of 37.5% and 17.0% which is considerably better than the previous state-of-the-art. The neural network, which had 60 million parameters and 650,000 neurons, consisted of five convolutional layers, some of which are followed by max-pooling layers, and three fully-connected layers with a final 1000-way softmax. To make training faster, they used non-saturating neurons and a very efficient GPU implementation of the convolution operation. To reduce overfitting in the fully-connected layers they employed a recently-developed regularization method called dropout that proved to be very effective. They also entered a variant of their model in the ILSVRC-2012 competition and achieved a winning top-5 test error rate of 15.3%, compared to 26.2% achieved by the second-best entry.

Andrey et al. [13] examined the evolution of the most efficient models and trends in development of architecture of convolutional neural networks, which are currently used for classification of images include in the international competition, ILSVRC. More precisely, the key features of architecture and its annual variations are revealed on the background of increasing efficiency of practical application of these networks. The data of numerous experiments conducted over the past few years are summarized, classes of applied problems are analyzed, and estimates are given for an effectiveness of use of the considered convolutional neural networks. In fact, these performance estimates are based on evaluation of probability of adequate classification of images. On this basis, a generalized algorithm is formulated, and practical recommendations are proposed taking into account the problem features.

Shenghua Cheng et al [14] explored the use of computers in automatic detection of breast cancer metastases to reduce the workload of the pathologists and reducing the cost of diagnosis. They presented a learning-based method for automated prediction of breast cancer metastasis stages. The method was trained and validated on Camelyon17 challenge datasets. The method consists of three parts: tissue extraction, tumor region detection and cancer metastasis stage prediction. The goal of Camelyon17 is to predict the pN-stage of each patient, which is decided by the metastasis status of his five lymph node WSIs. Thus, the key is to accurately predict WSI metastasis status (normal, isolated tumor cells, micro-metastases and macro-metastases). They presented a method consisting of three parts: tissue extraction, tumor region detection and patient pN-stage prediction as a solution to the problem. The tissue region extraction is based on the difference of pixel RGB values. The tumor regions in WSIs are detected by GoogLeNet-v3 finetuned on the ImageNet weights . The WSI status is classified by random forest algorithm with morphological features extracted from the predicted tumor heatmap, then the pNstage can be obtained by the given rule where the three key points for accurately predicting cancer metastasis stages are: 1) effectively deal with the imbalance of tumor and normal samples and the imbalance of tumor regions of various cases (isolated tumor cells, micro-metastases and macrometastases): 2) how to train the very deep network on the limited datasets of simple samples and hard samples; 3) how to prevent overfitting when learning a classifier of WSI metastasis status on 500 WSIs. Aiming at these challenges, we designed some effective strategies for training sample preparation, network training and the WSIs metastasis status classification. The method they presented includes three parts: tissue extraction , tumor region detection and cancer metastasis stage prediction.

Neha sharma et al. [2] presented an emphirical analysis of the performance of popular CNN's for identifying objects in real time video feeds. They analyzed the performance of differentof three popular networks: Alex Net, GoogLeNet,and ResNet50 on the three most popular data sets ImageNet, CIFAR10,and CIFAR100 for their study, since, testing the performance of a network on a single data set does not reveal its true capability and limitations. They are used videos as testing datasets but not as the training dataset. Their analysis showed that GoogLeNet and ResNet50 are able to recognize objects with better precision compared to Alex Net. They also talked about the performance of trained CNN's vary substantially across different categories of objects and discussed the possible reasons for this.

Chapter 2

Topology of the Neural Network Models and Back-Propagation

2.1 Fully Connected Neural Network Model

A Fully Connected Neural Network (FCN) can be referred to as a universal approximator which maps certain features extracted from the data of a function to target output labels depending on how close the output from the network matches the target label. This network is usually comprised of three (3) basic layers namely: the input layer, the hidden layers and the output layers.

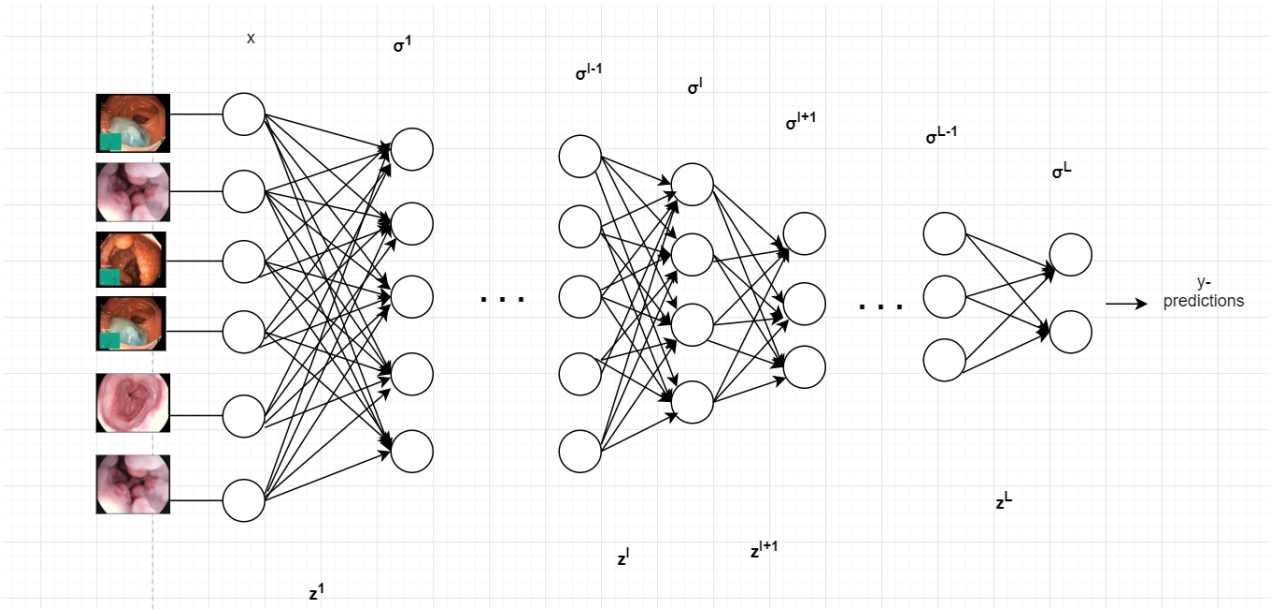


Figure 2.1: FCN model architecture

2.1.1 Initializing The Network Parameters and Forward Propagation

Given the input data $\mathbf{x} = [x_1, x_2, \cdots, x_m]^T \in \mathbb{R}^m$ for m-training samples in an l-layered neural network with $l = 1, \cdots, L$, the first layer being $l = 1$, $l + 1$ hidden layers and $l = L$ as the output layer. The network has learnable parameters called the weights, $\mathbf{w}$ and the biases, $\mathbf{b}$ which are initialized during the training of the network. The input data $x_i,\ i = 1, \cdots, m$ is being feed into the input layer and the neurons in the layer does a dot product between the weights, $\mathbf{w}$ and the inputs, $\mathbf{x}$ then it adds a bias, $\mathbf{b}$. The weighted sum as

$$output = \sum weight \times input \ \ data + bias \tag{2.1}$$

$$\mathbf{z} = w_1 x_1 + w_2 x_2 + \cdots + w_m x_m + b, \qquad i = 1, 2, \cdots, m \tag{2.2}$$

$$= \sum (\mathbf{w}^T \cdot \mathbf{x}) + b \tag{2.3}$$

The first layer only contains inputs so there are no weights and bias, but the hidden layers and the output layer have weights and bias terms. We have our parameters as $w_i, b_i, \quad i = 1, 2, \cdots, m$.

Generally speaking for the layers, it does a sum of the linear combination of the weights and the inputs with bias computation and can be defined as

$$z_i^{[l]} = \sum_i (w_i^{[l]})^T \cdot x_i^{[l-1]} + b_i^{[l]}, \qquad i = 1, 2, \cdots, m, \quad l = 1, 2, \cdots, L. \tag{2.4}$$

where the matrix of weights has dimensions as the tuple (size of layer l , size of layer $l-1$) i.e $w_i^{[l]} = [w_1^{[l]}, w_2^{[l]}, \cdots, w_{m-1}^{[l]}, w_m^{[l]}]^T \in \mathbb{R}^{(m_l, m_{l-1})}$ and the bias vector as (size of layer l , 1) i.e $b_i^{[l]} = [b_1^{[l]}, b_2^{[l]}, \cdots, b_{m-1}^{[l]}, b_m^{[l]}] \in \mathbb{R}^{(m_l, 1)}$.

In implementing the forward propagation, the result gotten from the computation is then passed through an activation function. The activation function is a non-linear function applied to a vector component gotten from the input neurons of the previous layer. This introduces non-linearity into the network layers and makes it capable of learning and computing complex data functions such as media data which are high-dimensional. If we have $\sigma(z)$ as an activation function and m represents the number of our training samples from our input data,if the inputs are passed through the activation then we have that:

$$\sigma_i^{[l]} = \sigma^l(z_i^{[l]}) \tag{2.5}$$

$$= \sigma^l\left(\sum_i (w_i^{[l]})^T \cdot x_i^{[l-1]} + b_i^{[l]}\right) \tag{2.6}$$

Thus for each layer we can say that:

$$\begin{aligned}
z_1^1 &= \sum (w_1^{[1]})^T \cdot x^{[0]} + b_1^{[1]} \quad \longleftrightarrow \sigma_1^{[1]} = \sigma^1(z_1^{[1]}) \\
z_2^1 &= \sum (w_2^{[1]})^T \cdot x^{[0]} + b_2^{[1]} \quad \longleftrightarrow \sigma_2^{[1]} = \sigma^1(z_2^{[1]}) \\
&\vdots \\
z_m^L &= \sum (w_m^{[L]})^T \cdot \sigma^{L-1} + b_m^{[L]} \quad \longleftrightarrow \sigma_m^{[L]} = \sigma^L(z_m^{[L]})
\end{aligned}$$

Some common activation functions are tabulated below

Table 2.1: Activation Functions

Activation	Formula
Sigmoid	$\sigma(z_i) = \frac{1}{1+\exp^{-z_i}}$
ReLU	$\sigma(z_i) = max(0, z_i)$
Softmax	$\sigma_i(z_i) = \frac{\exp^{-z_i}}{\sum_j \exp^{-z_j}}$

2.1.2 Loss Function and Error computation

Let $\mathfrak{L}(\hat{y}, y)$ be the loss function of our model parameters, the loss is an evaluation metric which aims to measure the variance between the predicted outputs, $\hat{y}$ of the neural network and the target class labels, y. It is used in measuring the performance of the network model. The most common loss function used in practice is the logarithmic loss (cross entropy) function. The cross entropy loss is used to compute the goodness of the predicted probabilities of the outputs with the good predictions having low values and the bad predictions having high loss values. For each target label, it multiplies the logarithm of the correctly predicted probabilities of the class and also multiplies the predicted probabilities of the other classes (i.e 1 - correctly predicted probabilities) by its logarithm for each sample of the class. The log of values which lies between 0 and 1 is negative, thus since we are taking log of the probabilities of certain class distributions then we take the negative log in order to obtain positive values for the loss. We have the log loss function when we take the average over a set of m training samples as:

$$\mathfrak{L} = -\sum_{i=1}^{m}\sum_{i}^{C} y_i \log(\hat{y}_i) \qquad C = no\ of\ classes \tag{2.7}$$

$$= -\frac{1}{m}\sum_{i=1}^{m} y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i) \qquad C = 2, binary \tag{2.8}$$

The binary cross entropy loss (log loss) is the loss function with 2 classes while the categorical cross entropy loss is still the cross entropy loss function but with a softmax activation function and can be used for multi-classification as well where the network is trained to output the probabilities over the C classes for each image.

2.1.3 Back Propagation and Optimization

The back propagation is an algorithm that computes the derivative of the loss with respect to the models parameters for every layer in the neural network. During training of the network's model, the error is computed after the predictions are made by the model then the derivative of the error is then computed with respect to the parameters and then propagated backwards from the output layers to the first layer.

The aim of back propagation is to minimize our objective function which is the loss function and get the optimum model parameters (i.e weights, bias) that properly minimizes the loss, $\mathfrak{L}$. The output of the neural network is a function of each layer which is computed during forward propagation. The back propagation uses the chain rule to compute the gradient of these functions, the derivative of the error for one hidden layer can be used to compute for the derivatives of the error for the previous layers. Back propagation tells use how quickly the loss changes when our parameters are changed slightly. We seek to find the gradient with respect to the model's parameters, thus we compute $\frac{\partial \mathfrak{L}}{\partial \mathbf{w}}$ and $\frac{\partial \mathfrak{L}}{\partial \mathbf{b}}$. For the networks computation of the output vector for the $m-th$ element in the last layer

$$\sigma_m^l = \left[\sigma\Big(\sum_m w_{m,n}^l \Big[\cdots\Big[\sigma\Big(\sum_j w_{j,k}^2\Big[\sum_i w_{i,j}^1 x_j^1 + b_i^1\Big)\Big] + b_j^2\Big)\Big]\cdots\Big]_{m-1} + b_m^L\Big)\right]_m \tag{2.9}$$

Assuming we start the gradient computation from the last layer, L we have

$$\mathbf{z}^L = \mathbf{w}^L \sigma^{L-1} + \mathbf{b}^L \tag{2.10}$$

$$z_i^L = \sum_i (w_{i,j}^L)\sigma_j^{L-1} + b_i^L \tag{2.11}$$

$$\sigma_i^L = \sigma(z_i^L) \tag{2.12}$$

with z_i^l as the activation function for neuron i in the layer l, $w_{i,j}^l$ with the weights for each layer l for the $i-th$ row and $j-th$ column. We have $j-$ neurons in the $l-1$ layer. Consider we use the softmax function on the last layer which is given by

$$\sigma_i^L = \hat{y}_i^L = \sigma(z_i^L) = \frac{e^{z_i^L}}{\sum_{j=1}^{m} e^{z_j^L}} \tag{2.13}$$

then we have that the derivative of the activation, σ_i^L with respect to the parameters z_i^L given by

$$\frac{\partial \sigma_i^L}{\partial z_i^L} = \frac{\partial \sigma(z_i^L)}{\partial z_i^L} = \frac{\partial}{\partial z_i^L}\left[\frac{e^{z_i^L}}{\sum_{j=1}^{m} e^{z_j^L}}\right] \tag{2.14}$$

$$= \frac{e^{z_i^L}\sum_{i=1}^{m} e^{z_i^L} - e^{z_i^L}\cdot e^{z_i^L}}{\left[\sum_{i=1}^{m} e^{z_i^L}\right]^2} \tag{2.15}$$

$$= \frac{e^{z_i^L}\left(\sum_{i=1}^{m} e^{z_i^L} - e^{z_i^L}\right)}{\sum_{i=1}^{m} e^{z_i^L}\cdot\sum_{i=1}^{m} e^{z_i^L}} \tag{2.16}$$

$$= \frac{e^{z_i^L}}{\sum_{i=1}^{m} e^{z_i^L}}\cdot\frac{\left(\sum_{i=1}^{m} e^{z_i^L} - e^{z_i^L}\right)}{\sum_{i=1}^{m} e^{z_i^L}} \tag{2.17}$$

$$= \frac{e^{z_i^L}}{\sum_{i=1}^{m} e^{z_i^L}} \cdot \left(\frac{\sum_{i=1}^{m} e^{z_i^L}}{\sum_{i=1}^{m} e^{z_i^L}} - \frac{e^{z_i^L}}{\sum_{i=1}^{m} e^{z_i^L}} \right) \tag{2.18}$$

$$= \frac{e^{z_i^L}}{\sum_{i=1}^{m} e^{z_i^L}} \cdot \left(1 - \frac{e^{z_i^L}}{\sum_{i=1}^{m} e^{z_i^L}} \right) \tag{2.19}$$

$$= \sigma_i^L (1 - \sigma_i^L) \tag{2.20}$$

$$= \hat{y}_i^L (1 - \hat{y}_i^L) \tag{2.21}$$

Now we compute the derivatives for the weights of the $L-$th layer

$$\frac{\partial \mathfrak{L}}{\partial w_{i,j}^L} = \frac{\partial \mathfrak{L}}{\partial \hat{y}_i^L} \cdot \frac{\partial \hat{y}_i^L}{\partial z_{i,j}^L} \cdot \frac{\partial z_i^L}{\partial w_{i,j}^L} \tag{2.22}$$

$$\tag{2.23}$$

where

$$\frac{\partial \mathfrak{L}}{\partial \hat{y}_i^L} = \left(\frac{-y_i^L}{\hat{y}_i^L} + \frac{1 - y_i^L}{1 - \hat{y}_i^L} \right) \qquad i = 1, \cdots, m \tag{2.24}$$

$$\frac{\partial \hat{y}_i^L}{\partial z_{i,j}^L} = \hat{y}_i^L (1 - \hat{y}_i^L) \tag{2.25}$$

$$\frac{\partial z_i^L}{\partial w_{i,j}^L} = \frac{\partial \left(\sum_i (w_{i,j}^L) \sigma_j^{L-1} + b_i^L \right)}{\partial w_{i,j}^L} = \sigma_i^{L-1} \tag{2.26}$$

Thus we have that

$$\frac{\partial \mathfrak{L}}{\partial w_{i,j}^L} = \left(\frac{-y_i^L}{\hat{y}_i^L} + \frac{1 - y_i^L}{1 - \hat{y}_i^L} \right) \cdot \hat{y}_i^L (1 - \hat{y}_i^L) \cdot \sigma_i^{L-1} \tag{2.27}$$

$$= \frac{-y_i^L (1 - \hat{y}_i^L) + \hat{y}_i^L (1 - y_i^L)}{\hat{y}_i^L (1 - \hat{y}_i^L)} \cdot \hat{y}_i^L (1 - \hat{y}_i^L) \cdot \sigma_i^{L-1} \tag{2.28}$$

$$= -y_i^L + y_i^L \hat{y}_i^L) + \hat{y}_i^L - \hat{y}_i^L y_i^L) \cdot \sigma_i^{L-1} \tag{2.29}$$

$$= (\hat{y}_i^L - y_i^L) \cdot \sigma_i^{L-1} \tag{2.30}$$

$$\Rightarrow \Delta_i w_{i,j}^L = \frac{\partial \mathfrak{L}}{\partial w_{i,j}^L} = (\hat{y}_i^L - y_i^L) \cdot \sigma_i^{L-1} \tag{2.31}$$

In the computation of the derivative for the biases of our model, the derivative

$$\frac{\partial z_i^L}{\partial b_i^L} = \frac{\partial \left(\sum_i (w_{i,j}^L) \sigma_j^{L-1} + b_i^L \right)}{\partial b_i^L} = 1 \tag{2.32}$$

Also

$$\frac{\partial \mathfrak{L}}{\partial b_i^L} = \frac{\partial \mathfrak{L}}{\partial z_i^L} \cdot \frac{\partial z_i^L}{\partial b_i^L} \tag{2.33}$$

$$= \frac{\partial \mathfrak{L}}{\partial \hat{y}_i^L} \cdot \frac{\partial \hat{y}_i^L}{\partial z_{i,j}^L} \cdot \frac{\partial z_i^L}{\partial b_i^L} \tag{2.34}$$

Thus we have that

$$\frac{\partial \mathfrak{L}}{\partial b_{i,j}^L} = \left(\frac{-y_i^L}{\hat{y}_i^L} + \frac{1 - y_i^L}{1 - \hat{y}_i^L} \right) \cdot \hat{y}_i^L (1 - \hat{y}_i^L) \cdot 1 \tag{2.35}$$

$$= (\hat{y}_i^L - y_i^L) \cdot 1 \tag{2.36}$$

$$\Rightarrow \Delta_i b_i^L = \frac{\partial \mathfrak{L}}{\partial b_i^L} = (\hat{y}_i^L - y_i^L) \tag{2.37}$$

The backpropagation is also done recursively for the hidden layers and for this layer we suppose the activation function used in its computation is the ReLU activation given by

$$\sigma_i^l = \hat{y}_i^l = \max(0, z_i^l) = \mathbb{1}_{[0,z_i^l]}(z_i^l) \tag{2.38}$$

then we have that the derivative of the activation, σ_i^l with respect to the parameters z_i^l given by

$$\frac{\partial \sigma_i^l}{\partial z_i^l} = \frac{\partial \sigma(z_i^l)}{\partial z_i^l} = \frac{\partial}{\partial z_i^l}\left[\begin{cases} 0 & if\ z_i^l \leq 0 \\ z_i^l & if\ z_i^l > 0 \end{cases}\right] \tag{2.39}$$

$$= \begin{cases} 0 & if\ z_i^l \leq 0 \\ 1 & if\ z_i^l > 0 \end{cases} \tag{2.40}$$

$$= \mathbb{1}_{[0,1]}(z_i^l) \tag{2.41}$$

Thus we compute the derivatives for the weights using the chain rule

$$\frac{\partial \mathfrak{L}}{\partial w_{i,j}^l} = \frac{\partial \mathfrak{L}}{\partial \hat{y}_i^l} \cdot \frac{\partial \hat{y}_i^l}{\partial z_{i,j}^l} \cdot \frac{\partial z_i^l}{\partial w_{i,j}^l} \tag{2.42}$$

$$\tag{2.43}$$

Thus we have that

$$\frac{\partial \mathfrak{L}}{\partial w_{i,j}^l} = \left(\frac{-y_i^l}{\hat{y}_i^l} + \frac{1-y_i^l}{1-\hat{y}_i^l}\right) \cdot \mathbb{1}_{[0,1]}(z_i^l) \cdot \sigma_i^{l-1} \tag{2.44}$$

$$\Rightarrow \Delta_i w_{i,j}^l = \frac{\partial \mathfrak{L}}{\partial w_{i,j}^l} = \left(\frac{-y_i^l}{\hat{y}_i^l} + \frac{1-y_i^l}{1-\hat{y}_i^l}\right) \cdot \mathbb{1}_{[0,1]}(z_i^l) \cdot \sigma_i^{l-1} \tag{2.45}$$

Next for the biases, the derivative

$$\frac{\partial \mathfrak{L}}{\partial b_i^l} = \frac{\partial \mathfrak{L}}{\partial z_i^l} \cdot \frac{\partial z_i^l}{\partial b_i^l} \tag{2.46}$$

$$= \frac{\partial \mathfrak{L}}{\partial \hat{y}_i^l} \cdot \frac{\partial \hat{y}_i^l}{\partial z_{i,j}^l} \cdot \frac{\partial z_i^l}{\partial b_i^l} \tag{2.47}$$

$$= \left(\frac{-y_i^l}{\hat{y}_i^l} + \frac{1-y_i^l}{1-\hat{y}_i^l}\right) \cdot \mathbb{1}_{[0,1]}(z_i^l) \cdot 1 \tag{2.48}$$

$$= \left(\frac{-y_i^l}{\hat{y}_i^l} + \frac{1-y_i^l}{1-\hat{y}_i^l}\right) \cdot \mathbb{1}_{[0,1]}(z_i^l) \tag{2.49}$$

$$\Rightarrow \Delta_i b_i^l = \frac{\partial \mathfrak{L}}{\partial b_i^l} = \left(\frac{-y_i^l}{\hat{y}_i^l} + \frac{1-y_i^l}{1-\hat{y}_i^l}\right) \cdot \mathbb{1}_{[0,1]}(z_i^l) \tag{2.50}$$

In the computation, a little change $\Delta_i w_{i,j}^l$ and $\Delta_i b_i^l$ is being added to neurons parameters, passed back into the layers and is passed through an activation which then forward propagates these changes through the layers of the network making the overall loss to change by $\frac{\partial \mathfrak{L}}{\partial z_i^l} = \Delta_i z_i^l$.

The backpropagation does these parameter update using the Gradient Descent(GD) algorithms, and iterate until convergence. There are different weight update methods which we call the optimizers. To minimize the error the parameters are updated in the opposite direction with the delta rule where:

$$NewWeight = OldWeight - learningrate \quad \times \quad Derivaterate \tag{2.51}$$

$$w_{i,j} = w_{i,j} - \eta \Delta_i w_{i,j} \tag{2.52}$$

$$b_i = b_i - \eta \Delta_i b_i \tag{2.53}$$

for each m training samples, where the constant η is the learning rate which controls how slow or fast the parameters get updated after each iteration.

2.2 Convolutional Neural Network Model

CNN's are biologically inspired deep feedforward , backpropragate neural networks which are sparsely connected to the input layer and are designed to process pixel data of an image in an alternating manner till specific features of the problem image is learned. A standard architecture of the convolutional neural network is comprised of the convolutional layers with a nonlinear activation function, the pooling layers and the fully connected layers.

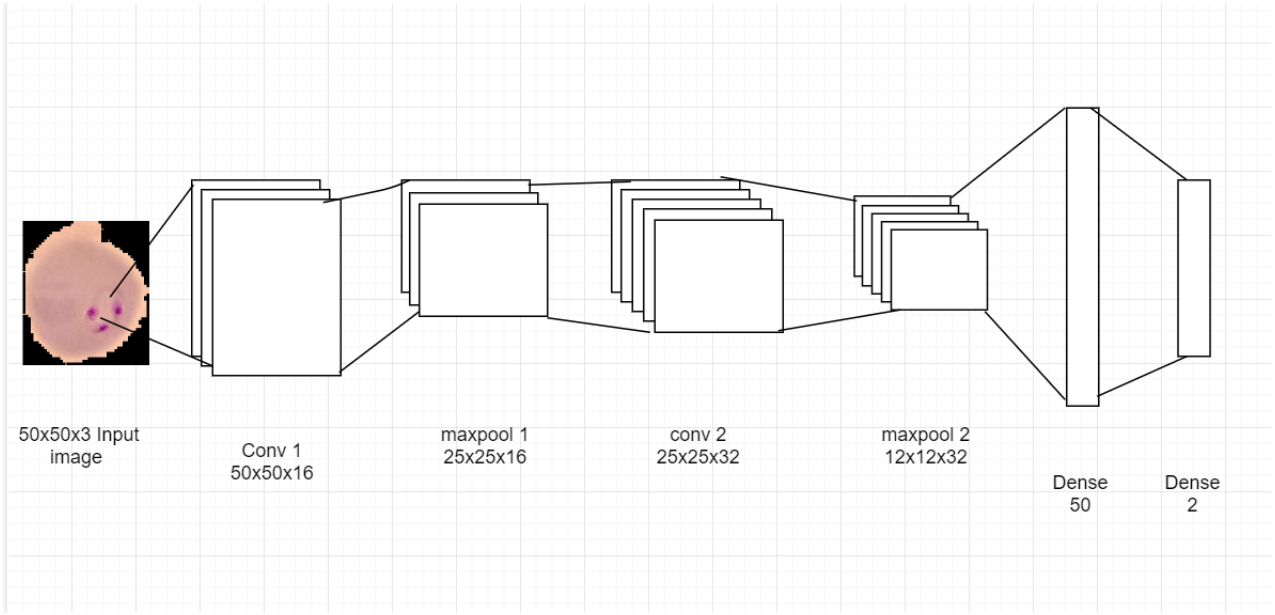


Figure 2.2: CNN Model Architecture

2.2.1 Initializing The Network Parameters

Considering a number of m training image samples, a given input image x which consists of an array of pixel intensities has input dimensions $x \in \mathbb{R}^{h \times w \times d}$ where h is the height, w is the width and d is the number of input channels or depth. We are also given p number of filters, k with dimension $k \in \mathbb{R}^{f \times f \times d \times p}$ and for every filter a bias b having dimensions $b \in \mathbb{R}^p$.

2.2.2 The Convolution Layer + Activation

For the convolution layer, the input image from the sample is convolved with p filters where each filter aims to extract and learn specific features from the image. The filter is passed in equal and finite strides over local regions (receptive fields) in the image and at each location computes the sum of the element-wise dot product between it's element and the elements of the connected receptive field as well as adds the bias per filter. Every computation produces p extracted feature map of the input image as our output for the next layer. Mathematically, the process which generates the output of a convolution layer is given by

$$(x * k)_{ij} = \sum_{m=0}^{f-1} \sum_{n=0}^{f-1} \sum_{q=1}^{d} k(m,n)x(i-m, j-n) + b \tag{2.54}$$

$$= \sum_{m=0}^{f-1} \sum_{n=0}^{f-1} \sum_{q=1}^{d} k_{m,n,l} \cdot x_{i+m,j+n,d} + b \tag{2.55}$$

$$= \sum_{m=0}^{f-1} \sum_{n=0}^{f-1} k_{m,n} \cdot x_{i+m,j+n} + b \quad if \quad \#d = 1 \tag{2.56}$$

Thus we have the compact form of the convolved input vector with the shared weights to be given by

$$z_{i,j}^l = \sum_{m=0}^{f-1} \sum_{n=0}^{f-1} k_{m,n}^l \cdot \sigma_{i+m,j+n}^{l-1} + b_{i,j}^l \tag{2.57}$$

$$= \sum_{m=0}^{f-1} \sum_{n=0}^{f-1} w_{m,n}^l \cdot \sigma_{i+m,j+n}^{l-1} + b_{i,j}^l \tag{2.58}$$

$$\sigma_{i,j}^l = \sigma(z_{i,j}^l) \tag{2.59}$$

where $\sigma_{i,j}^l$ is the activation function applied to the output from the convolution at layer $l = 1, \cdots, L$, $w_{m,n}^l$ is the shared weight matrices which connects the neurons of layer l with the neurons of layer $l-1$ and b^l is the bias of layer l. The activation function is applied on the output of the convolutions to introduce non-linearity into the network. We use the ReLU activation to remove all non negative entries and replace them

with 0. The activated output is then propagated forward as an input for the next layer which is the pooling layer.

2.2.3 The Pooling layer

For the pooling layer, the spatial size of the activated outputs gotten from the previous convolution layer is reduced by reducing the amount of parameters and computation in the network without much loss of significant information. It then transforms it into the desired values which is dependent on the type of pooling method. The two (2) most common types of pooling operation are the

1. MaxPooling Layer : It takes the maximum values from each specified block of the input. The result being propagated forward from the convolution layer is reduced to $(\frac{h}{k} \times \frac{w}{k}) = (h_{\prime} \times w_{\prime})$ dimension, when a pooling kernel of size $(k \times k)$ is slid over the inputs with dimension $(h \times w)$, as the output dimension for the pooling layer.

$$y^l_{i_{\prime},j_{\prime},q_{\prime}} = \max_{0\leq i\leq h, 0\leq i\leq w, 0\leq q\leq d} z^l_{i_{\prime},j_{\prime},q_{\prime}} \tag{2.60}$$

2. Average Layer : It takes the average of the values of the block for each number of kernel(filter).

$$y^l_{i_{\prime},j_{\prime},q_{\prime}} = \frac{1}{hw} \sum_{0\leq i\leq h, 0\leq i\leq w, 0\leq q\leq d} z^l_{i_{\prime},j_{\prime},q_{\prime}} \tag{2.61}$$

with $0 \leq i_{\prime} \leq h_{\prime}, 0 \leq j_{\prime} \leq w_{\prime}, 0 \leq q_{\prime} \leq d_{\prime}$.

2.2.4 The Dense layer and Loss

This is the same as the regular fully connected neural network , the output matrix from the pooling layer is then vectorized (flattened) and sent to a fully connected layer where we use a softmax activation which trains and classifies the images using backpropagation and predicts the label class of the input image just as described explicitly in section 2.1, subsection 2.1.1 - 2.1.2. The loss function is the cross entropy loss in equation 2.8

$$\mathfrak{L} = -\frac{1}{m}\sum_{i=1}^{m} y_i \log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i) \tag{2.8}$$

2.2.5 BackPropagation

Since the dimension of the input is $\mathbb{R}^{h\times w}$ and the weight kernel dimension is $\mathbb{R}^{f\times f}$ then the output of the convolution(feature map) is of dimension $\mathbb{R}^{(h-f+1)\times(w-f+1)}$. The derivative of the loss is denoted by $\frac{\partial \mathfrak{L}}{\partial w^l_{m^{\prime},n^{\prime}}}$ with $\partial w^l_{m^{\prime},n^{\prime}}$ as the a signifier of a slight change in eaxh pixel of the weight kernel.

Thus the derivative with respect to the model parameter is computed using chain rule and we have

$$\frac{\partial \mathfrak{L}}{\partial w^l_{m^{\prime},n^{\prime}}} = \sum_{i=0}^{h-f}\sum_{j=0}^{w-f} \frac{\partial \mathfrak{L}}{\partial z^l_{i,j}} \frac{\partial z^l_{i,j}}{\partial w^l_{m^{\prime},n^{\prime}}} \tag{2.62}$$

$$\tag{2.63}$$

with $z^l_{i,j}$ denoted by

$$z^l_{i,j} = \sum_{i=0}^{h-f}\sum_{j=0}^{w-f} w^l_{i,j}\sigma^{l-1}_{i+m,j+n} + b^l \tag{2.64}$$

and our ReLU activation as

$$\sigma^l_{i,j} = \hat{y}^l_{i,j} = \max(0, z^l_{i,j}) = \mathbb{1}_{[0,z^l_{i,j}]}(z^l_{i,j}) \tag{2.65}$$

$$\tag{2.66}$$

$$\frac{\partial \sigma_{i,j}^{l}}{\partial z_{i,j}^{l}} = \frac{\partial \sigma(z_{i,j}^{l})}{\partial z_{i,j}^{l}} = \frac{\partial}{\partial z_{i,j}^{l}} \left[\begin{cases} 0 & if\ z_{i,j}^{l} \leq 0 \\ z_{i}^{l} & if\ z_{i,j}^{l} > 0 \end{cases} \right] \tag{2.67}$$

$$= \begin{cases} 0 & if\ z_{i,j}^{l} \leq 0 \\ 1 & if\ z_{i,j}^{l} > 0 \end{cases} \tag{2.68}$$

$$= \mathbb{1}_{[0,1]}(z_{i,j}^{l}) \tag{2.69}$$

This implies that the derivatives of the models parameters $z_{i,j}^{l}$ with respect to the weights is

$$\frac{\partial z_{i,j}^{l}}{\partial w_{m',n'}^{l}} = \frac{\partial}{\partial w_{m',n'}^{l}} \left(\sum_{m} \sum_{n} w_{m,n}^{l} \sigma_{i+m,j+n}^{l-1} + b^{l} \right) \tag{2.70}$$

$$= \frac{\partial}{\partial w_{m',n'}^{l}} \left(w_{0,0}^{l} \sigma_{i+0,j+0}^{l-1} + \cdots + w_{m',n'}^{l} \sigma_{i+m',j+n'}^{l-1} + \cdots + b^{l} \right) \tag{2.71}$$

$$= \frac{\partial}{\partial w_{m',n'}^{l}} \left(w_{m',n'}^{l} \sigma_{i+m',j+n'}^{l-1} + b^{l} \right) \tag{2.72}$$

$$= \sigma_{i+m',j+n'}^{l-1} \tag{2.73}$$

Therefore the derivative of the loss with respect to our weight is given by

$$\frac{\partial \mathfrak{L}}{\partial w_{m',n'}^{l}} = \sum_{i=0}^{h-f} \sum_{j=0}^{w-f} \frac{\partial \mathfrak{L}}{\partial z_{i,j}^{l}} \frac{\partial z_{i,j}^{l}}{\partial w_{m',n'}^{l}} \tag{2.74}$$

$$= \sum_{i=0}^{h-f} \sum_{j=0}^{w-f} \frac{\partial \mathfrak{L}}{\partial z_{i,j}^{l}} \sigma_{i+m',j+n'}^{l-1} \tag{2.75}$$

The gradients obtained for the inputs results from the rotation of the weights in the weight (kernel) matrix for which the full convolution is computed where the rotated matrix is multiplied by the gradients of the output with respect to the loss. The new sets of weights are then being updated during the backpropagation process with the convolution operation and the dimension of the new set of outputs is $\mathbb{R}^{(h'-f+1)\times(w'-f+1)}$ using chain rule.The derivative of the overall model parameters with respect to the loss is computed where $l = L-1, L-2, \cdots, 1$ and $z_{i',j'}^{l} = x_{i',j'}^{l}$. Thus,

$$\frac{\partial \mathfrak{L}}{\partial x_{i',j'}^{l}} = \sum_{i=0} \sum_{j=0} \frac{\partial \mathfrak{L}}{\partial x_{i,j}^{l+1}} \frac{\partial x_{i,j}^{l+1}}{\partial x_{i',j'}^{l}} \tag{2.76}$$

$$= \sum_{m=0}^{f-1} \sum_{n=0}^{f-1} \frac{\partial \mathfrak{L}}{\partial x_{i'-m,j'-n}^{l+1}} \frac{\partial x_{i'-m,j'-n}^{l+1}}{\partial x_{i',j'}^{l}} \tag{2.77}$$

$$= \sum_{m=0}^{f-1} \sum_{n=0}^{f-1} \delta_{i'-m,j'-n}^{l+1} \frac{\partial x_{i'-m,j'-n}^{l+1}}{\partial x_{i',j'}^{l}} \tag{2.78}$$

$$= \frac{\partial}{\partial x_{i',j'}^{l}} \left(\sum_{m'} \sum_{n'} w_{m',n'}^{l+1} \sigma_{i'-m+m',j'-n+n'}^{l} + b^{l+1} \right) \tag{2.79}$$

$$= \frac{\partial}{\partial x_{i',j'}^{l}} \left(\sum_{m'} \sum_{n'} w_{m',n'}^{l+1} \sigma \left(x_{i'-m+m',j'-n+n'}^{l} \right) + b^{l+1} \right) \tag{2.80}$$

$$= \frac{\partial}{\partial x_{i',j'}^{l}} \left(w_{m',n'}^{l+1} \sigma \left(x_{0-m+m',0-n+n'}^{l} \right) + \cdots + w_{m,n}^{l+1} \sigma \left(x_{i',j'}^{l} \right) + \cdots + b^{l+1} \right) \tag{2.81}$$

$$= \frac{\partial}{\partial x_{i',j'}^{l}} \left(w_{m,n}^{l+1} \sigma \left(x_{i',j'}^{l} + b^{l+1} \right) \right) \tag{2.82}$$

$$= w_{m,n}^{l+1} \frac{\partial}{\partial x_{i',j'}^{l}} \left(\sigma \left(x_{i',j'}^{l} + b^{l+1} \right) \right) \tag{2.83}$$

$$= w_{m,n}^{l+1} \frac{\partial \sigma_{i',j'}^{l}}{\partial x_{i',j'}^{l}} \tag{2.84}$$

with m and n ranging from $0 \leq m, n \leq f-1$. We see that $\sigma(x^l_{i'-m+m',j'-n+n'}) = \sigma(x^l_{i',j'})$ and $w^{l+1}_{m',n'} = w^{l+1}_{m,n}$ when $m' = m,\ n' = n$

$$\frac{\partial \mathfrak{L}}{\partial x^l_{i',j'}} = \sum_{m=0}^{f-1}\sum_{n=0}^{f-1} \frac{\partial \mathfrak{L}}{\partial x_{i'-m,j'-n}} w^{l+1}_{m,n} \frac{\partial \sigma^l_{i',j'}}{\partial x^l_{i',j'}} \tag{2.85}$$

For the pooling layer no learning takes place since it is just used in dimension reduction and the backpropagation in the fully connected layers take place as in subsection 2.1.3.

In summary, there are two passes in the computation of the gradients used for backpropagation in the convolution layer of a convolutional neural network [15].

1. $\frac{\partial \mathfrak{L}}{\partial w^l_{i,j}} = convolution\ \left(\ input\ x^l_{i,j},\ gradient\ loss\ from\ previous\ layers\ \frac{\partial \mathfrak{L}}{\partial z^l_{i,j}}\right)$

2. $\frac{\partial \mathfrak{L}}{\partial x^l_{m',n'}} =\ full\ \ convolution\ \left(\ rotated\ filter\ \ w^l_{i,j},\ gradient\ loss\ from\ previous\ layers\ \frac{\partial \mathfrak{L}}{\partial w^l_{m',n'}}\right)$

The model parameters are then updated using an optimization algorithm

$$x^l_{i,j} = x^l_{i,j} - \eta \Delta_{i,j} x^l_{i,j} \tag{2.86}$$
$$w^l_{i,j} = w^l_{i,j} - \eta \Delta_{i,j} w^l_{i,j} \tag{2.87}$$
$$b^l_{i,j} = b^l_{i,j} - \eta \Delta_{i,j} b^l_{i,j} \tag{2.88}$$

for each m training samples.

2.2.6 Optimization Algorithms

Optimization algorithms are used in minimizing the error computed by our loss function $\mathfrak{L}$ in order to gradually get the network model step by step towards an optimum solution. This is done by updating the model's parameters with the gradient of our loss function for the training samples with respect to these parameters which is computed using backpropagation. There are several optimizers available in deep learning libraries such as keras, to enable this process.We give a brief explanation of how these optimization algorithms are constructed. Some of these algorithms are:

Stocastic Gradient Descent(SGD)

This is one of the most popular optimization algorithm used in neural network models and it is one of the variant of the GD algorithm. The GD iteratively updates the parameters to minimize the loss. If the gradient of the loss is high, the models learns faster since the update goes in the direction of the steepest descent and as the gradient approaches or is equal to zero the model stops learning. The closer the model gets to the optimum value, it takes learns to take smaller steps which we call the learning rate. The SGD does the evaluation and update of the parameters in the direction of our gradient for each or a subset of the training sample in our dataset with each sample or subset of samples being chosen at random. For each iteration, the updated weights become better than the previous updates if the learning rate is not to high although this is a bit computationally expensive and the convergence might be slow significantly more slowly than other.

Algorithm 1: SGD algorithm

1 for each i in a batch of training sample , the data is shuffled and then:
2 for each k in the shuffled data, the gradient of the loss is computed and then:

$$\phi_i = \phi_i - \eta \frac{\partial \mathfrak{L}}{\partial \phi_i}(\phi_i) \tag{2.89}$$

$$\tag{2.90}$$

SGD with Nestorov + Momentum

The SGD has a number of parameters which can be used to improve the performance of the algorithm such as Momentum and Nesterov. One variation of the SGD is the SGD with Momentum which is used to increase the speed of convergence, it speeds up the gradients with respect to the loss in the right direction. It uses a portion of the information from a recent previous update to determine the next update. Let v_t be the current update, v_{t-1} as the previous update and β as the portion taken. Thus we have that

Algorithm 2: SGD + Momentum algorithm

1 v_t, v_{t-1}, ϕ are our parameters and $\beta \in [0, 1]$ is a hyperparameter.t is the time step.

$$v_t = \beta v_{t-1} + \eta \frac{\partial \mathfrak{L}}{\partial \phi_i}(\phi_i) \tag{2.91}$$

$$\phi_{i,j} = \phi_i - v_t \tag{2.92}$$

$$\tag{2.93}$$

The SGD also uses another parameter called the Nestorov Accelarated Gradient (NAG). Nesterov momentum is a simple change to normal momentum. It takes a decayed average of past steps and steps in that direction first. Then we compute the gradient from that new position using our data, performing a correction. We thus update the weights twice on each iteration: once using momentum and using our gradient algorithm.

Algorithm 3: SGD Nesterov Momentum

1

$$v_t = \beta v_{t-1} + \eta \frac{\partial \mathfrak{L}}{\partial \phi_i}(\phi_i - \beta v_{t-1}) \tag{2.94}$$

$$\phi_i = \phi_i - v_t \tag{2.95}$$

$$\tag{2.96}$$

Adagrad

Adagrad is an algorithm which implements the gradient descent with a variable learning rate, with parameters which are used less given large gradients and the common ones given small gradients. It is an improvement on the SGD which issues a different and suitable learning rate for each parameter at every time step, t

Algorithm 4: Adagrad Algorithm

1 $\frac{\partial \mathfrak{L}}{\partial \phi_i}(\phi_{t,i})$ as the current gradient of the loss function at each time step

2 For each t, it updates for every parameter $\phi_{i,j}$ to give:

3

$$\phi_{t+1,i} = \phi_{t,i} - \eta \frac{\partial \mathfrak{L}}{\partial \phi_i}(\phi_{t,i}) \tag{2.97}$$

$$\tag{2.98}$$

4 As an update rule, the updates for every parameter is computed based on previous gradients

5

$$\phi_{t+1,i} = \phi_{t,i} - \frac{\eta}{\sqrt{G_{t,ii}} + \epsilon} \frac{\partial \mathfrak{L}}{\partial \phi_i}(\phi_{t,i}) \tag{2.99}$$

$$\tag{2.100}$$

6 with $G_{t,ii} \in \mathbb{R}^{d \times d}$ is the diagonal matrix containing diagonal elements which are the sum of squares of all previous gradients

7 ϵ is a regularization term which removes zero division thus provides numerical stability

Adadelta

Adadelta is an adaptation of Adagrad that uses momentum methods to correct the monotonically decreasing learning rate problem. It doesn't accumulate previous squared gradients but rather restricts it to a fixed size and it recursively defines the sum of all gradients as a decaying mean of all the previous squared gradients. The moving average $E[\left[\frac{\partial \mathfrak{L}}{\partial \phi_i}\right]^2]_t$, which makes sure its only the previous mean and the current information of gradient that is accounted for at t:

Algorithm 5: AdaDelta Algorithm

1 $E\left[\left[\frac{\partial \mathfrak{L}}{\partial \phi_i}\right]^2\right]_t$, β - the momentum term.

$$E\left[\left[\frac{\partial \mathfrak{L}}{\partial \phi_i}\right]^2\right]_t = \beta E\left[\left[\frac{\partial \mathfrak{L}}{\partial \phi_i}\right]^2\right]_{t-1} + (1-\beta)\left[\frac{\partial \mathfrak{L}}{\partial \phi_i}\right]^2_t \tag{2.101}$$

$$\phi_{t+1,i} = \phi_{t,i} - \eta \frac{\partial \mathfrak{L}}{\partial \phi_i}(\phi_{t,i}) \tag{2.102}$$

2 and lets have the update in simple terms with the learning rate to be divided as such

3

$$\Delta\phi_t = -\frac{\eta}{\sqrt{E\left[\left[\frac{\partial \mathfrak{L}}{\partial \phi_i}\right]^2\right]_t + \epsilon}} \frac{\partial \mathfrak{L}}{\partial \phi_i}(\phi_{t,i}) \tag{2.103}$$

4 The squared of the parameters updates are then taken and we have the Adadelta update rule as

$$E\left[\Delta\phi^2\right]_t = \beta E[\Delta\phi^2]_{t-1} + (1-\beta)\Delta\phi_t^2 \tag{2.104}$$

$$\Delta\phi_t = -\frac{\left[\sqrt{E\left[\Delta\phi^2\right]_t + \epsilon}\right]_{t-1}}{\left[\sqrt{E\left[\frac{\partial \mathfrak{L}}{\partial \phi_i}(\phi_{t,i})\right]_t + \epsilon}\right]_t} \frac{\partial \mathfrak{L}}{\partial \phi_i}(\phi_{t,i}) \tag{2.105}$$

$$\phi_{t+1} = \phi_t + \Delta\phi_t$$

Rmsprop

RMSprop is an adaptive learning rate optimization algorithm which tries to correct the issue of varying magnitudes of gradients. The learning rate updates individually over time as the learning speeds up in the right direction. It gets the mean of the squared gradients for each parameter and then divides the learning rate by the root mean square.

Algorithm 6: RMSprop Algorithm

1 $E\left[\left[\frac{\partial \mathfrak{L}}{\partial \phi_i}\right]^2\right]_t$, β - the momentum term.

2 The RMSprop update rule as

$$E\left[\left[\frac{\partial \mathfrak{L}}{\partial \phi_i}\right]^2\right]_t = \beta E\left[\left[\frac{\partial \mathfrak{L}}{\partial \phi_i}\right]^2\right]_{t-1} + (1-\beta)\frac{\partial \mathfrak{L}}{\partial \phi_i}(\phi_{t,i}) \tag{2.106}$$

$$\phi_{t+1} = \phi_t - \frac{\eta}{\sqrt{E\left[\left[\frac{\partial \mathfrak{L}}{\partial \phi_i}\right]^2\right]_t + \epsilon}} \frac{\partial \mathfrak{L}}{\partial \phi_i}(\phi_{t,i}) \tag{2.107}$$

with $\beta = 0.9$ and a good learning rate set to 0.001

Adam

The Adaptive Moment Estimation (Adam) is an optimization algorithm which stores the estimates of the mean, m_t and the variance, v_t which are exponentially decaying averages of the past squared gradients

Algorithm 7: Adam Algorithm

The estimates are computed as

$$m_t = \beta_1 m_{t-1} + (1-\beta_1)\left[\frac{\partial \mathfrak{L}}{\partial \phi_i}(\phi_{t,i})\right] \tag{2.108}$$

$$v_t = \beta_1 v_{t-1} + (1-\beta_2)\left[\frac{\partial \mathfrak{L}}{\partial \phi_i}(\phi_{t,i})\right]^2 \tag{2.109}$$

For previous time steps they are biased towards zero if the decay rates, $\beta_i,\ i = 1, 2$ are small

These biases are corrected and the new estimates are

$$\begin{aligned} \hat{m}_t &= \frac{m_t}{1-\beta_1^t} \\ \hat{v}_t &= \frac{v_t}{1-\beta_2^t} \end{aligned} \tag{2.110}$$

Thus the update rule for the paramaters using adam is

$$\phi_{t+1} = \phi_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon}\hat{m}_t \tag{2.111}$$

Adam is a combination of RMSprop and momentum. RMSprop contributes the exponentially decaying average of past squared gradients v_t, while momentum accounts for the exponentially decaying average of past gradients ,m_t.

AdaMax

The AdaMax uses the infinity norm as opposed to how in algorithms like the Adam, the estimate v_t scales the gradient inversely proportionally to the ℓ_2 norm of the previous gradients and the current squared gradient

Algorithm 8: Adamax Algorithm

This updates can be generalized using the ℓ_p norms as such

$$v_t = \beta_1^p v_{t-1} + (1-\beta_2^p)\left[\frac{\partial \mathfrak{L}}{\partial \phi_i}(\phi_{t,i})\right]^p \tag{2.112}$$

But most norms greater than 2 or large p values are numerically unstable, so the Adamax was implemented using ℓ_∞ norm for v_t as they are know to converge in a stable value with u_t denoting the ∞ norm-constrained v_t:

$$u_t = \beta_2^\infty v_{t-1} + (1-\beta_2^\infty)\left|\frac{\partial \mathfrak{L}}{\partial \phi_i}(\phi_{t,i})\right|^\infty \tag{2.113}$$

$$= \max\left(\beta_2 \cdot v_{t-1}, \left|\frac{\partial \mathfrak{L}}{\partial \phi_i}(\phi_{t,i})\right|\right) \tag{2.114}$$

Thus the update rule for the paramaters using Adamax is

$$\phi_{t+1} = \phi_t - \frac{\eta}{\hat{u}_t}\hat{m}_t \tag{2.115}$$

Nadam

Nesterov-accelerated Adaptive Moment Estimation (Nadam) is a variation of Adam which has Nesterov momentum.

Algorithm 9: Nadam Algorithm

---

1 The estimates are computed as

$$m_t = \beta_1 m_{t-1} + (1-\beta_1)\left[\frac{\partial \mathfrak{L}}{\partial \phi_i}(\phi_{t,i})\right] \tag{2.116}$$

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t} \tag{2.117}$$

$$\tag{2.118}$$

Thus the update rule for the paramaters using adam is

2

$$\phi_{t+1} = \phi_t - \frac{\eta}{\sqrt{\hat{v}_t}+\epsilon}\hat{m}_t \tag{2.119}$$

3 which can be expanded as

$$\phi_{t+1} = \phi_t - \frac{\eta}{\sqrt{\hat{v}_t}+\epsilon}\left(\frac{\beta_1 m_{t-1}}{1-\beta_1^t} + \frac{(1-\beta_1)\left[\frac{\partial \mathfrak{L}}{\partial \phi_i}(\phi_{t,i})\right]}{1-\beta_1^t}\right) \tag{2.120}$$

with $\beta_1 \hat{m}_{t-1} = \frac{\beta_1 m_{t-1}}{1-\beta_1^t}$ as the bias-corrected estimate of the momentum vector of the previous time step.Thus

$$\phi_{t+1} = \phi_t - \frac{\eta}{\sqrt{\hat{v}_t}+\epsilon}\left(\beta_1 \hat{m}_{t-1} + \frac{(1-\beta_1)\left(\frac{\partial \mathfrak{L}}{\partial \phi_i}(\phi_{t,i})\right)}{1-\beta_1^t}\right) \tag{2.121}$$

4 Thus the nestorov momentum is added by replacing the bias-corrected estimate of the momentum vector of the previous time step $\hat{m}_{t-1}$ with the bias-corrected estimate of the current momentum vector $\hat{m}_t$.

5 The update rule for the paramaters using Nadam is

6

$$\phi_{t+1} = \phi_t - \frac{\eta}{\sqrt{\hat{v}_t}+\epsilon}\left(\beta_1 \hat{m}_t + \frac{(1-\beta_1)\left(\frac{\partial \mathfrak{L}}{\partial \phi_i}(\phi_{t,i})\right)}{1-\beta_1^t}\right) \tag{2.122}$$

Chapter 3

Application on Selected Kaggle Datasets

For both datasets, the data were pre-processed in the similar manner. Both datasets had the images for different classes in different folders, then every image was read in and resized using Opencv which is used in computer vision as an imaging library. Then the arrays of images from all folders where passed into a list after target labels has been associated with each image and appended to a target list. The Numpy library which is a numerical library was then used to save these list as a ".npy" file which is saved in the local memory so it can be loaded anytime and from anywhere still using numpy without necessarily repeating the process of reading in so many images over and over.

The neural network models for the medical datasets were implemented on kaggle using the Keras framework with the data being split into training (80%) and testing (20%) data. Some parts of the train set was also used for validation as well. This is then converted to float and normalized, the target train and test sets were encoded since we have multiple classes. We implement the FCN and CNN models then make comparisons between both models and their performance.

3.1 Analysis for the Malaria dataset

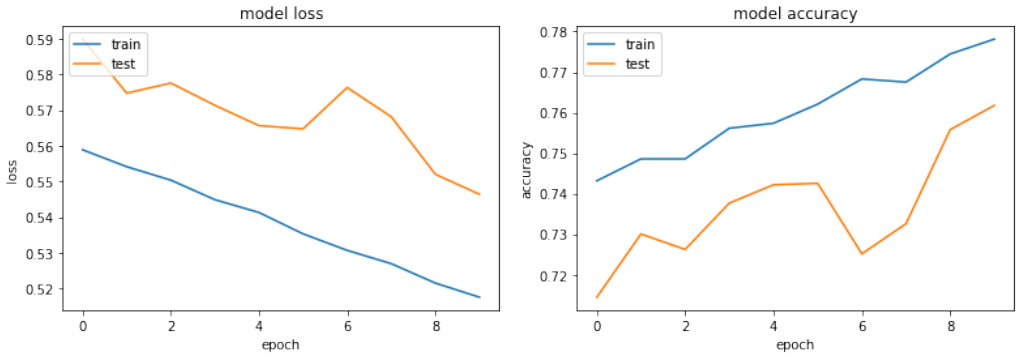
For the FCN, 2-Layer NN with 1 hidden layer was implemented on the Malaria dataset. The train set consisted of 22046 training samples and validated on 5512 samples which contained two(2) classes, the parasitized and the uninfected class.

The train samples for our problem has an input shape of 50 by 50 by 3 since the image is colored and has 3 channels. The hidden layer has 32 neurons with a relu activation function and a softmax activation in the last layer to compute the probabilities of each class for our predicted output. It was trained using categorical cross entropy loss and the SGD as our optimzer to enable a reduction of our loss function by estimating it repeatedly and intuitively adjusting as fit. The training was performed for 10 epochs (number of iterations) and the validation set is evaluated at the end of each epoch with a learning rate of $\eta = 0.01$. The results obtained are tabulated and the accuracy and loss for FCN is shown in table 3.1.

Table 3.1: FCN Loss and Accuracy after 10 epochs for the Malaria Dataset

Epochs	Train		Valid			
	Loss	mae	Acc	Val-loss	Val-mae	Val-acc
1	0.5589	0.3658	0.7432	0.5899	0.3768	0.7144
2	0.5541	0.3624	0.7486	0.5747	0.3659	0.7300
3	0.5504	0.3592	0.7486	0.5775	0.3651	0.7262
4	0.5449	0.3557	0.7561	0.5713	0.3734	0.7377
5	0.5413	0.3536	0.7574	0.5656	0.3638	0.7422
6	0.5353	0.3494	0.7621	0.5647	0.3573	0.7426
7	0.5307	0.3463	0.7683	0.5763	0.3675	0.7251
8	0.5270	0.3437	0.7675	0.5681	0.3594	0.7326
9	0.5215	0.3404	0.7745	0.5520	0.3501	0.7558
10	0.5176	0.3370	0.7781	0.5464	0.3496	0.7618

Due to the stochastic nature of our training algorithm, the results seem to vary but after several training we see that the loss is steadily decreasing for the train set with the loss after 10 epochs to be 0.5176 and the accuracy increasing and after 10 epochs it is 0.7781. But we can see that the FCN didn't perform so well on test set as well as it did on the training set. The test set has a loss of 0.5464 after 10 epochs and the accuracy after 10 epochs is 0.7618.The graphs for the evaluation is shown below in figure 3.1



(a) Loss graph for Malaria Dataset (b) Accuracy graph for Malaria Dataset

Figure 3.1: Visualizing the Loss and Accuracy for the FCN Malaria Dataset Model

We first construct a simple CNN with the intention of implementing a much deeper neural network after understanding what makes the model perform better. Thus, in this regards we construct a CNN with two(2) convolution layers for which the outputs are passed through a relu activation function, two maxpool layers and two dense layers. Each train sample maintains an input shape of $(h \times w \times d) = (50 \times 50 \times 3)$. The input into our first convolution layer is then convolved with our kernel(filters) having a size of $(f \times f) = (2 \times 2)$ and 16 filters. These output from our first convolution which we call our feature maps are then downsampled by passing it through the first maxpooling layer which has kernel size 2. These downsampled output with a new input shape of $(25 \times 25 \times 16)$ and passed to the second convolutional layer with 32 filters and a relu activation. It is again passed through maxpool which reduces the dimension to $(12 \times 12 \times 32)$, this is then flattened into a vectorized form of size 4608 in order to succesfully pass it into the dense layers.

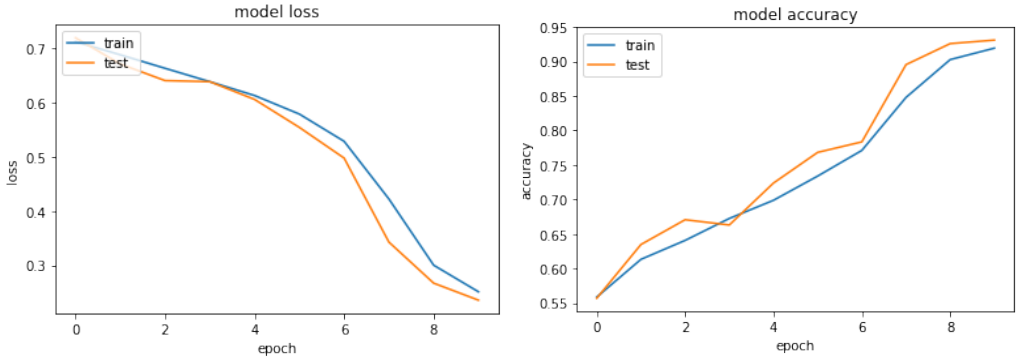
For the dense layer, it has one(1) hidden layer has 50 neurons with a relu activation function and a softmax activation. It was trained using categorical cross entropy loss and the SGD as our optimzer as was done in FCN. The training was performed for 10 epochs and the validation set is evaluated at the end of each epoch with a learning rate of $\eta = 0.01$. The results obtained are tabulated and the accuracy and loss for CNN is shown in table 3.2.

Table 3.2: CNN Loss and Accuracy after 10 epochs for the Malaria Dataset

Epochs	Train		Valid			
	Loss	mae	Acc	Val-loss	Val-mae	Val-acc
1	0.7129	0.4908	0.5586	0.7200	0.4829	0.5570
2	0.6886	0.4692	0.6133	0.6716	0.4556	0.6348
3	0.6637	0.4462	0.6408	0.6410	0.4350	0.6707
4	0.6390	0.4242	0.6725	0.6390	0.4132	0.6631
5	0.6134	0.4043	0.6989	0.6062	0.4049	0.7239
6	0.5792	0.3792	0.7340	0.5546	0.3752	0.7683
7	0.5289	0.3465	0.7712	0.4979	0.3264	0.7836
8	0.4224	0.2774	0.8480	0.3428	0.2244	0.8955
9	0.3002	0.1866	0.9028	0.2670	0.1718	0.9260
10	0.2511	0.1441	0.9194	0.2356	0.1370	0.9311

After several training as well we see that the loss is quickly decreased for the train set with the loss after 10 epochs to be 0.2511 and the accuracy rapidly increased after 10 epochs it was 0.9194. Although on the test

set it seemed to perform better than the train set with a loss of 0.2356 after 10 epochs and the accuracy after 10 epochs is 0.9311. The graphs for the evaluation is shown below in figure 3.2



(a) Loss graph for Malaria Dataset

(b) Accuracy graph for Malaria Dataset

Figure 3.2: Visualizing the Loss and Accuracy for the CNN Malaria Dataset Model

We now review the performance of our CNN model, this can be seen when we get our confusion matrix and the classification report. The confusion matrix shows us in detail the predictions made by our model on our test data and also shows the number of misclassified data samples which helps us adjust our model to perform better as desired. In the confusion matrix, the true positive (TP) is the result gotten when the model correctly predicts the positive class , the true negative (TN) is the result gotten when the model correctly predicts the negative class., the false positive (FP) is the result gotten when the model incorrectly predicts the positive class and the false negative (FN) is the result gotten when the model incorrectly predicts the negative class.

The classification report describes the actual counts for each class in the test data, some of the metrics of evaluations are recall, precision, and the f1-score. The accuracy of the model is computed by taking the ratio of all the correct predictions (i.e TP+FP) to total number predictions (i.e TP+FP+TN+FN). The precision is the accuracy of the positive class predictions which is computed by taking the ratio of TP to the sum of TP and FP, while the recall or the true positive rate is the fraction of positive class that were correctly identified which is computed by taking the ratio of TP to the sum of TP and FN. The f1-score is used to penalize models which have high accuracy for a class but low recall for that class and can be computed by taking the harmonic mean of the precision and recall ($\frac{2\times(precision\times recall)}{precision+recall}$).

As seen in table 3.3, the model predicted 2586 samples as parastisized and 2926 samples as the unin-

Table 3.3: Confusion Matrix and Classification Report for Malaria

Malaria Classes	0. Parasitized	1. Uninfected	recall	total(support)
0. Parasitized	2470	264	0.90	2734
1. Uninfected	116	2662	0.96	2778
Precision	0.96	0.91	-	-
f1-score	0.93	0.93	-	-
total	2586	2926	-	5512

fected class. For the first class there was a total of 2734 samples which were are parasitized and the model was succesful in identifying 2470 out of the predicted as parasitized but misclassified 264 as Uninfected. It also correctly classified 2662 out of the 2778 uninfected class as uninfected and misclassified 166 as parasitized. The recall for the parasitized is $\frac{2470}{2734} = 0.90$ with precision as $\frac{2470}{2586} = 0.96$ which is good as it gives a good fi-score of 0.93. The recall for the uninfected is 0.96 with precision as 0.91 which gives a good f1-score of 0.93.

The Receiver Operating Characteristic (ROC) Curve determines the model performance by plotting the the proportion of actual positive cases, which got predicted correctly against the proportion of actual negative

cases, which got predicted correctly.

It is used to evaluated the ability of the model to differentiate between classes while accessing the performance for each class. The model performed well for both the parasitized and uninfected class.

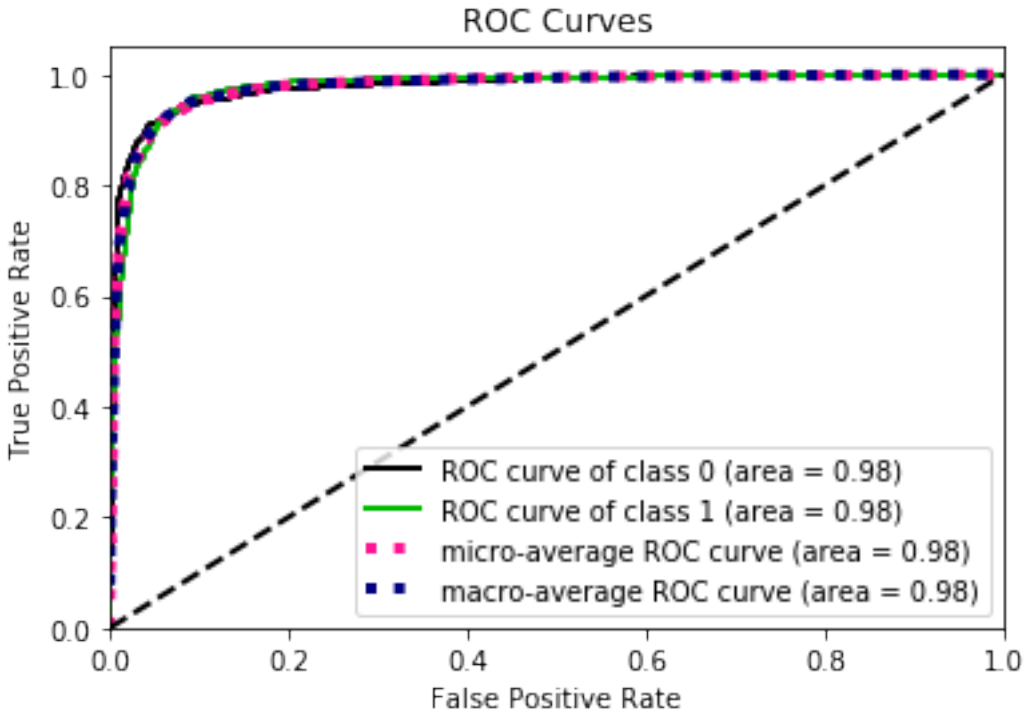


Figure 3.3: ROC Curve

3.2 Analysis for the Kvasir dataset

Similarly, a 2-Layer FCN with 1 hidden layer was implemented on the Kvasir dataset which contains endoscopic images of the human GI tract. The train set consisted of 8000 training samples and validated on 1600 samples which contained eight(8) classes,the dyed and lifted polyp, dyed resection margins, esophagitis, normal cecum, normal pylorus, normal z-line, polyps and ulcerative colitis classes.

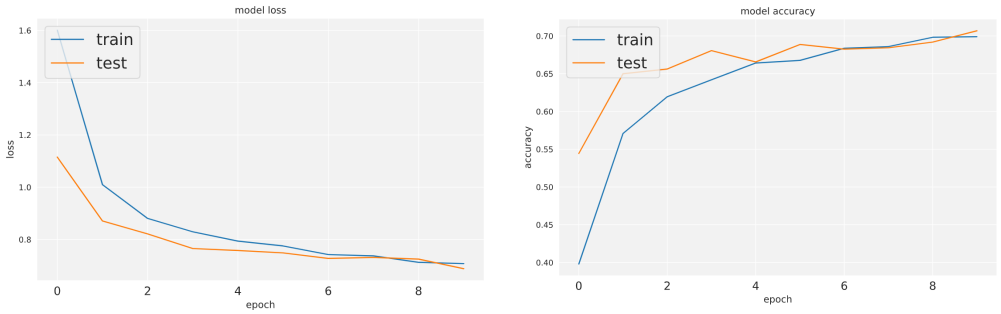
The train samples has an input shape of $(50 \times 50 \times 3)$, the hidden layer has 32 neurons with a relu activation function and a softmax activation in the last layer. It was also trained using categorical cross entropy loss and the optimization algorithm as SGD. The training was performed for 10 epochs and the validation set is evaluated at the end of each epoch with a learning rate of $\eta = 0.01$. The results obtained are tabulated and the accuracy and loss for FCN is shown in table 3.4.

The loss for the train set after 10 epochs to be 0.6130 and the accuracy after 10 epochs it is 0.7384. The FCN didn't perform so well on test set as well as it did on the training set. The test set has a loss of 0.6602 after 10 epochs and the accuracy after 10 epochs is 0.7050.The graphs of the accuracy and loss is shown below in figure 3.4

We use a similar architecture for our CNN model as in section 3.1. Each train sample maintains an input shape of $(h \times w \times d) = (50 \times 50 \times 3)$. The input into our first convolution layer is then convolved with our kernel(filters) having a size of $(f \times f) = (2 \times 2)$ and 16 filters. These output from our first convolution which we call our feature maps are then downsampled by passing it through the first maxpooling layer which has kernel size 2. These downsampled output with a new input shape of $(25 \times 25 \times 16)$ and passed to the second convolutional layer with 32 filters and a relu activation. It is again passed through maxpool which reduces the dimension to $(12 \times 12 \times 32)$, this is then flattened into a vectorized form of size 4608 in order to succesfully pass

Table 3.4: FCN Loss and Accuracy after 10 epochs for the kvasir Dataset

Epochs	Train		Valid			
	Loss	mae	Acc	Val-loss	Val-mae	Val-acc
1	1.1345	0.1497	0.5289	0.9120	0.1293	0.6000
2	0.8103	0.1200	0.6442	0.7845	0.1163	0.6569
3	0.7624	0.1132	0.6677	0.7658	0.1100	0.6488
4	0.7184	0.1072	0.6892	0.7394	0.1089	0.6700
5	0.6962	0.1039	0.6953	0.6790	0.1042	0.7069
6	0.6746	0.1013	0.7027	0.6764	0.1029	0.7137
7	0.6530	0.0956	0.7189	0.6826	0.1018	0.6981
8	0.6340	0.0959	0.7267	0.6631	0.0974	0.6931
9	0.6253	0.0946	0.7256	0.6602	0.1000	0.7031
10	0.6130	0.0928	0.7384	0.6602	0.0958	0.7050



(a) Loss graph

(b) Accuracy graph

Figure 3.4: Visualizing the Loss and Accuracy for the FCN Kvasir Dataset Model

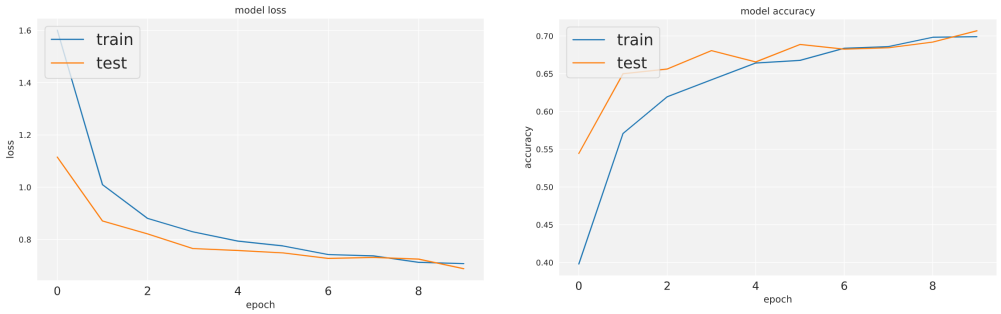
it into the dense layers.

For the dense layer, it has a hidden layer has 50 neurons with a relu activation function and a softmax activation. It was trained using categorical cross entropy loss and the SGD as our optimzer as was done in FCN. The training was performed for 10 epochs and the validation set is evaluated at the end of each epoch with a learning rate of $\eta = 0.01$. The results obtained are tabulated and the accuracy and loss for CNN is shown in table 3.5.

Table 3.5: CNN Loss and Accuracy after 10 epochs for the kvasir Dataset

Epochs	Train		Valid			
	Loss	mae	Acc	Val-loss	Val-mae	Val-acc
1	1.6812	0.1958	0.3698	1.2177	0.1665	0.5194
2	1.0435	0.1434	0.5766	0.9751	0.1279	0.5687
3	0.8837	0.1237	0.6255	0.9004	0.1213	0.5687
4	0.8200	0.1155	0.6533	0.7668	0.1111	0.6775
5	0.7774	0.1103	0.6687	0.7567	0.1088	0.6863
6	0.7611	0.1083	0.6727	0.7733	0.1072	0.6700
7	0.7349	0.1047	0.6883	0.7284	0.1067	0.7044
8	0.7188	0.1027	0.6981	0.7443	0.1025	0.6813
9	0.7090	0.1015	0.6975	0.7338	0.1002	0.6906
10	0.6942	0.0996	0.7108	0.6975	0.0993	0.7006

The loss for the train set quickly decreased with the loss after 10 epochs to be 0.0.6942 and the accuracy rapidly increased after 10 epochs it was 0.7108. Although on the train set it seemed to perform better than the test set with a loss of 0.6975 after 10 epochs and the accuracy after 10 epochs is 0.7006.The graphs of the accuracy and loss is shown below in figure 3.5



(a) Loss graph (b) Accuracy graph

Figure 3.5: Visualizing the Loss and Accuracy for the CNN Kvasir Dataset Model

We also compute the confusion matrix and the classification report based on how the CNN performed on the test data once training was done as seen in table 3.6. The model predicted 161 samples as dyed-lifted polyps, 212 samples as the dyed-resection margins, 134 samples as the esophagitis, 240 samples as the normal cecum, 232 samples as the normal pylorus, 261 samples as the normal z-line, 215 samples as the polyps and 145 samples as the ulcerative colitis.

For the dyed-lifted polyps class, there was a total of 199 samples which were actually are dyed-lifted polyps and the model was successful in identifying 105 out of the predicted as dyed-lifted polyps but misclassified 80 as dyed-resection margins, 1 as normal cecum and 13 as polyps. There was a total of 189 samples which were actually are dyed-resection margins and the model was successful in identifying 130 out of the predicted as dyed-resection margins but misclassified 53 as dyed-lifted polyps, 6 as polyps and 1 as ulcerative colitis. There was a total of 197 samples which were actually are esophagitis and the model was successful in identifying 108 out of the predicted as esophagitis but misclassified 1 as dyed-resection margins, 6 as normal pylorus and 82 as normal cecum.There was a total of 204 samples which were actually are normal cecum and the model was successful in identifying 180 out of the predicted as normal cecum but misclassified 80 as dyed-resection margins, 17 as polyps and 7 as ulcerative colitis.

Table 3.6: Confusion Matrix and Classification Report for Kvasir dataset

Endoscopic Classes	0.	1.	2.	3.	4.	5.	6.	7.	recall	total
0. Dyed-lifted polyps	105	80	0	1	0	0	13	0	0.53	199
1. Dyed-resection margins	53	130	0	0	0	0	5	1	0.69	189
2. Esophagitis	0	1	108	0	6	82	0	0	0.55	197
3. Normal Cecum	0	0	0	180	0	0	17	7	0.88	204
4. Normal Pylorus	0	0	1	0	195	5	2	0	0.96	203
5. Normal Z-line	0	0	25	0	22	174	0	1	0.78	222
6. Polyps	2	0	0	29	6	0	125	22	0.68	184
7. Ulcerative colitis	1	1	0	30	3	0	53	114	0.56	202
Precision	0.65	0.61	0.81	0.75	0.84	0.67	0.58	0.79	-	-
f1-score	0.58	0.65	0.65	0.81	0.90	0.72	0.63	0.66	-	-
total	161	212	134	240	232	261	215	145	-	1600

There was a total of 203 samples which were actually are normal pylorus and the model was successful in identifying 195 out of the predicted as normal pylorus but misclassified 1 as dyed-resection margins, 5 as normal z-line and 2 as polyps.There was a total of 222 samples which were actually are normal z-line and the model was successful in identifying 174 out of the predicted as normal z-line but misclassified 25 as esophagitis, 22 as normal pylorus and 1 as ulcerative colitis. There was a total of 184 samples which were actually are polyps and the model was successful in identifying 125 out of the predicted as polyps but misclassified 2 as dyed-lifted polyps, 29 as normal cecum,29 as normal pylorus and 22 as ulcerative colitis. There was a total of 202 samples

which were actually are ulcerative colitis and the model was successful in identifying 114 out of the predicted as ulcerative colitis but misclassified 1 as dyed-lifted polyps,1 as dyed-resection margins, 30 as normal cecum,3 as normal pylorus and 53 as polyps.

The worst recall is the dyed-lifted polyps as 0.53 with precision as 0.65 and f1-score of 0.58. The best recall is the normal pylorus as 0.96 with precision as 0.84 and f1-score of 0.90. The model had a hard time differentiating the esophagitis class from three other classes as it has a recall of 0.55 but a good precision of 0.81, hence the f1-score penalizes the model and has a value of 0.65.

Also we visualize the ROC curve for the 8 classes which is used to evaluate the ability of the model to differentiate between classes, after several training the model was able to properly differentiate between the classes.

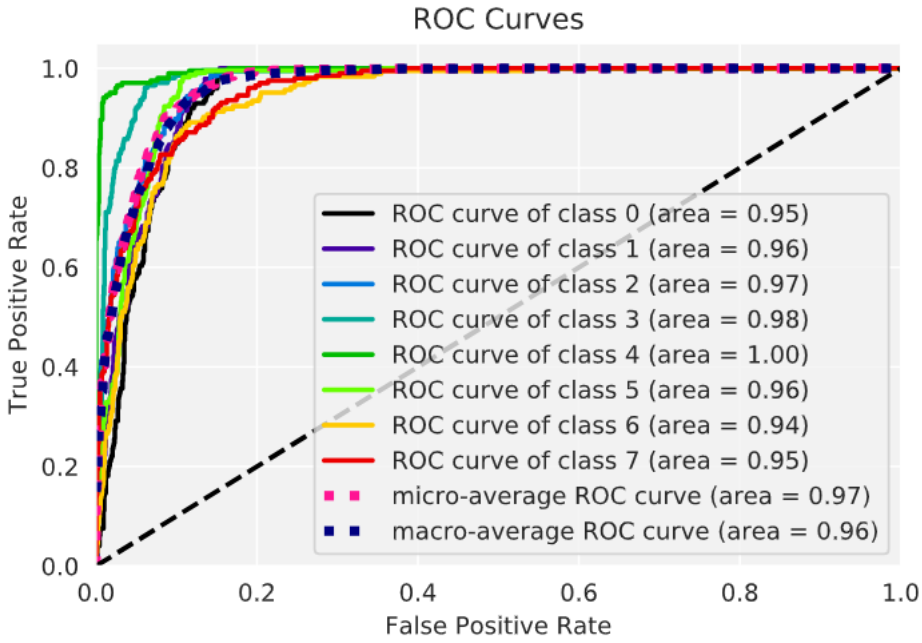


Figure 3.6: ROC Curve

Chapter 4

Hyperparameter Optimization and Optimization Algorithms Analysis

4.1 Results from Hyperparameter Optimization

Hyperparameters are the variables which determines the network structure, determine how the network is trained and are set before the model algorithm begins its learning. There are several hyperparameters which can be tune to enable the neural network function better such as the learning rate η, epochs (number of iterations), regularization parameters, numbers of neurons, number of layers and much more. We analyze for a few of these hyperparameters and how they affect the accuracy of these models. These results were obtained using the python environment in Kaggle kernel with the keras library.

Firstly, we implemented several variations of the models by adjusting the hyperparameters of our model in order to find the appropriate parameters needed to build a better and deeper model. We begin with the FCN and try to tweak it by either increasing the epochs , increasing the number of nodes. We denote the FCN with 2-layer(i.e an input layer, 1 hidden layer and an output layer) as FCN2, FCN with 3-layer (i.e 2 hidden layers) as FCN3 and FCN with with 4-layer (i.e 3 hidden layers) as FCN4. In some cases we add the regularization parameter which helps in reducing overfitting, overfitting is a problem where the model performs well on the train set but when used in evaluating new samples from the test data it doesn't perform well enough. Thus techniques like regularization, batch normalization, earling stopping can be used in tackling this problem.

We use the $L1$ and $L2$ regularization in this work, The L1 regularization adds a penalty equal to the sum of the absolute value of the coefficients and can also shrink some parameters to zero and its given by

$$\mathfrak{L}_{L1} = \mathfrak{L} + \lambda \sum_{i=0}^{m} |z_i| \quad \lambda,\ z_i \textit{ are the parameters.} \tag{4.1}$$

whereas the L2 regularization adds a penalty equal to the sum of the squared value of the coefficients and will force the parameters to be relatively small, the bigger the penalization, the smaller (and the more robust) the coefficients are. It is given by

$$\mathfrak{L}_{L2} = \mathfrak{L} + \lambda \sum_{i=0}^{m} |z_i|^2 \quad \lambda,\ z_i \textit{ are the parameters.} \tag{4.2}$$

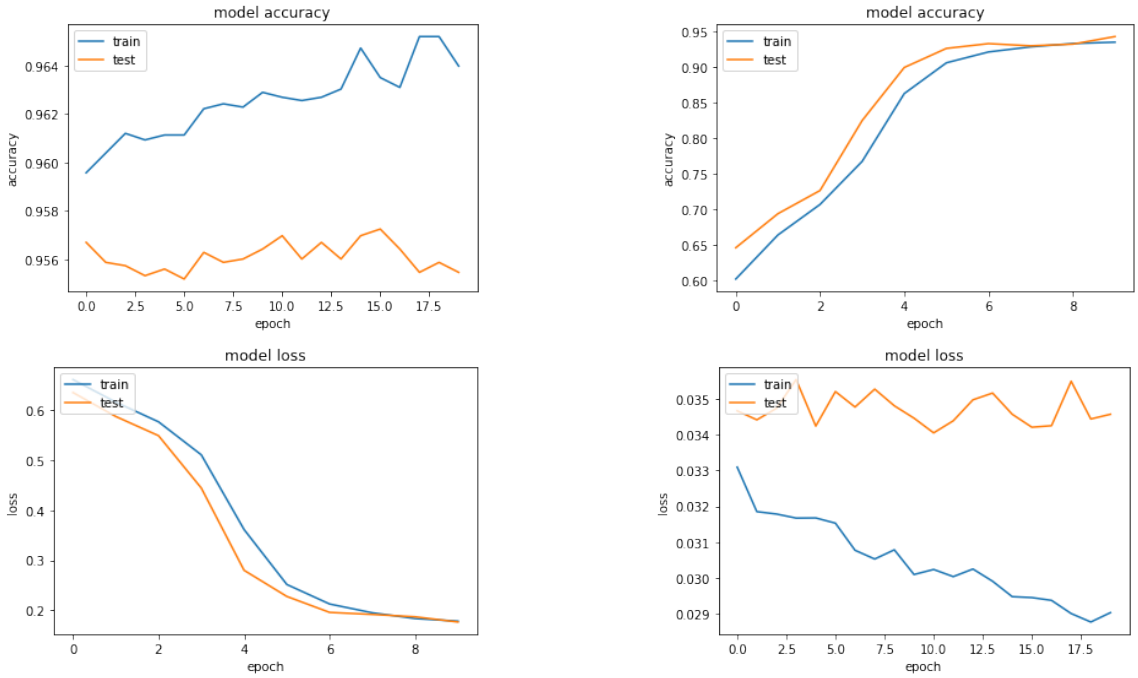
where λ is the regularization parameter.

For the malaria dataset,a performance comparison showed that with an increase in the layers of the FCN there was a noticable increase in the accuracy of the model with FCN4 having an accuracy of the 0.7732 and also a reduction in loss. It was also seen that an appropriate learning rate was $\eta = 0.01$ as to small or too big learn rates impedes the ability of the model to give better predictions. Although, adding regularization reduced the overfitting when evaluating the test samples but didn't necessarily improve the accuracy of the model.
Also, in the first training for 20 epochs with no regularization there was indication of overfitting as can be seen in figure 4.0 for the CNN model.

Table 4.1: Finetuning the hyperparameters for different variations of the FCN model

	η	λ	epochs	Mloss	MAcc	Kloss	KAcc	parameters
variations								
FCN2	0.01	0.0	10	0.5626	0.7153	0.6876	0.6825	160, 130
FCN2	0.01	0.0	30	0.5349	0.7426	0.6552	0.7019	160, 130
FCN2l	0.001	0.0	10	0.6010	0.6769	0.7903	0.6675	160, 130
FCN3	0.01	0.0	10	0.5433	0.7259	0.6391	0.7131	644, 354
FCN3r	0.001	0.001	10	0.6426	0.6684	0.8099	0.6713	644, 354
FCN4	0.01	0.0	10	0.4942	0.7732	0.6762	0.6731	2, 630, 402
FCN4r	0.01	0.001	10	0.5379	0.7698	0.6729	0.7044	2, 630, 402
FCN2 nodes								
32	0.01	0.001	10	0.6053	0.7052	0.6768	0.7050	160, 130
64	0.01	0.001	10	0.6060	0.7008	0.7373	0.6856	160, 130
128	0.01	0.001	10	0.6030	0.7159	0.6782	0.7106	160, 130

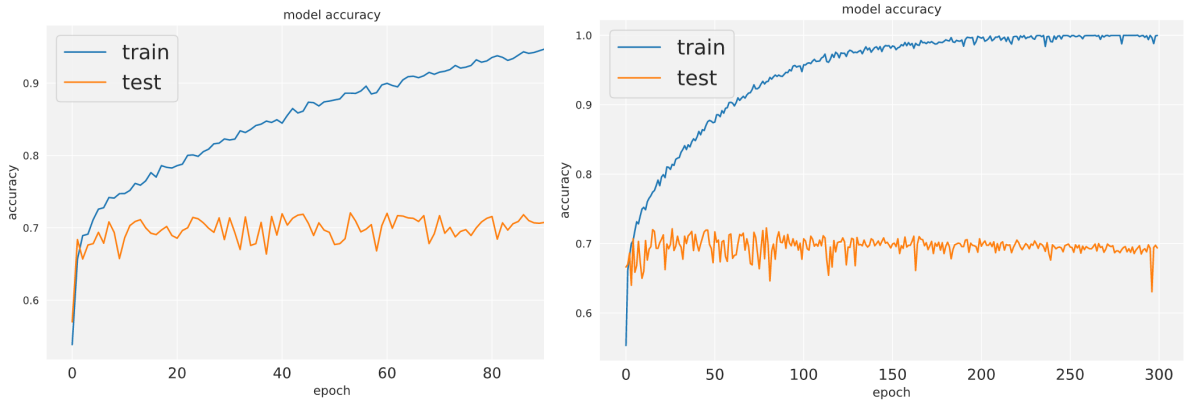
Figure 4.0: Malaria samples with and without regularization



The model had an accuracy of 96.43% with loss 0.0290 on the train set and accuracy of 95.58% with loss 0.0351 on the test set. The model with regularization had an accuracy of 93.48% with loss 0.2149 on the train set and accuracy of 93.65% with loss 0.2135 on the test set.

For the kvasir dataset, although the model improved for in FCN3 with accuracy of 0.7131 and a loss of 0.6391 but still surpassed that of FCN4 even with regularization in FCN4. The FCN3 with regularization performed much lower with accuracy 0.6713 than that of FCN because the regularization was much smaller in FCN3r. We also try to fine tune another hyperparameter, the number of nodes (neurons) with a set learning rate, $\eta = 0.01$ and regularization of $\lambda = 0.001$. It was observed that with an increase in the number of neurons, the accuracy of the model increased when the neurons for the FCN2 is 128 with the best for the malaria dataset as 0.7159 and that of the kvasir dataset as 0.7106.

It was noticed that when the number of iterations for the FCN2 was adjusted to 30 epochs , the model



(a) Accuracy graph: 100 epoch

(b) Accuracy graph: 300 epoch

Figure 4.1: Accuracy: Observation of the epochs on for kvasir samples

performed better than at 10 epochs as shown in table 4.1. Thus we investigate further by finetuning the epochs for the model and evaluating them on the data samples from our datasets with the $\eta = 0.01$ and $\lambda = 0.001$ as shown in table 4.2. For the malaria dataset, the model performed well with an accuracy of 0.9487 and loss of 0.1993 on the trainset. For the kvasir dataset, the accuracy shot up to 0.9997 and the loss decreased from 0.6356 to 0.0440 after 300 epochs.

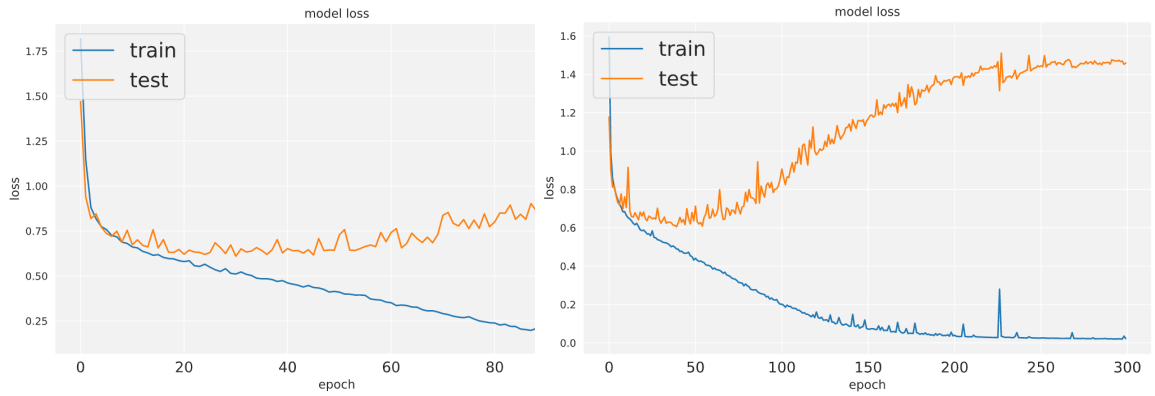
Table 4.2: Finetuning the no of iterations of the FCN model

FCN epochs	Mloss	MAcc	Kloss	KAcc
train				
10	0.6059	0.7065	0.6356	0.7419
30	0.5475	0.7568	0.4842	0.8200
50	0.4538	0.8276	0.3691	0.8734
100	0.3131	0.9065	0.1902	0.9591
300	0.1993	0.9487	0.0440	0.9997
test				
10	0.6024	0.7056	0.6954	0.7000
30	0.5575	0.7549	0.6620	0.7144
50	0.4766	0.8169	0.7249	0.7081
100	0.3846	0.8745	0.9015	0.6925
300	0.4039	0.8703	0.3063	0.6938

The train accuracy reached 0.9000 at epoch 65 with loss 0.1018 and at epoch 177 the accuracy shot up to 0.9900 with loss 0.1018. When evaluated on the test set even with regularization there was still a big difference between the losses and the accuracy. The loss of the malaria samples reduced from 0.6024 to 0.3846 for the 100 epoch when the model was trained with 300 epoch. It kept fluctuating with the difference between the train and the test increasing. It started fluctuating an at a point reached 0.9998 accuracy with loss 0.0441, while the test accuracy increased rather slowly and at epoch 300 it became 0.3063 with loss 0.6937. The best accuracy on the test set was at the epoch 30.

Several variations of the models was also implemented by adjusting the hyperparameters of the CNN model. We denote the CNN with 2-layer convolutional layer as Conv2 and CNN with 2-layer convolutional layer as Conv3. In some cases, we add the regularization parameters, batch normalization layers or dropout or both.

For the CNN, there was an increase from 0.9401 to 0.9517 on the malaria dataset when the number of



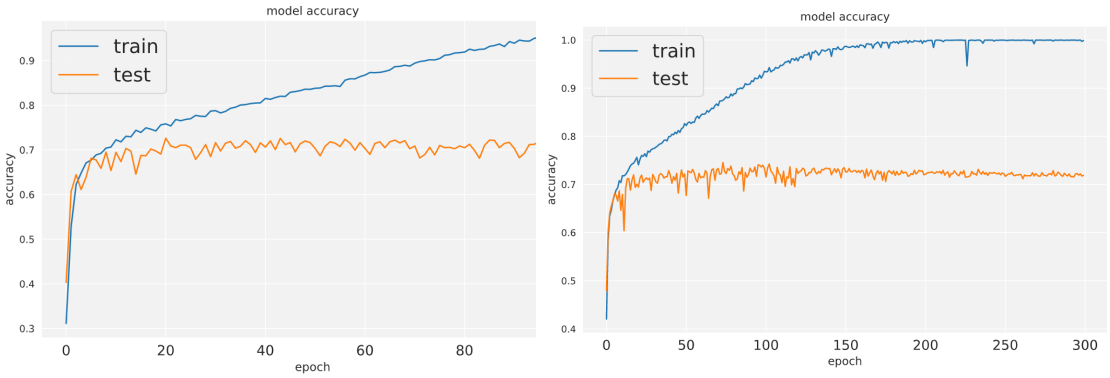
(a) 100 epoch loss (b) 300 epoch loss

Figure 4.2: Loss: Observation of the epochs on for kvasir samples

Table 4.3: Finetuning the hyperparameters for different variations of the CNN model

variations	η	Mloss	MAcc	Kloss	KAcc	parameters
Conv2	0.01	0.1713	0.9401	0.6745	0.7052	232,840
Conv2l1reg	0.01	0.2045	0.9414	0.6234	0.7225	232,840
Conv2l2reg	0.001	0.1968	0.9358	0.7108	0.6813	232,840
Conv2d25	0.01	0.5674	0.7393	0.6865	0.6837	232,840
Conv2b	0.01	0.1452	0.9523	0.8080	0.6307	233,032
Conv2bd25	0.1	0.4185	0.8944	0.6643	0.7119	233,032
Conv2bd50	0.1	0.3661	0.9171	0.6972	0.6856	233,032
Conv3	0.01	0.1596	0.9517	0.6970	0.6750	125,896
Conv3b	0.01	0.1414	0.9517	0.6240	0.7500	126,344

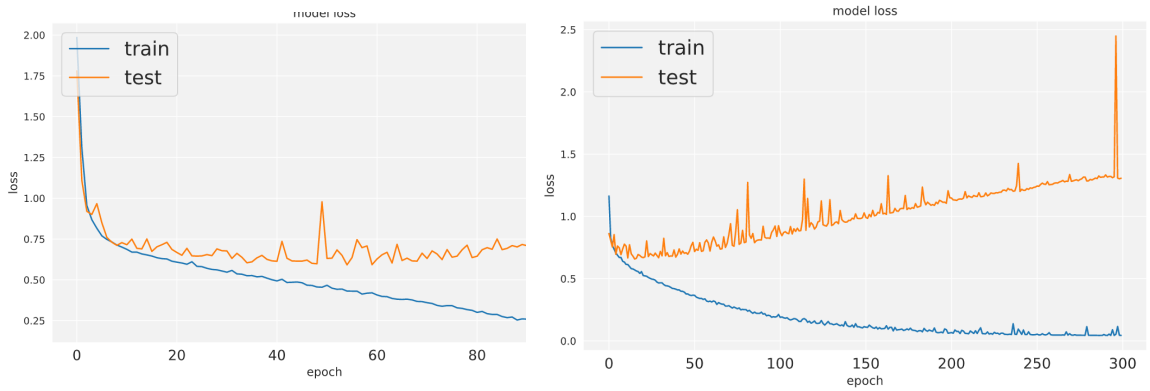
convolutional layers was increased and also a decrease in the loss from 0.1713 to 0.1596. For the kvasir dataset, the accuracy reduced from 0.7052 to 0.6750 possibly because CNN's require more samples to learn and perform better. Although when batch normalization layers was added to the Conv3, it did better with an accuracy of 0.7500 with a loss of 0.6240. The Conv2l1reg which is the 2 convolutional model with l1 regularization and $\eta = 0.01$ also performed better. The loss and accuracy for malaria samples was 0.2045 and 0.9414 respectively, for the kvasir samples the loss was 0.6234 with accuracy 0.7225. Adding the batch layer seemed to improve the accuracy for malaria samples but not so much for kvasir samples. The Conv2bd variations which is the 2 convolution layer with batch normalization layer and dropout either 0.25 or 0.5 improved the models performance. When the dropout value was 0.50 it did much better than when the value was 0.25 on the malaria samples but the reverse was the case on the kvasir samples.



(a) 100 epoch accuracy (b) 300 epoch accuracy

Figure 4.3: Accuracy: Observation of the epochs on for kvasir samples with regularization

We also investigated how adjusting the number of iterations for the CNN with 2 convolutional layers changed the models performance on our data samples , the performance of the model is shown in table 4.4 with the $\eta = 0.01$ and $\lambda = 0.001$. For the malaria dataset, the model performed surprisingly well with an accuracy of 1.000 and loss of 0.0056 on the train set at epoch 300. For the kvasir dataset, the accuracy reached 0.9992 and the loss decreased from 0.7064 to 0.0220 after 300 epochs. The train accuracy for the kvasir samples had a loss reducing to 0.0198 from epoch 121 with accuracy 0.9991. The loss kept decreasing with loss 0.0088 and train accuracy of 1.000 but slow improvement on the test.



(a) 100 epoch loss

(b) 300 epoch loss

Figure 4.4: Loss: Observation of the epochs on for kvasir samples with regularization

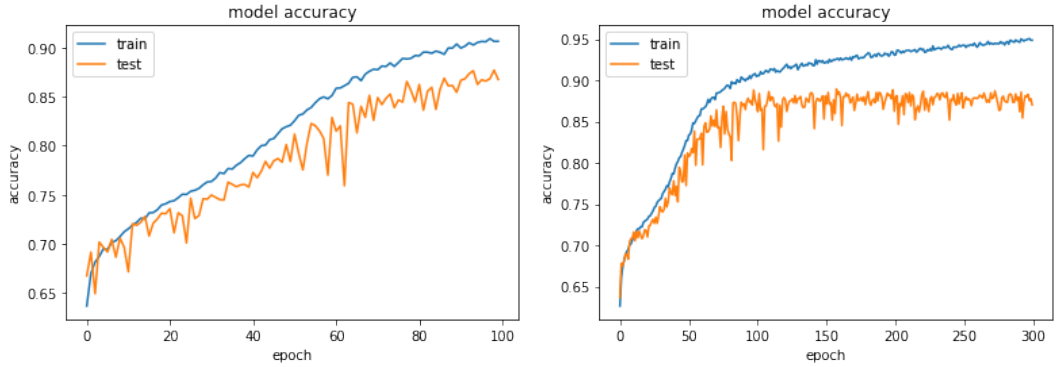
Table 4.4: Finetuning the no of iterations of the CNN model

CNN epochs	Mloss	MAcc	Kloss	KAcc
train				
10	0.1971	0.9316	0.6763	0.7064
30	0.1345	0.9578	0.5143	0.7837
50	0.1029	0.9672	0.4358	0.8266
100	0.0381	0.9935	0.1564	0.9528
300	0.0056	1.000	0.0220	0.9992
test				
10	0.2005	0.9365	0.7026	0.6731
30	0.1632	0.9543	0.6019	0.7269
50	0.1973	0.9452	0.6970	0.6906
100	0.1988	0.9520	0.9148	0.7100
300	0.2569	0.9476	1.4534	0.7269

Evaluating the model on the test set even with regularization there was still a big difference between the losses and the accuracy. The loss of the malaria samples reduced from 0.2005 to 0.1988 for the 100 epoch with accuracy increased from 0.9365 to 0.9520. It kept fluctuating with the performance difference between the train and the test increasing. The model was evaluated on the test with 300 epoch had an accuracy of 0.9457 for malaria samples and accuracy of 0.7269 for the kvasir samples.

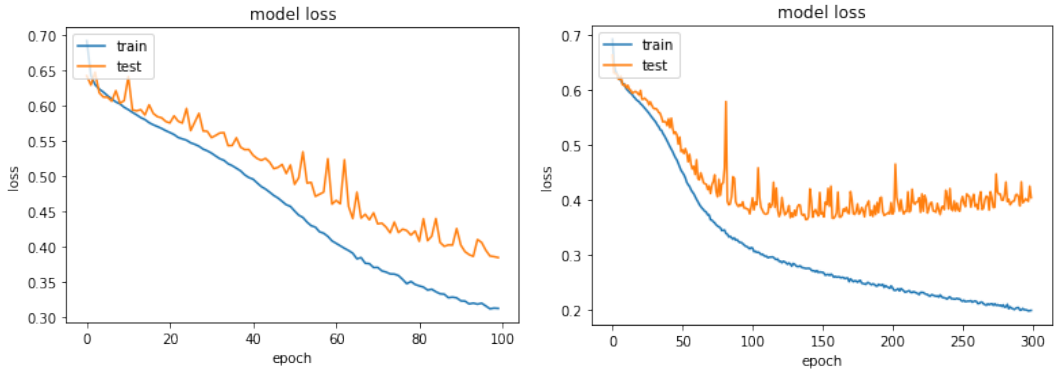
Thus using the observations gotten from finetuning our parameters we build better models. For the FCN, we build a 3 layer model with increased number of nodes, regularization as 0.001, learning rate , $\eta = 0.01$ and evaluated for epochs 100 and 300. For the CNN, we implement a model with 3 convolutional layer, 3 maxpooling layer, batch normalization layers, 2 dense layers, dropout with value 0.5 regularization as 0.001, learning rate , $\eta = 0.01$ and evaluated for epochs 100 and 300.

Now we show graphically the loss and accuracy after several trainings on both dataset samples with a clear decrease in overfitting. The graphs for the Kvasir samples before and after overfitting are shown in figures 4.1, 4.2, 4.5 and 4.4.



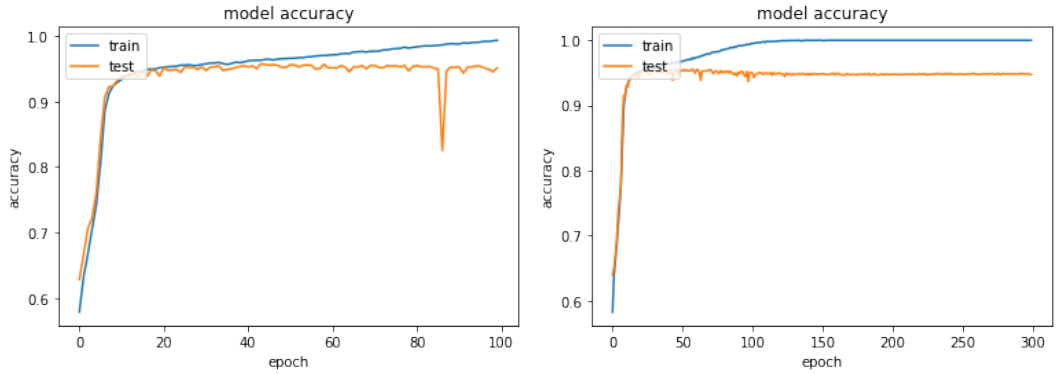
(a) 100 epoch accuracy (b) 300 epoch accuracy

Figure 4.5: FCN accuracy Observation of the epochs on for malaria samples with regularization



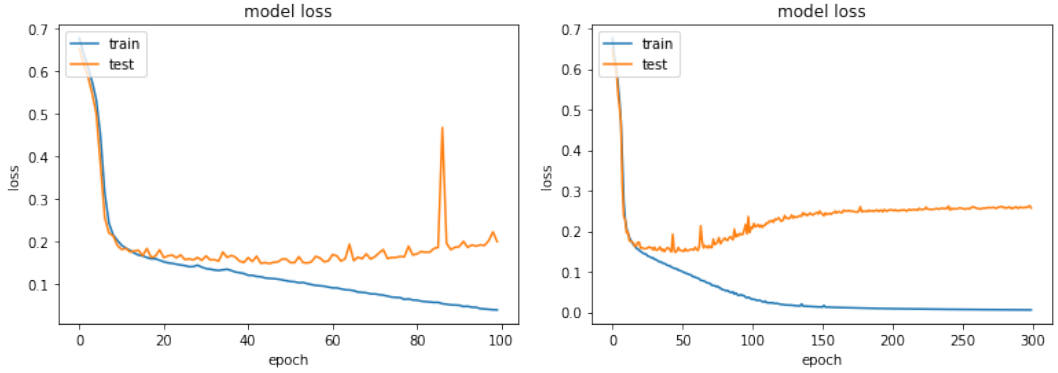
(a) 100 epoch loss (b) 300 epoch loss

Figure 4.6: FCN loss Observation of the epochs on for malaria samples with regularization



(a) 100 epoch accuracy (b) 300 epoch accuracy

Figure 4.7: CNN accuracy Observation of the epochs on for malaria samples with regularization



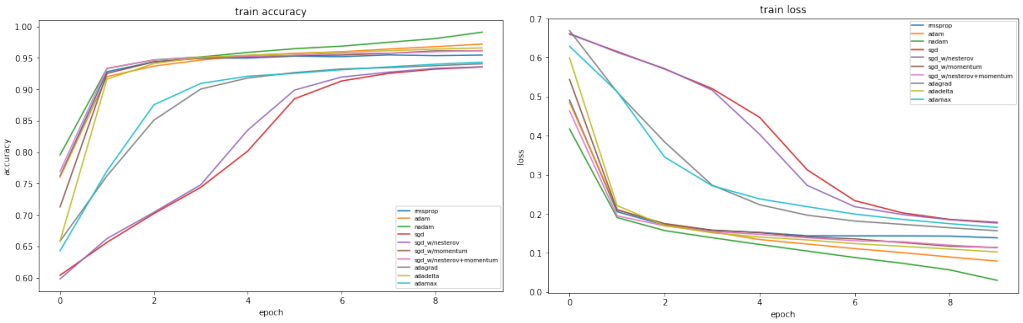
(a) 100 epoch loss (b) 300 epoch loss

Figure 4.8: CNN accuracy Observation of the epochs on for malaria samples with regularization

4.2 Analysis of Results for the Optimization Algorithms

We compare how using different optimization algorithms which we call our optimizers can affect the performance of our neural network to be able to produce slightly better and faster results when updating our models parameters. We tried different optimizers such as the SGD, SGD with Nesterov momentum and compared with other adaptive optimizer such as the Adam, RMSprop and AdaGrad and observed how good it gets at generalizing on the test set for our image classification problem. The summary of our analysis for both datasets are shown in table 4.5 and table 4.6. We chose the hyperparameters which achieved the lowest training loss for certain number of iterations.

Training our CNN model using the SGD, we observed that with parameter updates there are fluctuations and it slows the process of convergence to an exact minimum which will also keep overshooting due to frequent fluctuations. For the malaria samples, The SGD variants (i.e momentum and nesterov) performed much better than the SGD with a significant increase in accuracy and decrease in loss. Amongst the variiants of the SGD, the SGD with momentum performed significantly well on both the train set with loss of 0.1138 as well as on the test set with a loss of 0.1625. The accusracy on the test set improved from 0.9387 to 0.9536. The momentum value was set at $\beta = 0.9$ and added to the SGD since the was increase in fluctuations, the momentum speeds up the SGD and directs it towards a more suitable direction and reduces the fluctuations in that direction.



(a) Accuracy graph (b) Loss graph

Figure 4.9: Optimizer Comparison Loss and Accuracy for the Malaria Dataset Model on the Train samples

The best performance on both the train loss and train accuracy was the adaptive optimizer, Nadam [3]. It had an accuracy of 0.9910, it was followed by the Adam with 0.9721 on the training samples also the SGD with nestorov momentum had a loss of 0.1133. SGD nestorov momentum performed better than the Adamax with a loss of 0.1654 and the Adagrad with a loss of 0.1025. The optimizer with the best test accuracy and loss was a non-adaptive algorithm, SGD with Momentum with an accuracy of 0.9536, whereas the adaptive method with the best accuracy and loss was the Adadelta which had an accuracy of 0.9505. During training, the adaptive methods appear to be perform significantly better than the non-adaptive methods, but for epoch 10 on the test set we see the opposite even though the loss for the adaptive methods were reducing with the difference between the best performing and the worst performing optimizer being less than 0.03. We observe

Table 4.5: Comparison of Optimizers for the Malaria Dataset

Optimizer	Train		Test	
	Loss	Acc	Test-loss	Test-acc
1. RMSprop	n=1: 0.4910 n=5: 0.1526 n=10: 0.1388	n=1: 0.7621 n=5: 0.9498 n=10: 0.9546	n=1: 0.2477 n=5: 0.1685 n=10: 0.1652	n=1: 0.9084 n=5: 0.9459 n=10: 0.9496
2. Adam	n=1: 0.4854 n=5: 0.1343 n=10: 0.0790	n=1: 0.7602 n=5: 0.9539 n=10: 0.9721	n=1: 0.2515 n=5: 0.1649 n=10: 0.1861	n=1: 0.9006 n=5: 0.9479 n=10: 0.9472
3. Nadam	n=1: 0.4170 n=5: 0.1217 n=10: 0.0300	n=1: 0.7956 n=5: 0.9588 n=10: 0.9910	n=1: 0.2272 n=5: 0.1725 n=10: 0.2158	n=1: 0.9225 n=5: 0.9421 n=10: 0.9458
4. SGD	n=1: 0.6597 n=5: 0.4468 n=10: 0.1781	n=1: 0.6040 n=5: 0.8016 n=10: 0.9357	n=1: 0.6217 n=5: 0.3852 n=10: 0.1784	n=1: 0.6578 n=5: 0.8572 n=10: 0.9387
5. SGD w/Nesterov	n=1: 0.6614 n=5: 0.4035 n=10: 0.1763	n=1: 0.5982 n=5: 0.8349 n=10: 0.9358	n=1: 0.6963 n=5: 0.3200 n=10: 0.1751	n=1: 0.5689 n=5: 0.8882 n=10: 0.9389
6. SGD w/Momentum	n=1: 0.5437 n=5: 0.1518 n=10: 0.1138	n=1: 0.7128 n=5: 0.9512 n=10: 0.9614	n=1: 0.2632 n=5: 0.1751 n=10: 0.1625	n=1: 0.9002 n=5: 0.9468 n=10: 0.9536
7. SGD w/NestMom	n=1: 0.4632 n=5: 0.1474 n=10: 0.1133	n=1: 0.7694 n=5: 0.9525 n=10: 0.9619	n=1: 0.2098 n=5: 0.1647 n=10: 0.1510	n=1: 0.9276 n=5: 0.9478 n=10: 0.9528
8. Adagrad	n=1: 0.6688 n=5: 0.2234 n=10: 0.1568	n=1: 0.6584 n=5: 0.9180 n=10: 0.9407	n=1: 0.5553 n=5: 0.2147 n=10: 0.1774	n=1: 0.7255 n=5: 0.9240 n=10: 0.9356
9. Adadelta	n=1: 0.5987 n=5: 0.1408 n=10: 0.1025	n=1: 0.6588 n=5: 0.9544 n=10: 0.9659	n=1: 0.3259 n=5: 0.1571 n=10: 0.1550	n=1: 0.9058 n=5: 0.9510 n=10: 0.9505
10. Adamax	n=1: 0.6285 n=5: 0.2382 n=10: 0.1654	n=1: 0.6430 n=5: 0.9209 n=10: 0.9432	n=1: 0.5898 n=5: 0.2439 n=10: 0.1857	n=1: 0.6782 n=5: 0.9291 n=10: 0.9410

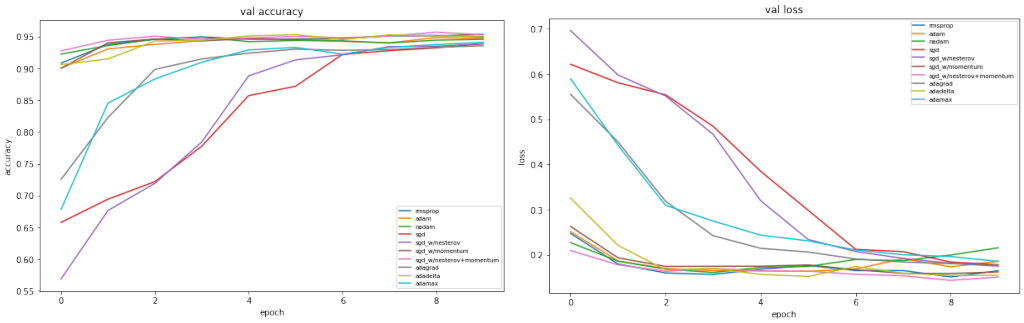
a similar performance between the SGD and SGD with Nesterov momentum. We show graphically the results we observes in figure 4.9 for the training and in figure 4.10 on the test samples of the malaria dataset.

For the Kvasir endoscopic samples, The SGD variants still performed much better than the SGD with a significant increase in accuracy and decrease in loss after 30 epochs. Amongst the variants of the SGD, the SGD with Nesterov momentum performed significantly well on both the train set with loss of 0.0755 and an accuracy of 0.9701. It also performed well on the test set with a loss of 0.1625 and an accuracy of 0.9701. The loss of the SGD on the train set was 0.1527 and an accuracy of 0.9293. On the test set the loss decreased from 0.3593 to 0.1596 and the accuracy improved from 0.8750 to 0.9243.

Table 4.6: Comparison of Optimizers for the Kvasir Dataset

Optimizer	Train		Test	
	Loss	Acc	Test-loss	Test-acc
1. RMSprop	n=1: 0.2185 n=10: 0.1044 n=20: 0.0440 n=30: 0.0346	n=1: 0.8978 n=10: 0.9547 n=20: 0.9851 n=30: 0.9911	n=1: 0.1756 n=10: 0.1376 n=20: 0.1476 n=30: 0.1531	n=1: 0.9107 n=10: 0.9365 n=20: 0.9386 n=30: 0.9388
2. Adam	n=1: 0.2133 n=10: 0.1010 n=20: 0.0577 n=30: 0.0549	n=1: 0.9013 n=10: 0.9558 n=20: 0.9791 n=30: 0.9807	n=1: 0.1591 n=10: 0.1329 n=20: 0.1321 n=30: 0.1337	n=1: 0.9238 n=10: 0.9381 n=20: 0.9413 n=30: 0.9409
3. Nadam	n=1: 0.2080 n=10: 0.0660 n=20: 0.0244 n=30: 0.0226	n=1: 0.9006 n=10: 0.9736 n=20: 0.9932 n=30: 0.9942	n=1: 0.1571 n=10: 0.1488 n=20: 0.1710 n=30: 0.1722	n=1: 0.9267 n=10: 0.9389 n=20: 0.9419 n=30: 0.9412
4. SGD	n=1: 0.3678 n=10: 0.1863 n=20: 0.1626 n=30: 0.1527	n=1: 0.8750 n=10: 0.9118 n=20: 0.9236 n=30: 0.9293	n=1: 0.3593 n=10: 0.1839 n=20: 0.1697 n=30: 0.1577	n=1: 0.8750 n=10: 0.9134 n=20: 0.9178 n=30: 0.9245
5. SGD w/Nesterov	n=1: 0.3717 n=10: 0.1836 n=20: 0.1635 n=30: 0.1543	n=1: 0.8750 n=10: 0.9143 n=20: 0.9229 n=30: 0.9272	n=1: 0.3640 n=10: 0.1793 n=20: 0.1623 n=30: 0.1596	n=1: 0.8750 n=10: 0.9134 n=20: 0.9178 n=30: 0.9243
6. SGD w/Momentum	n=1: 0.3400 n=10: 0.1466 n=20: 0.1211 n=30: 0.0991	n=1: 0.8759 n=10: 0.9308 n=20: 0.9452 n=30: 0.9573	n=1: 0.2659 n=10: 0.1564 n=20: 0.1400 n=30: 0.1450	n=1: 0.8800 n=10: 0.9240 n=20: 0.9316 n=30: 0.9334
7. SGD w/NestMom	n=1: 0.2900 n=10: 0.1414 n=20: 0.1136 n=30: 0.0755	n=1: 0.8835 n=10: 0.9339 n=20: 0.9494 n=30: 0.9701	n=1: 0.1950 n=10: 0.1515 n=20: 0.1334 n=30: 0.1346	n=1: 0.9080 n=10: 0.9228 n=20: 0.9389 n=30: 0.9392
8. Adagrad	n=1: 0.2057 n=10: 0.1085 n=20: 0.0890 n=30: 0.0796	n=1: 0.9051 n=10: 0.9539 n=20: 0.9622 n=30: 0.9696	n=1: 0.1560 n=10: 0.1252 n=20: 0.1236 n=30: 0.1222	n=1: 0.9241 n=10: 0.9402 n=20: 0.9437 n=30: 0.9427
9. Adadelta	n=1: 0.2508 n=10: 0.1187 n=20: 0.0637 n=30: 0.0525	n=1: 0.8863 n=10: 0.9463 n=20: 0.9760 n=30: 0.9817	n=1: 0.1881 n=10: 0.1505 n=20: 0.1318 n=30: 0.1338	n=1: 0.9038 n=10: 0.9306 n=20: 0.9385 n=30: 0.9400
10. Adamax	n=1: 0.2269 n=10: 0.1217 n=20: 0.0920 n=30: 0.0857	n=1: 0.8969 n=10: 0.9440 n=20: 0.9629 n=30: 0.9665	n=1: 0.1726 n=10: 0.1411 n=20: 0.1241 n=30: 0.1215	n=1: 0.9138 n=10: 0.9336 n=20: 0.9379 n=30: 0.9406

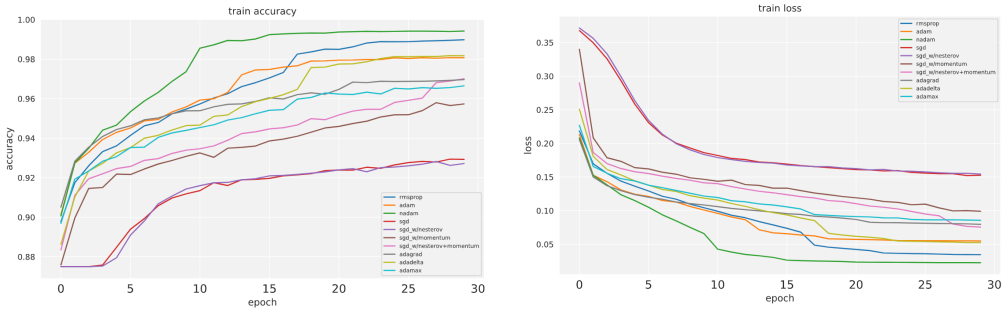
The best performance on both the train loss and train accuracy was still the adaptive optimizer, Nadam [7]. It had an accuracy of 0.9942, it was followed by the RMSprop with 0.99911 and then the Adadelta with accuracy of 0.9817 on the training samples. The loss of the RMSprop rapidly reduced from 0.2185 to 0.0346 with the accuracy significantly increased from 0.8978 to 0.9991. The loss of the Nadam on the train set decreased from 0.2080 to 0.0226.



(a) Accuracy graph (b) Loss graph

Figure 4.10: Optimizer Comparison Loss and Accuracy for the Malaria Dataset Model on the Test samples

The optimizer with the best test accuracy and loss was an adaptive algorithm, Adagrad with an accuracy of 0.9427 and loss 0.1222, whereas the non-adaptive method with the best accuracy and loss was the SGD nesterov momentum which had an accuracy of 0.9392 with loss 0.1346.

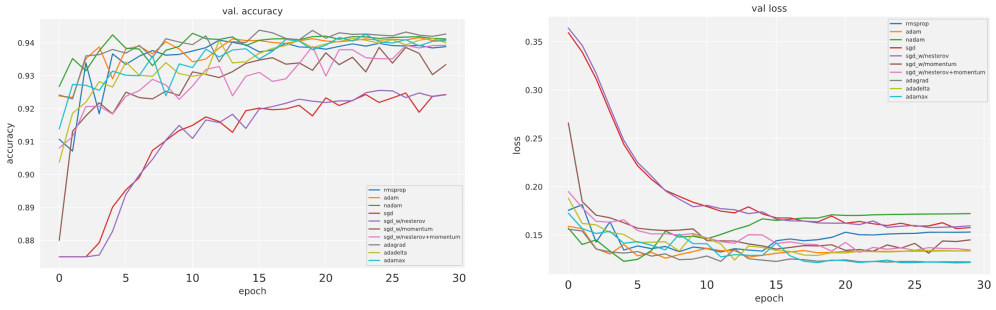


(a) Accuracy graph (b) Loss graph

Figure 4.11: Optimizer Comparison Loss and Accuracy for the Kvasir Dataset Model on the Train samples

During training, the adaptive methods appear to be perform significantly better than the non-adaptive methods for 30 epoches and also the same on the test set with the loss for the adaptive methods were reducing signficantly with the difference between the best performing and the worst performing optimizer being less than 0.02. We observe a similar performance between the SGD and SGD with Nesterov momentum also for the kvasir samples. We show graphically the results we observes in figure 4.11 for the training and in figure 4.12 on the test samples of the Kvasir dataset.

The adaptive methods have worse generalization on the test samples in malaria dataset even thought the



(a) Accuracy graph (b) Loss graph

Figure 4.12: Optimizer Comparison Loss and Accuracy for the Kvasir Dataset Model on the Test samples

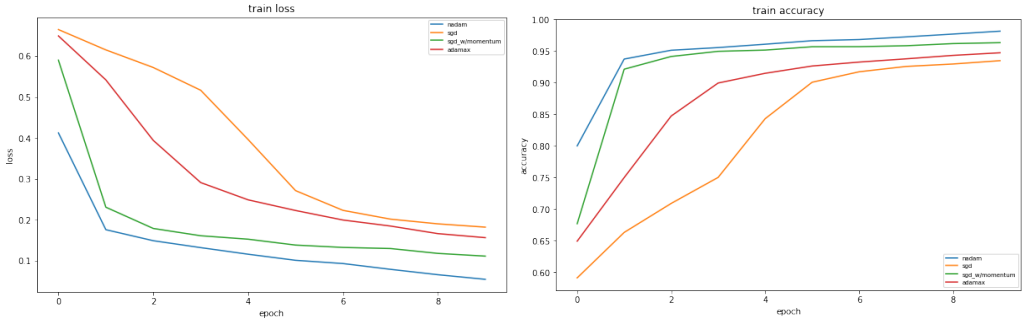
performance on the results from the training process where great.

Thus we used the hyperparameters which we observed improved the network's model performance to build deeper CNN model and we trained our model more with the observed optimizers which performed best and also

the worst for both datasets. Further training on the malaria samples showed that the Nadam still performed best with a loss of 0.0545 and an accuracy of 0.9810. The SGD had the worst predictions with a loss of 0.1820 and an accuracy of 0.9344. The Adamax had a loss of 0.1563 and an accuracy of 0.9469 while the SGD momentum had a loss of 0.1112 and an accuracy of 0.9629. This analysis is shown in figure 4.13 and table 4.7.

Table 4.7: Best and worst optimizers after finetuning for the Malaria train samples

train	MAcc	Mloss
Nadam	98.10%	0.0545
SGD	93.44%	0.1820
Adamax	94.69%	0.1563
SGDmom	96.29%	0.1112



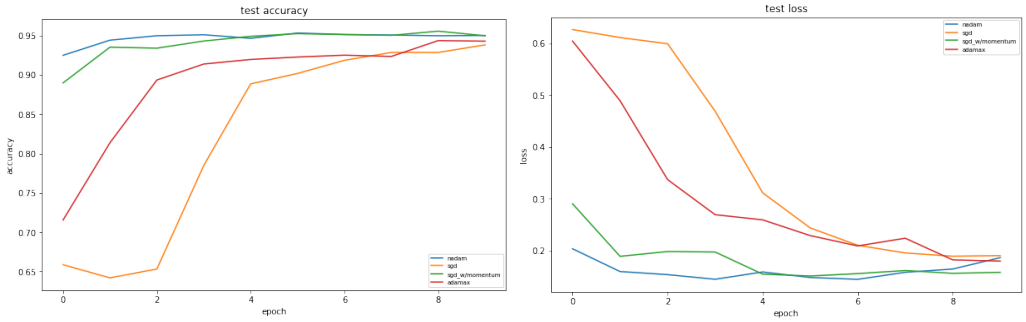
(a) Accuracy graph (b) Loss graph

Figure 4.13: Further Optimizer Comparison Loss and Accuracy for the malaria Dataset Model on the Test samples

The analysis as shown in figure 4.14 and table 4.8 on the test samples, Nadam had a loss of 0.1861 and with the best accuracy of 0.9499. The SGD had a loss of 0.1900 and an accuracy of 0.9380. The Adamax had a loss of 0.1794 and with the worst accuracy of 0.9270 while the SGD momentum had a loss of 0.1577 and an accuracy of 0.9496.

Table 4.8: Best and worst optimizers after finetuning for the Malaria test samples

test	MAcc	Mloss
Nadam	94.99%	0.1861
SGD	93.80%	0.1794
Adamax	92.70%	0.1900
SGDmom	94.96%	0.1577



(a) Accuracy graph (b) Loss graph

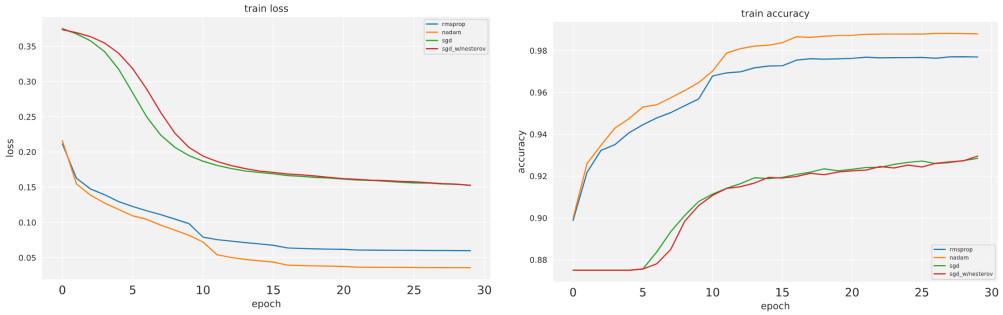
Figure 4.14: Further Optimizer Comparison Loss and Accuracy for the Malaria Dataset Model on the Test samples

Further training on the kvasir samples showed that the Nadam also performed best with a loss of 0.0358

and an accuracy of 0.9880. The SGD nesterov had the worst predictions with a loss of 0.1526 and an accuracy of 0.9285. The RMSprop had a loss of 0.0599 and an accuracy of 0.9770 while the SGD had a loss of 0.1527 and an accuracy of 0.9295.This analysis is shown in figure 4.15 and table 4.9.

Table 4.9: Best and worst optimizers after finetuning for the Kvasir train samples

train	KAcc	Kloss
Nadam	98.80%	0.0358
SGD	92.95%	0.1527
RMSprop	97.70%	0.0599
SGDnest	92.85%	0.1526



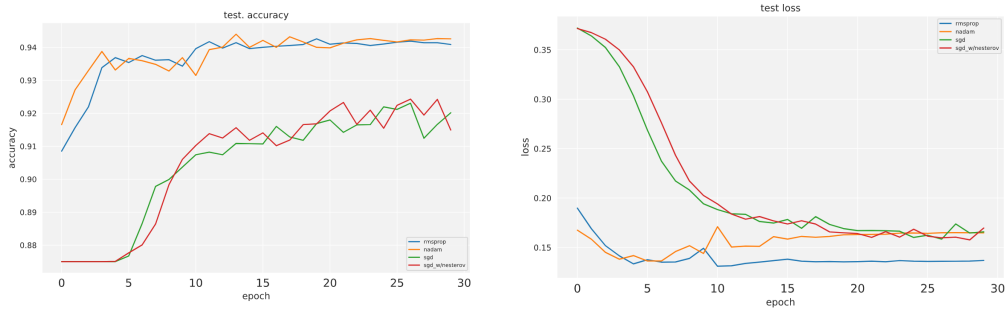
(a) Accuracy graph (b) Loss graph

Figure 4.15: Further Optimizer Comparison Loss and Accuracy for the Kvasir Dataset Model on the Test samples

The analysis as shown in figure 4.16 and table 4.10 on the test samples, Nadam had a loss of 0.1648 and with the best accuracy of 0.9426. The SGD nesterov had a loss of 0.1696 and with the worst accuracy of 0.9149. The RMSprop had a loss of 0.1368 and an accuracy of 0.9409 while the SGD had a loss of 0.1658 and an accuracy of 0.9202.

Table 4.10: Best and worst optimizers after finetuning for the Kvasir test samples

test	KAcc	Kloss
Nadam	94.26%	0.1648
SGD	92.02%	0.1658
RMSprop	94.09%	0.1368
SGDnest	91.49%	0.1696



(a) Accuracy graph (b) Loss graph

Figure 4.16: Further Optimizer Comparison Loss and Accuracy for the Kvasir Dataset Model on the Test samples

Chapter 5

Conclusion

In this work we implemented two (2) neural network model, the Fully Connected Neural Network (FCN) and the of the Convolutional Neural Network (CNN). We explained the main components of these networks and how the backpropagation works in them. The models were then trained on two medical image datasets on kaggle namely the malaria cell images dataset and the kvasir dataset. We utilized the GPU memory of the kaggle kernel with a ram size of 13gb because the normal jupyter kernels on standard laptops with GPU trains slowly or fails during the fitting of the model when the amount of trainable parameters are to big. We carry out the image classification task by training our models to make good predictions when equiped with the right parameters in turn reducing the workload and misdiagnosis of such diseases by medical practitioners. This aids the treatment of the patients who posibbly have these disease and reduces the severity of the disease if properly classified. It also makes the it faster whereas humans would take hours to classify these images, the network models if trained appropriately would learn to correctly predict the classes of these images with minimal error.

We analysed the performance of these network models on both datasets. For the malaria dataset, we noticed a bit of overfitting due to the large number of trainable parameters which was counteracted using techniques like regularization. The CNN performed better with an accuracy of 93.11% while the FCN had an accuracy of 76.18% at 10 epochs after analysing the performance of both models. Although the FCN is a good classifier, they are not good feature extractors but the CNN best quality is identifying and extracting the features from the images during training. For the Kvasir data samples, the CNN model performed worse than the FCN model on the test samples after 10 epochs. The CNN had an accuracy of 70.06% whereas the FCN had an accuracy of 70.30%. The possible reason could be because the dataset has fewer training samples and the CNN generally need more training samples to learn better. It also needs to more convolutional layers so that the network can learn more significant features of the images.

We then try to see how adjustments to the hyperparameters of our models as well as the optimizers can improve the models performance. This helps us identify the best elements to use which after observation from our analysis was then used to build a deeper CNN model which showed significant improvement with an accuracy of 94.99% on the malaria dataset and accuracy of 94.26% on the kvasir dataset for the CNN model. It was observed that although the adaptive optimizers such as Nadam performed better with a high accuracy on the training samples, it didn't generalize so well on the test samples whereas the non-adaptive methods such as SGD with momentum although seemed to perform lower than the adaptive optimizers with the difference in accuracy being less than 0.4, they performed better than some of the adaptive algorithms. This suggest that in training more complex and deeper networks the adaptive methods still have a lot of improvements to be made in order to improve its generalization on the test samples.

Future Work

Further future work can be carried out by analyzing how higher order optimization algorithms can affect the performance of deep learning models on the medical imaging datasets and further image processing applications such as image synthesis, segmentation, de-noising, detection and reconstruction. Also to solve the issue of insufficient medical imaging training samples, a fast developing state-of-the-art method known as the Generative Adversarial Networks (GANs) can be used to mimic data sample distributions thus creating a synthesis of data images which looks realistic and close in similarity to the actual data samples.

Bibliography

[1] F. Altaf, S. Islam, N. Akhtar, and N. K. Janjua, "Going deep in medical image analysis: Concepts, methods, challenges and future directions," *arXiv preprint arXiv:1902.05655*, 2019.

[2] N. Sharma, V. Jain, and A. Mishra, "An analysis of convolutional neural networks for image classification," *Procedia computer science*, vol. 132, pp. 377–384, 2018.

[3] G. A. Eze, "Applications of neural network models in kaggle competitions," in *International Student Scientific Conference Of Applied Mathematics and Computer Science, Lviv Oblast, Lviv, Ukraine, April 18-19, 2019*, (Lviv, Ukraine), pp. 33–36, ISSCAMCS – 2019, 2019.

[4] N. N. Baxter, R. Sutradhar, S. S. Forbes, L. F. Paszat, R. Saskin, and L. Rabeneck, "Analysis of administrative data finds endoscopist quality measures associated with postcolonoscopy colorectal cancer," *Gastroenterology*, vol. 140, no. 1, pp. 65–72, 2011.

[5] T. de Lange, P. Halvorsen, and M. Riegler, "Methodology to develop machine learning algorithms to improve performance in gastrointestinal endoscopy," *World journal of gastroenterology*, vol. 24, no. 45, p. 5057, 2018.

[6] K. Pogorelov, O. Ostroukhova, M. Jeppsson, H. Espeland, C. Griwodz, T. de Lange, D. Johansen, M. Riegler, and P. Halvorsen, "Deep learning and hand-crafted feature based approaches for polyp detection in medical videos," in *2018 IEEE 31st International Symposium on Computer-Based Medical Systems (CBMS)*, pp. 381–386, IEEE, 2018.

[7] K. Pogorelov, O. Ostroukhova, A. Petlund, P. Halvorsen, T. de Lange, H. N. Espeland, T. Kupka, C. Griwodz, and M. Riegler, "Deep learning and handcrafted feature based approaches for automatic detection of angiectasia," in *2018 IEEE EMBS International Conference on Biomedical & Health Informatics (BHI)*, pp. 365–368, IEEE, 2018.

[8] K. Pogorelov, K. R. Randel, C. Griwodz, S. L. Eskeland, T. de Lange, D. Johansen, C. Spampinato, D.-T. Dang-Nguyen, M. Lux, P. T. Schmidt, M. Riegler, and P. Halvorsen, "Kvasir: A multi-class image dataset for computer aided gastrointestinal disease detection," in *Proceedings of the 8th ACM on Multimedia Systems Conference*, MMSys'17, (New York, NY, USA), pp. 164–169, ACM, 2017.

[9] S. Ameling, S. Wirth, D. Paulus, G. Lacey, and F. Vilarino, "Texture-based polyp detection in colonoscopy," in *Bildverarbeitung für die Medizin 2009*, pp. 346–350, Springer, 2009.

[10] Y. Amano, N. Ishimura, K. Furuta, K. Okita, M. Masaharu, T. Azumi, T. Ose, K. Koshino, S. Ishihara, K. Adachi, *et al.*, "Interobserver agreement on classifying endoscopic diagnoses of nonerosive esophagitis," *Endoscopy*, vol. 38, no. 10, pp. 1032–1035, 2006.

[11] F. G. Zanjani, S. Zinger, and P. N. De, "Automated detection and classification of cancer metastasis in whole-slide histopathology images using deep convnets," 2017.

[12] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25* (F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, eds.), pp. 1097–1105, Curran Associates, Inc., 2012.

[13] A. Arsenov, I. Ruban, K. Smelyakov, and A. Chupryna, "Evolution of convolutional neural network architecture in image classification problems,"

[14] S. Cheng, S. Zeng, and J. Yu, "Automatic prediction of breast cancer metastasis stages via deep convolutional networks,"

[15] S. Pavithra, "Convolutions and backpropagation," 2018.

[16] V. Dumoulin and F. Visin, "A guide to convolution arithmetic for deep learning," *arXiv preprint arXiv:1603.07285*, 2016.

[17] Z. Zhang, "Derivation of backpropagation in convolutional neural network (cnn)," *University of Tennessee, Knoxville, TN*, 2016.

[18] S. Kajan, D. Pernecký, and J. Goga, "Application of neural network in medical diagnostics," *SK: University of Technology in Bratislava*, 2014.

[19] J. Wu, "Cnn for dummies," *Nanjing University*, vol. 202, 2015.

[20] D. Cascio, V. Taormina, and G. Raso, "Deep convolutional neural network for hep-2 fluorescence intensity classification," *Applied Sciences*, vol. 9, no. 3, p. 408, 2019.

[21] S. Padmanabhan, "Convolutional neural networks for image classification and captioning," 2016.

[22] A. Calderón, S. Roa, and J. Victorino, "Handwritten digit recognition using convolutional neural networks and gabor filters," in *Proceedings of International Congress on Computational Intelligence CIIC*, 2003.

[23] S. L. Hijazi, R. R. Kumar, and C. Rowen, "Using convolutional neural networks for image recognition," 2017.

[24] M. D. McDonnell and T. Vladusich, "Enhanced image classification with a fast-learning shallow convolutional neural network," in *2015 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–7, IEEE, 2015.

[25] G. Andrew, "Convolutional neural networks," 2014.

[26] G. Grzegorz, "Convolutional neural networks backpropagation: from intuition to derivation.," 2016.

[27] Jefkine, "Backpropagation in convolutional neural networks," 2016.

[28] A. Caterini, "A novel mathematical framework for the analysis of neural networks," Master's thesis, University of Waterloo, 2017.

Annex A

Acronyms

Acronyms	meaning
λ	the regularization parameter
η	the learning rate
Mloss	the loss from the training on the malaria samples
MAcc	the accuracy from the training on the malaria samples
Kloss	the loss from the training on the kvasir samples
KAcc	the accuracy from the training on the kvasir samples
FCN2	FCN with 2-layer
FCN2l	FCN with 2-layer with different learning rate
FCN3	FCN with 3-layer
FCN3r	FCN with 3-layer and regularization
FCN4	FCN with 4-layer
FCN4r	FCN with 4-layer and regularization
Conv2	model with 2 Convolution layer
Conv2l1reg	model with 2 Convolution layer + l1 regularization
Conv2l2reg	model with 2 Convolution layer + l2 regularization
Conv2d25	model with 2 Convolution layer + dropout = 0.25
Conv2b	model with 2 Convolution layer + batch normalization
Conv2bd25	model with 2 Convolution layer + batch normalization, dropout = 0.25
Conv2bd50	model with 2 Convolution layer + batch normalization, dropout = 0.50
Conv3	model with 3 Convolution layer
Conv3b	model with 3 Convolution layer + batch normalization