Deep Learning in Malaria and Covid19 Detection

By

Nahid Islam
17321002
Omar Sadat
17121084
Tazwar Prodhan Shaer
17121094
Md. Adnan Islam
16121131

A thesis submitted to the Department of Electrical and Electronic Engineering in partial fulfillment of the requirements for the degree of
B.Sc. in EEE

Department of Electrical and Electronic Engineering
Brac University
January 2021

Declaration

It is hereby declared that

1. The thesis submitted is our own original work while completing degree at Brac University.
2. The thesis does not contain material previously published or written by a third party, except where this is appropriately cited through full and accurate referencing.
3. The thesis does not contain material which has been accepted, or submitted, for any other degree or diploma at a university or other institution.
4. We have acknowledged all main sources of help.

Student's Full Name & Signature:

Nahid Islam 17321002	**Omar Sadat** 17121084
Tazwar Prodhan Shaer 17121094	**Md. Adnan Islam** 16121131

Approval

The thesis/project titled “[Thesis/Project Title]” submitted by

1. Nahid Islam (17321002)
2. Omar Sadat (17121084)
3. Tazwar Pradhan Shaer (17121094)
4. Md. Adnan Islam (16121131)

of Spring, 2021 has been accepted as satisfactory in partial fulfillment of the requirement for the degree of B. Sc.in EEE on January 13,2021.

Examining Committee:

Supervisor:
(Member)

_____
Abu S.M. Mohsin, PhD
Assistant Professor, Department of Electrical and Electronic Engineering
Brac University

Program Coordinator:
(Member)

_____
Abu S.M. Mohsin, PhD
Assistant Professor, Department of Electrical and Electronic Engineering
Brac University

Departmental Head:
(Chair)

_____
Md. Mosaddequr Rahman, PhD
Professor and Chairperson, Department of Electrical and Electronic Engineering
Brac University

Ethics Statement

Abstract

Recent days deep learning is a promising field of research which has been used in biological applications. It is deep learning which is used in biological image detection along with the transformation of analysis data and also in interpreting those data. The revolution of DL enables the researchers and scientists in a practical manner that they can easily analysis the difficult data within very short period and more accurate way that was previously considered impossible. Here we will study the intersection between deep learning with computer image analysis that are pertinent to life scientists. We will study the principles of CNN, image classification, different CNN models and lastly the Flask application. Later we will implement deep learning with annotation of training data, selection of data and also present the training a range of neural network architectures and finally the deploying solutions. Here will use four different optimized neural network models named ResNet50, Inception_v3, Xception and VGG-19 to find out among these which will give the most accurate and precise results. The major purpose of our research is to find the model with the best accuracy for future use. From our findings we see that the Xception model has the highest accuracy among the other models we used. The conclusion points to the importance of implementing a proper model for detection of diseases using images and DL for progress in health technology and future researches.

Keywords: Deep Learning; Neural Network; Training Data; Deploying flask; Datasets; Prediction; Diseases Detection

Dedication (Optional)

A dedication is the expression of friendly connection or thanks by the author towards another person. It can occupy one or multiple lines depending on its importance.

You can remove this page if you want.

Acknowledgement

We would like to express the most profound and most sincere appreciation to our Thesis Supervisor Abu S.M. Mohsin, PhD, Assistant Professor of Department of Electrical & Electronic Engineering (EEE), Brac University, for his relentless direction and support throughout this work. Without his guidance and consistent help this dissertation would not have been achievable. Also lastly , we dedicate our work to our parents. Without their prayers and hard works this would have been impossible for us to do.

Table of Contents

List of Tables

List of Figures

List of Acronyms

AI	Artificial Intelligence
DL	Deep Learning
MV	Machine Vision
ReLU	Rectified Linear Unit
FC	Fully Connected
CNN	Convolution Neural Network
AF	Activation Function

Glossary

Thesis:	An extended research paper that is part of the final exam process for a graduate degree. The document may also be classified as a project or collection of extended essays.
Glossary:	An alphabetical list of key terms This is an optional page and can be removed if not used. Use one table row for each item to allow sorting using Word's table tools. Apply the style **1_Para_NoSpace** to table rows as shown here.

Chapter 1

Introduction

1.1 Motivation: Deep Learning in disease detection

It is always Considered that Health is the real wealth and if anyone is physically unable to do any sorts of work, meanwhile you will mentally be abused to forward for the rest of anyone's life journey. Technological advancement considered for previous centuries turns its optimistic eye in the premises of novel healthcare with increasing number of new techniques of pathological extensive research in biological science. [4] The increasing ability of health sector documentation development of numerus data analysis module has made possible the successful application of AI in health sector.[16] Despite recent enrichment in healthcare of AI application, it focusses only few for their most extensive research. [16, Fig.1] Among them- cancer, nervous system, cardiac and most recently the COVID-19. [17]

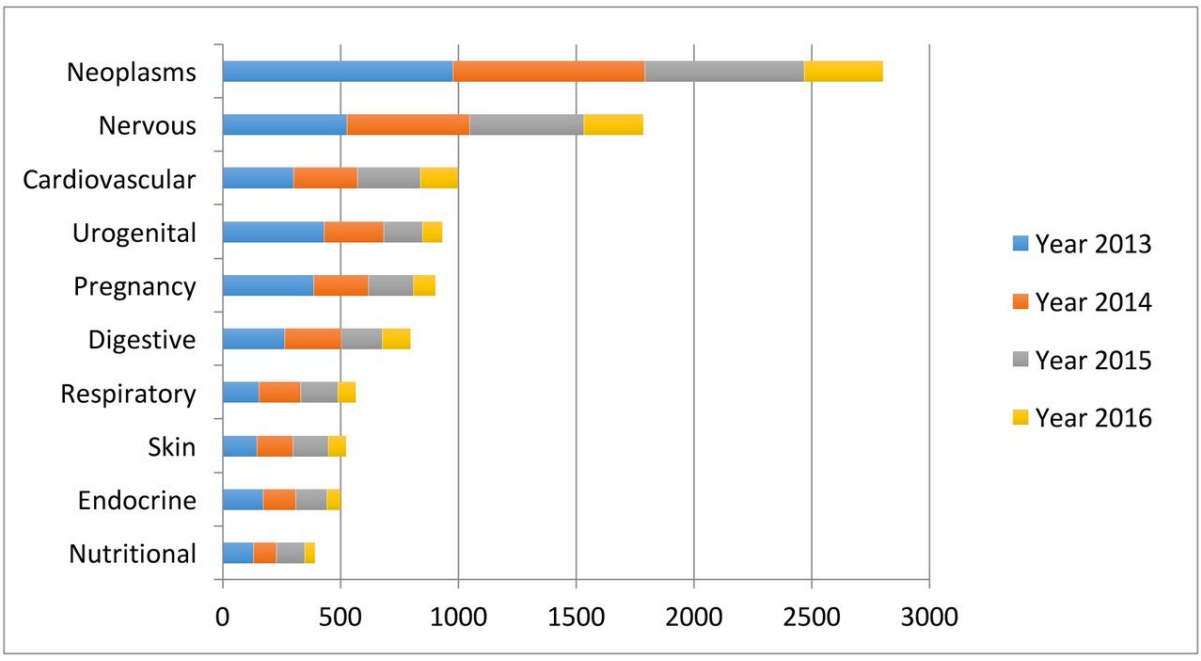


Figure 1: Leading 10 diseases considered in AI [16]

Not only that, AI system can play a vital role in assisting the physician with a view to providing latest medical and clinical information with proper patients care. Furthermore, AI system helps in reducing the errors of diagnostic tests and therapeutic reports which are considered inevitable in human clinical practices. [16] So, by being motivated with all these perspectives and

understanding the current COVID situation of the whole world we undertake a firm decision to design malaria and COVID-19 detection with more efficient way with the help of AI.

1.2 Objectives and Aims

Our aim is basically on COVID detection to help the physician understandable what necessary steps to be taken within the patients. Malaria detection is for comparing purposes as the system can detect which ever diseases it detects within the shortest period of time. Our main objective is to give the people for lower cost diseases detection and make a revolutionary change in diagnostic system of current situation within the country.

1.3 Background of Deep Learning

The fundamental concept of DL to be considered the proper analyzation of data with a view to increasing abstraction level of representation. This category of representation allows to comprehend data in such a manner of powerful process of learning and raw data complexity representation that makes useful implementation in various arena. [4] The DL algorithms stand on the underlying relationship comparing the information and decision making which is performed without any explicit instruction. [2] Many types of DL algorithms are developed for example: DNN ("Deep Neural Network"), RNN, Convolution Neural Network (CNN), DBN ("Deep Belief Network") and many more. In the eve of time, various exciting and amazing works of human sensory responses like visionary, speech recognition has been displayed with great potential in MV related task. [5] It is CNN which is holding the most spectacular algorithms for its best understanding the image context and prominent in performance in the field of image segmentation, classification of certain images, detection of particular images and analysis related duty. [6] CNN first come to light during 1989 for the purpose of processing of grid data and time order data and the architectural model was inspired by Hubel and Wiesel's basic structure of primate's visual cortex. [5] The most attractive properties of CNN are that it can spital or temporal correlation in data which incorporate into abundant learning stages and thus composed of a joint of different layers. [2] Nowadays, the enrichment of CNN crossed over academia. Many world best companies like Facebook recently developed new research group by exploring the model of CNN. [7] Finally, the availability of big data and advancement of hardware plays vital role for the recent success of CNN. Over the years, CNN have [17, Tab.1] undergone considerable research and advancement shown in below:[18]

Table-1.0: CNN over a timeline [17]

CNN Variant	Year of release	About
LeNet-5	1998	Standard template for all modern CNN
Alex Net	2012	Overlapping pooling and ReLU implementation
VGG-16	2014	Deeper CNN improve the efficiency
Inception-v1	2014	Introduced dense module instead of stacking
Inception-v3	2015	First algorithm with batch normalization
ResNet-50	2015	Deeper CNN without compromising Accuracy
Xception	2016	CNN based solely on deep-wise separable
Inception-v4	2016	Stem improvement and same no. of filters
Inception-ResNet	2017	Scaled up cardinality within module

1.4 Thesis Structure

In our paper, we basically focus on the different algorithm of deep learning such as Inception v3, VGG19, Xception, ResNet50 to predict our malaria and COVID model and finally we make comparison among the results of different model. Moreover, we will use flask app to implement the model whether our model can detect properly or not and lastly, we will finish our work by referring the future work and working area of detection many diseases through DL algorithm.

Chapter 2

Conceptual Overview

2.1 Architecture of CNN

A CNN architecture can be made up different layers which is organized by API. [2] Through the help of differential function each layer performs one volume of activation to another. The main types of layers are:

- Convolution layer
- Polling layer
- Fully connected layer

A special attention is required that here in all the four layers which contains numbers of parameters and hyperparameters. The impact of these parameter is that they act like a variable and optimized automatically during the training period. On the other hand, Hyperparameters are that kind of variables that needs to be set beforehand. [2] As below [1, Tab. 1] the different parameters and hyperparameters are shown:

Table-2.1: A list of Parameter and Hyperparameters in CNN [1]

Layers	Parameters	Hyperparameters
Convolution	kernel	Size of Kernel , kernel number, stride, padding, activation function
Polling	None	Pooling method, filter size, stride, padding
Fully Connected	Weights	Number of weights, Activation function
Others	None	Model architecture, optimizer, learning rate, loss function, mini-batch size, epochs, regulations, weight initialization, dataset splitting

In short, the architecture will be:

INPUT – CONV – POOL – FC

- **Input** (B*A*3) holds the raw image pixel values where B indicates the width, A is the height and 3 represents the three-color channels R, G, B.
- **Convolution layer** line up the feature and image patch which compute the output of neurons connected to the input regions. Then multiply each image pixel by the corresponding feature pixel and add them up. Numbers of filters are being used to this purpose. Finally, divide by the total number of pixels in the feature.
- **Pooling layer** performs the down sampling operation and shrink the image stack into smaller size. It picks a window size and a stride. Then allows to walk window across the filtered image and take maximum value from each window.
- **FC** is the final layer where the actual classification happens and compute the class score. Here the filtered and shrink images are taken and put them into a single value of list.

For better understanding, it is referred to [1, Fig.2] CNN architecture as following:

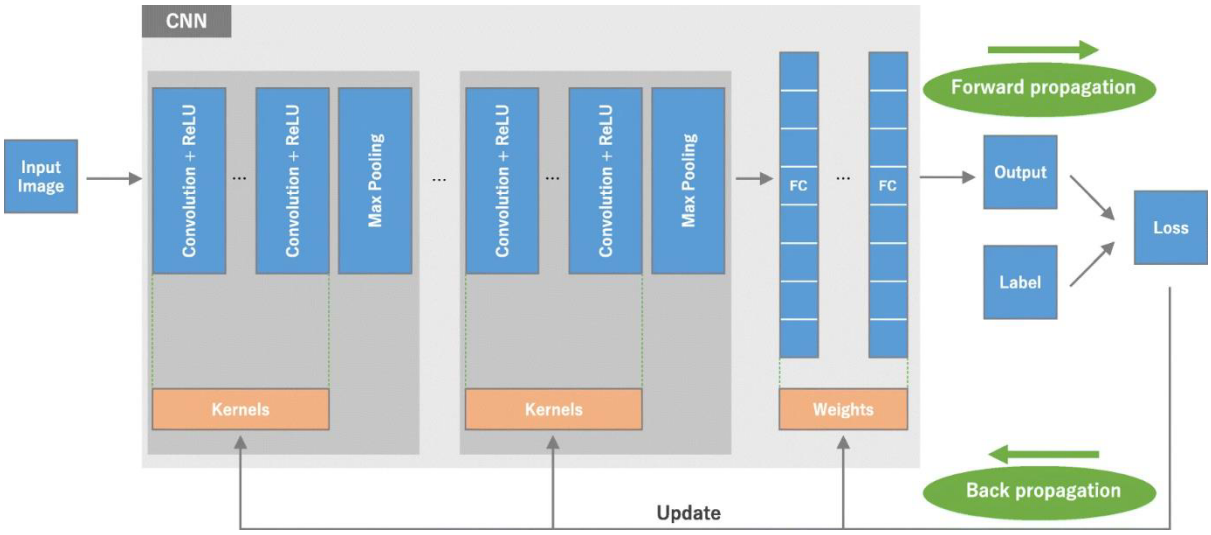


Figure 2: A description of the architecture of a convolutional neural network (CNN) and the training process. The stacking of several building blocks consists of a CNN: convolution layers, pooling layers (e.g., max pooling) and fully connected layers (FC) [1]

2.2 How does Computer read images

Computer usually reads an image as matrix of numbers between 0 to 255. For detecting colored image there is available three channels RED, GREEN & BLUE and each of the channels have associated with a matrix. Each channels of the matrix represents the intensity of the brightness

of the pixel. Again, for the greyscale or black and white image there is only one channel. [3] Color image size B*A*3; where 3 is the channels and B & A is the row and column of the matrix. Black and white image size B*A. The arbitrary machine-readable image [1, Fig.3] is shown here:

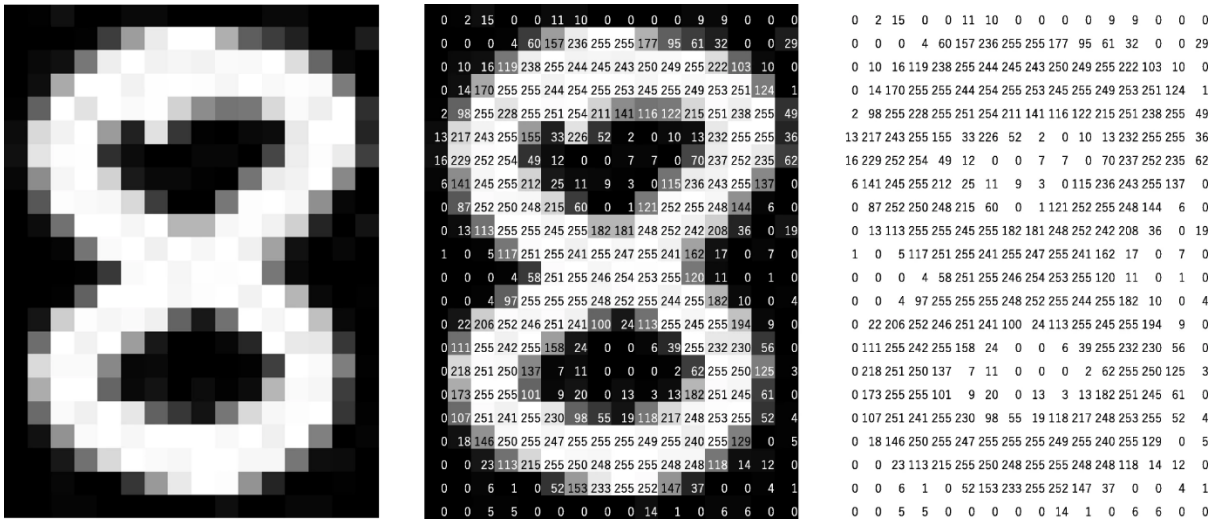


Figure 3: A picture is seen by a computer as an array of numbers. There are numbers between 0 and 255 in the matrix on the right, each corresponding to the pixel brightness of the left image. [1]

2.3 Basic CNN Components

A typical CNN generally consist of alternate layer of convolution followed by polling layers and finally one or more fully connected layer in it. [2] The arrangement of CNN components plays an important role in designing different regulatory units like batch organization and drop-out and thus enhance the performance. Here, in this section all the components of CNN architecture are discussed in a whole along with their features and working area.

2.3.1 Convolution Layer

This layer basically consists of different set of convolution kernels where each neuron acts like a kernel and performs feature extraction of many linear and nonlinear operations. Convolution kernels work by dividing the images into small pieces and the division of small blocks helps in extracting the features. [8] Moreover, kernel which make spiral with the images using specific

set of weights by multiplying the elements with corresponding elements of the respective fields. [2] Kernel is basically consisting of tiny order array of numbers which applies in input called a tensor. The output tensor is called feature map which is obtained by an element-wise product between each element of the kernel. Here, the input tensor which can be calculated at each location of the tensor and finally to obtain the output value of the corresponding positions, it is required to be summed up. [1] The two most important hyperparameter in convolution operation are often considered its shape and numeral number of kernels in it. The typical form is that 3x3, sometimes it may differ from 5x5 and 7x7 and determines the depth of arbitrary output feature maps.

Now, another key hyperparameter called padding which is typically called zero padding and it adds the zeros of rows and column on each side of the input tensor. This hyperparameter fits the middle of a kernel of the keep the same plane dimension and the outermost part throughout the convolution operations. [1, Fig.4]

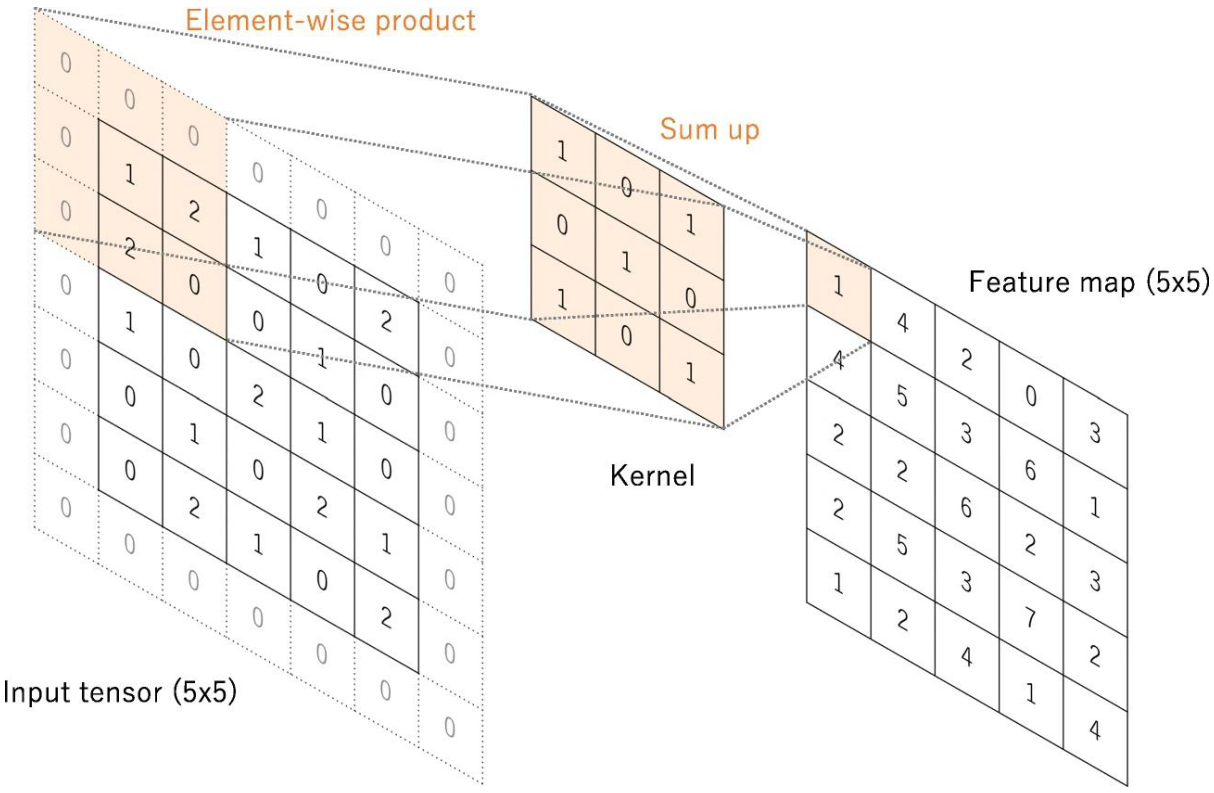


Figure 4: A zero-padding convolution operation [1]

Here in this particular example [1, Fig.4) The output function map is stored in the the input tensor 5x5. The kernel size and stride are kept 3x3 and 1 respectively. The distance between

two successive positions of the kernel is called stride which is also the hyperparameter od convolution operations. The most common choice of stride is 1, but sometimes it may be bigger than 1 for the purpose of down sampling. [1]

2.3.2 Non-Linear Activation function

The linear operation's convolution outputs now passed through a non-linear activation function. Activation function refers to the node of a CNN that is put at the end or in between of a neural network. [8]

The activation function is the nonlinear transformation that we do over the input signal. This transformed output is then sent to the next layer of neurons as input." — Analytics Vidhya

There are various kind of activation function [9, Fig.5] described below:

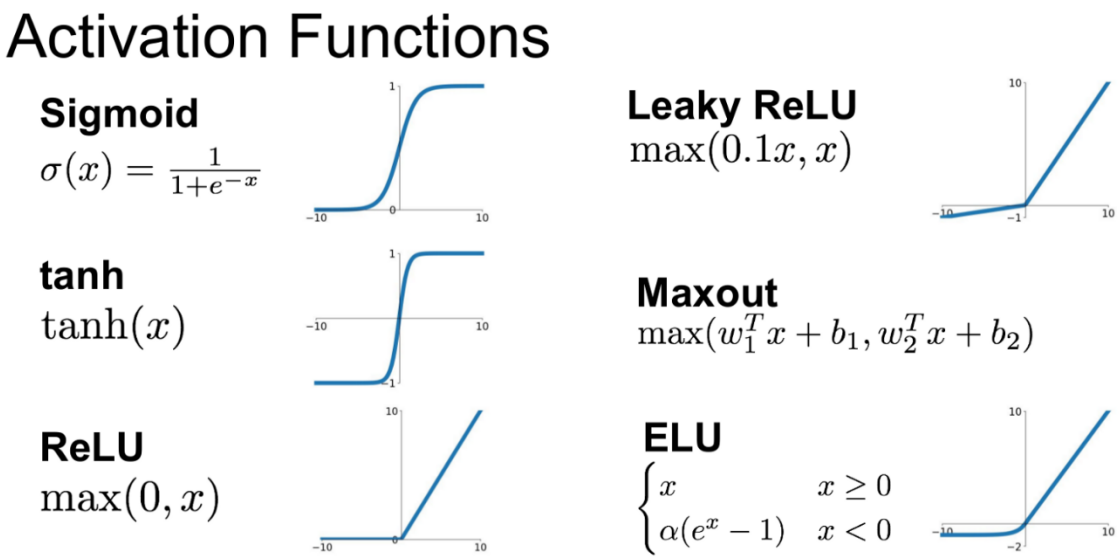


Figure 5: Activation Function [9]

Sigmoid: It is usually used in logistic regression. X is weights multiplied by input features plus bias. And this x is transferred to activation function. Once it is transformed, this will transform the value between 0 and 1. That is what a sigmoid function does. 0.5 is the threshold. Below that it is considered as 0 and above 0.5 it is considered as 1. Output of 1 indicates that the

neuron is activated and that means that neuron is transferring the signal and helping you to classify the final output.

ReLU: Similar to sigmoid, X is weights multiplied by input features plus bias. And this y is transferred to activation function.

If X is a negative value then the max is 0

If X is a positive number then the max is X

This is much more popular than sigmoid as it is faster and there is no gradient saturation problem.

Whenever we do the backpropagation, the derivative for X>0 will be 1 and the derivative for X<0 will be 0.

In regression problem statement we should try to use relu. In classification problem we can use ReLU in the middle layers but in the final layer we have to use sigmoid.

The only problem is that it gets saturated at the negative region that means gradient that region is zero.

Leaky ReLU: In order to solve the dead ReLU Problem, the negative part of ReLU is set to 0.01X instead of 0. This prevents the dying ReLU problem. Backpropagation is possible even if the input values are less than zero as there is a small positive gradient in the negative area.

However, it has not been fully proved that leaky ReLU is better than ReLU as the results were not consistent.

2.3.3 Pooling layer

This layer is sustained between convolution layers and performs down sampling operations. The function map's in-plane dimensionality is reduced in such a way that signify a translate invariance to small portion and distortions. Not only that, it also decreases the subsequent learning rate. [1] The most critical thing is there is no learnable variable in any of the pooling layers but other hyperparameters such a padding, stride, filter size is available just like convolution layer.

There are two operations in Pooling layer. One is Max-pooling and other is the Global average pooling. First, we discuss about Max-pooling:

Max pooling is a discretization process where sample values are calculated using a max filter to find the maximum value in the pool to down sample an input, reducing its dimensionality and to make decisions about the features based on the down sampled subsamples.[8]

Suppose we have a 4x4 matrix representing our input and a 2x2 Max filter to be used for down sampling

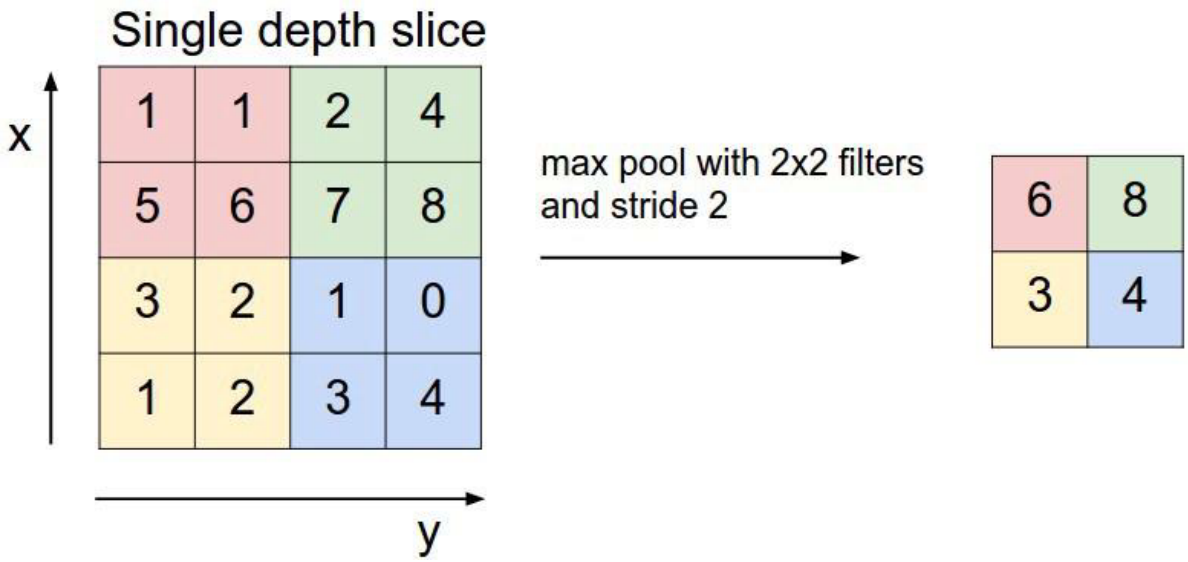


Figure 6: Max=pooling [1]

Example: As the most common pooling operations are called max pooling, patches are extracted from input feature maps and the maximum value is output in each patch. [1, Fig.6]

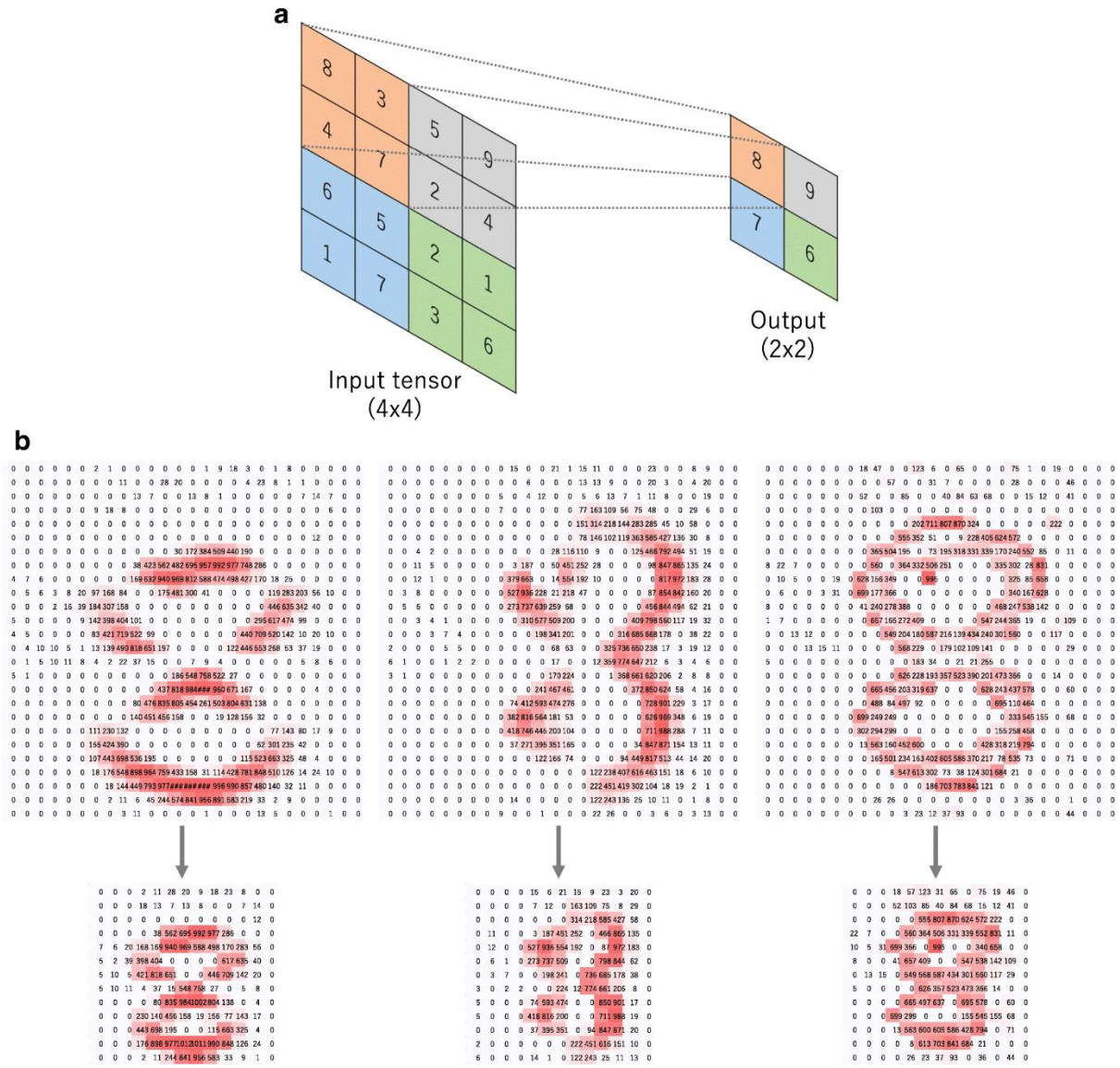


Figure 7: An example of pooling operation [1]

Here [1, Fig.6(a)], the filter size of 2x2, no padding, stride of 2. This operation extracts patches 2x2 from input tensor and maximize the output in each patch by discarding all the other values. The next one [1, Fig.6(b)], max pooling operations in the same image. The upper rows are down sampled by the factor of 2, from 26x26 to 13x13. [1]

Global average pooling: This kind of pooling is used for extreme down sampling where feature map with size of height x width is down sampled into 1x1 array by simply taking the

average of all the elements of each feature map. [1] This is rarely done once before entering into fully connected layer. Several advantages are being corporate within it, like:

Reduces the learning rate

Enables the CNN to accepts the input variable size

2.3.4 Fully Connected Layer

This is the last phase of a CNN network and the neurons have complete connection to all the activation from the past layers. By following bias offset with matrix multiplication, all the activation can be measured. [8] In the convolution or pooling layer, the output function maps a one-dimensional number array and is linked to single or more fully-connected layers. This functionality is called dense layer. [1] Finally, fully-connected layer produce as the equal amount of output nodes comparing the same number of classes [9, Fig. 7]. Actually, it is CNN which basically an architecture of hidden layers and fully-connected layer.

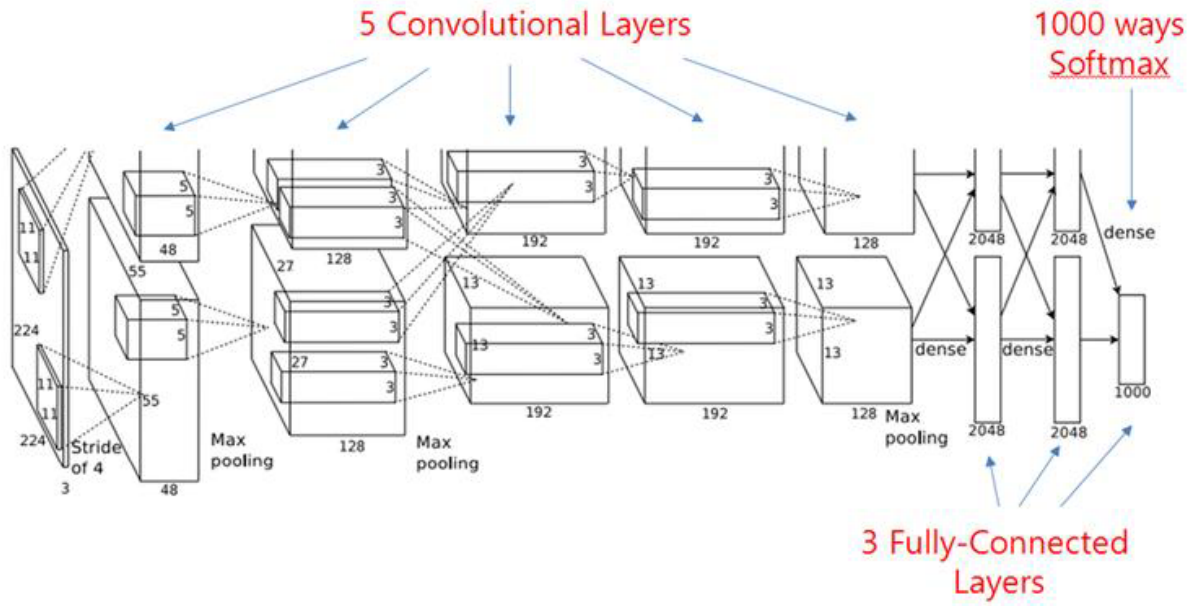


Figure 8: Fully-connected layer [9]

Each fully connected also followed by some non-linear activation function [9, Fig. 5] as we described before. But the most important issue is that the last layer fully-connected activation function.

These are very distinct from others since they normalize specific output values from the last completely connected layer to the probabilities of the target class. The value range must be between 0 and 1 and sum of the all values must be 1. [1] Typical choice for the last layer activation functions in accordance to their area of working is shown below: [1, Tab. 2]

Table-2.2: Commonly applied last layer activation function [1]

Task	***Activation Function***
Binary Classification	Sigmoid
Multi-class single class classification	SoftMax
Multi-class multi class classification	Sigmoid
Regression to continuous value	Identity

In our particular case, we basically use Sigmoid and SoftMax for the layer activation function. As we previously discussed about Sigmoid and now the brief description of SoftMax given below:

SoftMax makes the outputs for each category between 0 and 1, and divides by their sum for giving the probability of the input value being in a specific class.

$$S(xj) = \frac{e^{xj}}{\sum_{k=1}^{k} e^{xj}} \ , \ j = 1,2,\ldots.k \ldots\ldots\ldots\ldots (2.1)$$

x_j is [1, eq. (2.1)] weights multiplied by input features plus bias.

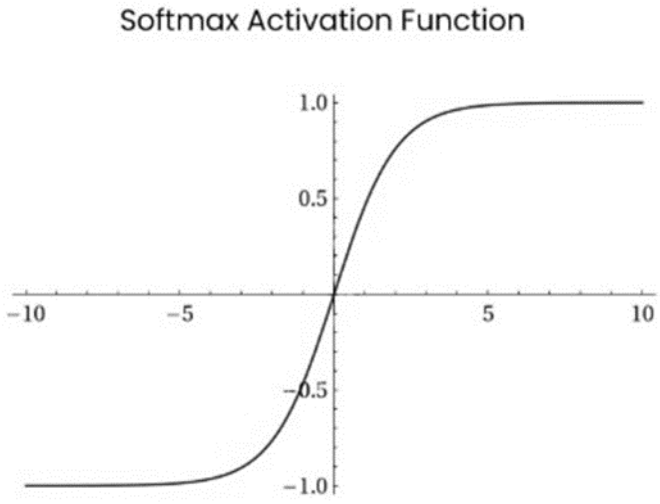


Figure 9: SoftMax AF

2.4 Training of CNN

Training a network significance that step of CNN architecture where training dataset are provided with the finding's kernel in convolution layers and weights in fully connected layers. [1] The weights are being made computer readable through a training process for adjusting the filter values and it is called backpropagation. [10] It is backpropagation which is mostly used in training the neural network whereas loss function and gradient decent optimization plays the main role in the algorithms. Step by step discussion about all these parameters and their functionality briefly described from next.

2.4.1 Backpropagation

It contains four distinct section:

- ✓ Forward pass
- ✓ Loss function
- ✓ Backward pass
- ✓ Weight updating

The training image which consists of a particular number of array pass through the whole network. For the first case of training, weights are generally in random initialization and the output doesn't give any preference to a reasonable conclusion. [10] As the network fails to develop the classification what is expected to be, it goes to the loss function back propagation.

2.4.2 Loss Function

This is referred as a cost function which makes a balance between the output and prediction through forward pass. [1] The most common case for defining the loss function is MSE (mean square error) [1, eq. (2.2)] which actually is:

$$\mathbf{E_{tot}} = \overline{\boldsymbol{\Sigma}} \frac{1}{2} [\mathbf{T} - \mathbf{O}]^2 \quad \text{...} \quad (2.2)$$

Where, E represents error, T is the actual target value and O is the predicted output value.

If we want to make out prediction right that means the predicted level be same as training level, we need to minimize the amount of loss we have: we need to find the weights that directly contribute to the loss of the network. [10]

Now the time is to perform the backward pass to find the weights which are responsible for the great loss to the network. Hence, the various kind of weight initialization techniques discussed for further understandings:

There are several weight initialization techniques in deep learning training such as- uniform distribution, Xavier (or Glorot) initialization, He initialization and so on. Modern deep learning always prefers advances initializers like Xavier and He as their performance is over pure random and definitely all zeros-initializations. [11]

- **Xavier Uniform (Glorot):**

$$\mathbf{W_{Xu}} = \left[-\frac{\sqrt{6}}{\mathbf{f_{in}+f_{out}}}, \frac{\sqrt{6}}{\mathbf{f_{in}+f_{out}}}\right] \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad (2.3)$$

  This initializer [12, eq. (2.3)] performs well with Sigmoid AF.

- **He Uniform:**

$$\mathbf{W_{Hu}} = \left[-\sqrt{\frac{6}{\mathbf{f_{in}}}}, \sqrt{\frac{6}{\mathbf{f_{in}}}}\right] \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad (2.4)$$

This method [11-12, eq. (2.4)] works well with ReLU activation function.

Once the calculation is done for weight initializing, the network now moves to the last step of weight update. All the weights of the filters have taken in such a way that the updating weights results in the opposite direction of the gradient. [10]

2.4.3 Gradient descent

This parameter is commonly used for minimizing the loss of the network by using an optimization algorithm such as kernels and weights in order to update learnable parameter. [1,5] Based on a hyperparameter called learning rate [10, Fig.9] the gradient of loss function enables us to provide the negative direction of gradient with an arbitrary step size. [1] The mathematical expression [1, eq. (2.5)] of gradient decent as below:

$$\boldsymbol{\omega} = \boldsymbol{\omega} - \boldsymbol{\alpha} * \frac{\partial \boldsymbol{L}}{\partial \boldsymbol{\omega}} \quad (2.5)$$

Where,

ω = weight (learnable parameter), α =learning rate, L=loss function

Note that, learning rate which play a vital role in training has to be set before the training starts. The next portion discussion will be about this hyperparameter.

2.4.4 Learning rate

Changing our weights too fast means moving the steps that are too large. As a result, we might skip over the best value for a given weight. This can make it hard to reduce the loss function. We use a changeable function called "The learning rate" that is just a tiny number, usually take 0.001 that can be multiplied the gradients by to scale them. This gives the surety that any changes we make to our weights are pretty low.

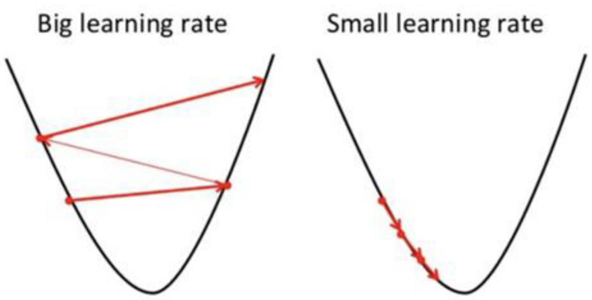


Figure 10: Learning Rate

Taking too small steps is not efficient either because then we might never end up with the right values for our weights. This might lead to our optimizer concentric to a local minimum due to the loss function, but not the global minimum. The learning rate assures that whatever change made to our weights so that the step size is neither too large or too little. The gradient descent optimization with learning rate [1,3, Fig.10] displayed here:

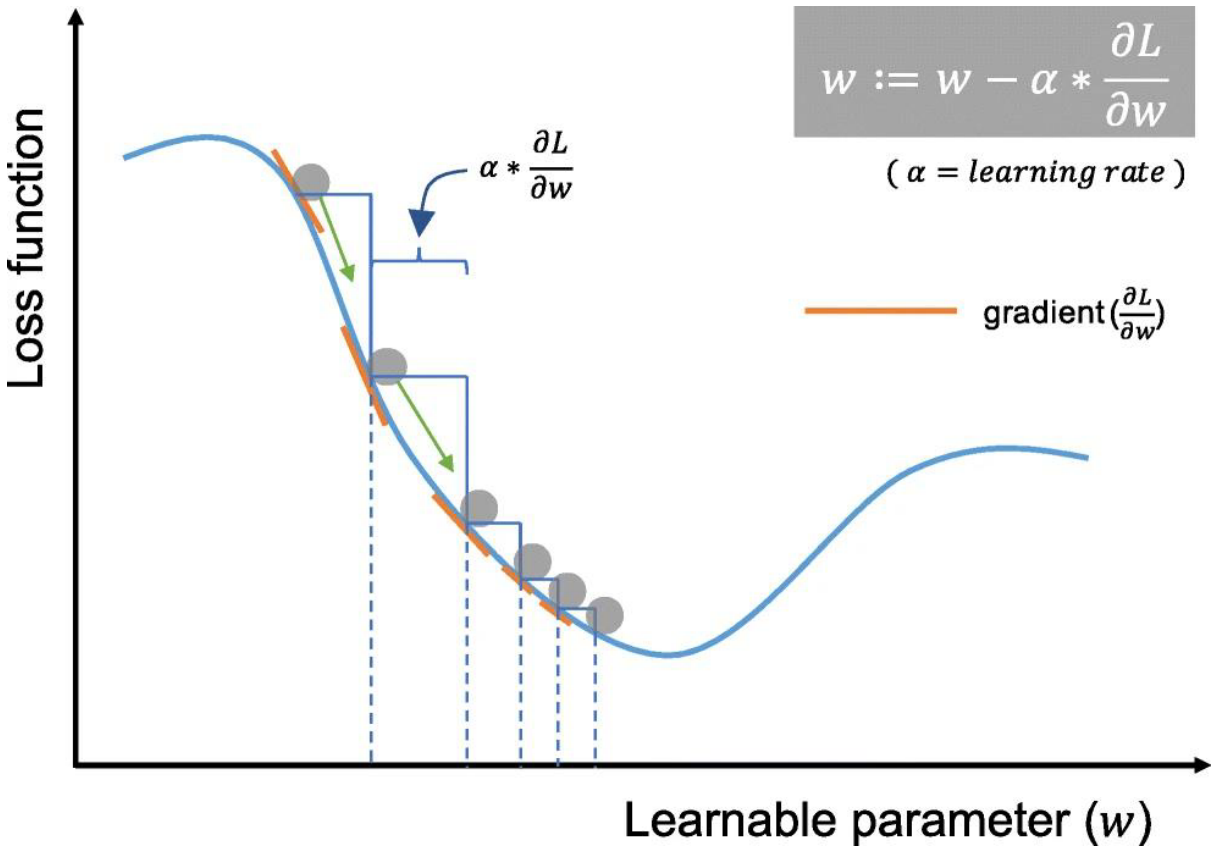


Figure 11: Gradient descent with optimization algorithm **[3]**

In practical life, because of the shortage of memory the gradient of loss function is being calculated by using mini-batch which is considered to be subset of the training dataset for updating different parameter. [1,5] Another highly effective gradient descent called stochastic (SGD) used widely in this field for training dataset in CNN. Now, this two hyperparameter briefly discussed as below:

Stochastic Gradient Descent: In Gradient Descent we were considering all the examples for every step of Gradient Descent. But if our dataset is very big then using gradient descent the sample will have to update the weights of all the records in each epoch. This is not efficient. As a solution, we use Stochastic Gradient Descent. In Stochastic Gradient Descent (SGD), the record is take at each move. We do the following steps in one epoch for SGD:

1. Take a record
2. Feed it to Neural Network
3. Calculate its gradient

4. Use the gradient we calculated in step 3 to update the weights

5. Repeat steps 1–4 for each record in training dataset

As we use one record at a time the cost will fluctuate over the training examples and it will not necessarily decrease. Be that as it may, over the long haul, you will see the cost diminishing. Likewise, it will never hit the minima in light of the fact that the cost is so fluctuating, however it will keep on moving around it. For bigger datasets, SGD might be utilized. When the dataset is huge, it converges quicker. The benefit of SGD is that it's computationally more affordable. Very big datasets consistently can be held in RAM as one single record of the dataset is entered in every time.[12]

Minibatch SGD: Since in SGD we use only one record at a time, this can slow down the computations. A mixture of Gradient Descent and SGD is used called Minibatch SGD. We do not use the entire dataset at once, nor do we use a single example at a time. A batch of a fixed number of training records is used. We do the steps in each epoch:

1. Pick a batch-size

2. Feed it to Neural Network

3. Calculate the mean gradient of the mini-batch

4. Use the mean gradient we calculated in step 3 to update the weights

5. Repeat steps 1–4 for the mini-batches we created

Much like SGD, in mini-batch gradient descent, the average cost over the epochs fluctuates since we average a small number of examples at a time. So, when we are using the mini-batch gradient descent we are updating our parameters frequently as well as we can use vectorized implementation for faster computations.[12]

If we have considered 10k records,

- GD:
  Epoch 1 → 1 Iteration (all the data)
- SGD:
  Epoch 1 → 10k Iterations (1 iteration for each record)

- Mini batch SGD:

  Epoch 1 → no. of Iterations = 10k / Batch size

  Batch size = specified

Now a days, many more improvement has been happened in Gradient descent algorithm and few of the proposed and widely used advanced algorithm are given for the better learning purposes: [1]

Stochastic Gradient Descent with Momentum: In SGD there is a lot of noise. SGD with momentum is method which helps remove this noise and leading to faster converging It is one of the most common optimization algorithms and it is used to train many state-of-the-art models.[12]

Adam: Adam is an optimization algorithm that can be used instead of the classical stochastic gradient descent procedure to update network weights iterative based in training data. [13] The attractive benefits of this algorithm are as:

- ✓ Straightforward to implement
- ✓ Computationally efficient
- ✓ Less memory required
- ✓ Invariant to diagonal rescale of gradient
- ✓ Well suited for problems that are large in terms of data and/or parameters
- ✓ Appropriate for non-stationary objectives
- ✓ Appropriate for problems with very noisy/sparse gradients
- ✓ Hyper-parameters have intuitive interpretation and typically require little tuning.

The process of forward pass, loss function, backward pass, and parameter update is one training iteration. The program will repeat this process for a fixed number of iterations for each set of training images (commonly called a batch).[10] Once finishing the parameter update on the last training example, hopefully the network should be trained well enough so that the weights of the layers are tuned correctly.

2.5 Data Augmentation

It is basically the step where invariance in Dataset has created. This data augmentation actually helps to create many more types of images with the same output. Thus, the size of dataset increases by flipping, horizontal shifting, vertical shifting Zooming and so on. [3,4]

In python, creating image dataset using data augmentation ***Keras*** is being used. The function called ***imageDataGenarator*** [7] perform the main role for implementation. It is obvious that in deep learning there is required to learn the features automatically from the dataset. This is only possible when there is a lot of training data available input samples are very high-dimensional such as images. CNN which is basically the pillar of deep learning algorithm are designed in such a way even though there is only a little data available from. ***imageDataGenarator*** class configure random transformation and normalization operation during image data training.

2.6 Algorithms we used

2.6.1 Inception v3

Inception V3 CNN base deep neural network with 48 layers of module and can convolute 1x13x3 and 5x5 convolution. [50] Inception v3 could classify images into a total of 1000 categories, including keyboard, pencil, mouse, and many other animals. This model was trained on more than one million images from the ImageNet database. [18] The fine-tuning architecture of Inception V3 [51, Fig.12] given below:

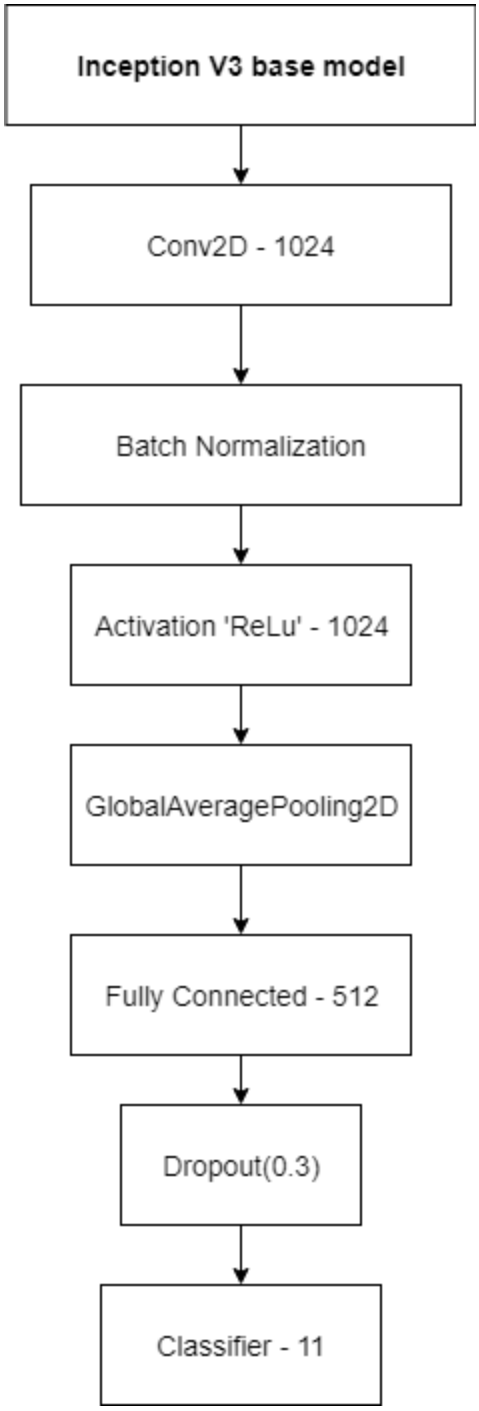


Figure 12: Inception V3 architecture [51]

2.6.2 Algorithm using VGG19

VGG-19 has 13 convolutional and 3 fully-connected layers. It used ReLUs as activation functions, just like in AlexNet. VGG-19 had 138 million parameters. A deeper version, VGG-19, was also constructed along with VGG-16. The Architecture is shown below: [51, Fig.13]

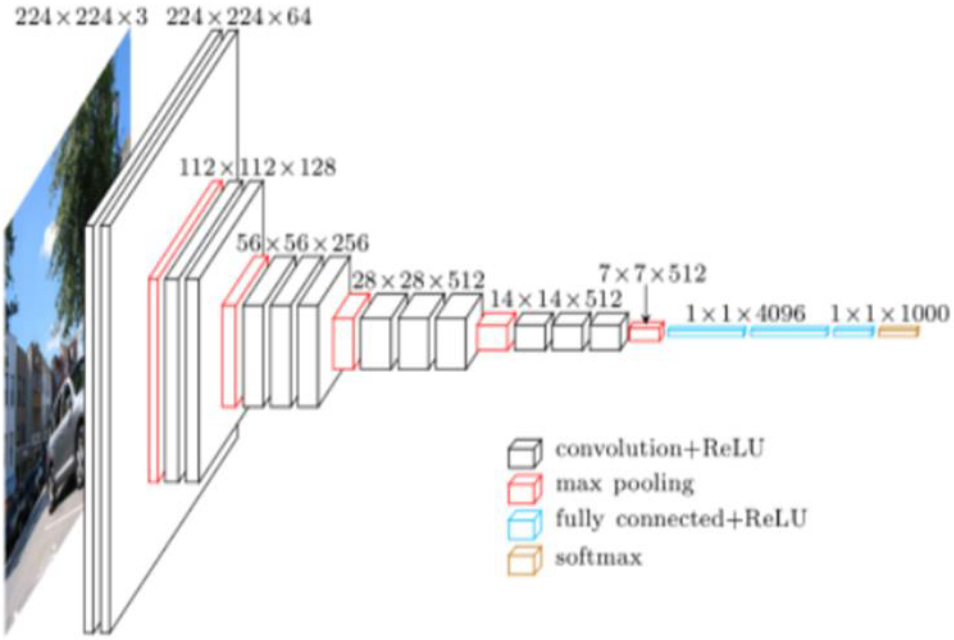


Figure 13: VGG19 Architecture [51]

2.6.3 Algorithm using Xception

Xception was 71 layers deep and had 23 million parameters. It was based on Inception-v3. Xception was heavily inspired by Inception-v3, albeit it replaced convolutional blocks with depth-wise separable convolutions. Xception [50, Fig.14] practically is a CNN based solely on depth-wise separable convolutional layers. [18]

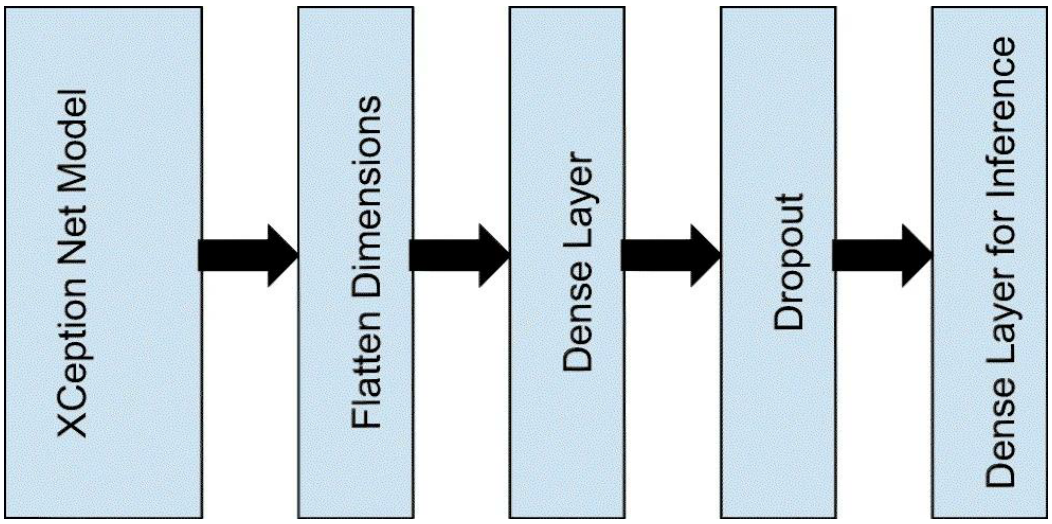


Figure 14: Xception architecture [50]

2.6.4 Algorithm using Resnet50

At 50 layers deep and sporting 25.5 million parameters, ResNet-50 [17] was trained on more than a million images from the ImageNet dataset. t was a combination of Inception v4 and ResNet-50. Scaled up cardinality within a module. [52, Fig.15]

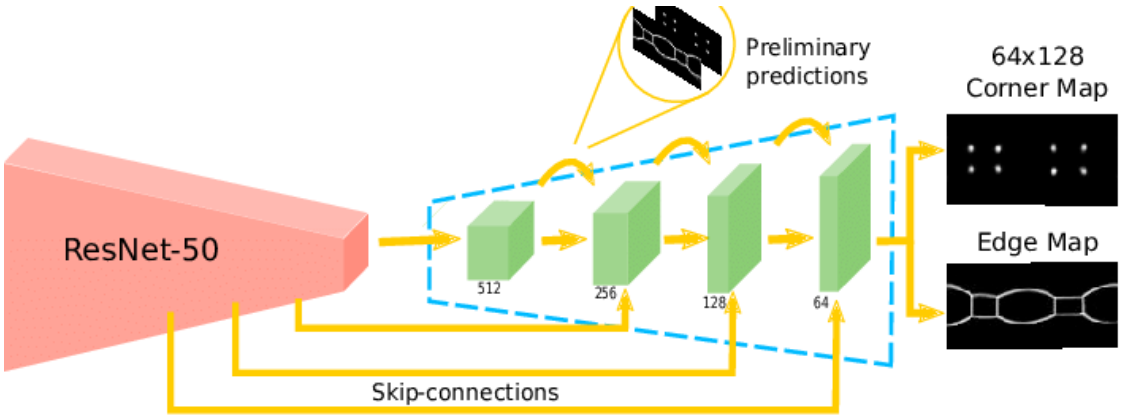


Figure 15: Resnet50 architecture view [52]

2.7 *Metrics* for performance evaluation

2.7.1ROC curve

ROC is short for Receiver Operating Characteristic. A ROC graph shows a classification's performance. [14, Fig.16] Two parameters are plotted by the ROC curve -

True Positive Rate or TPR

and

False Positive Rate or FPR.

TPR is also known as recall. [14]

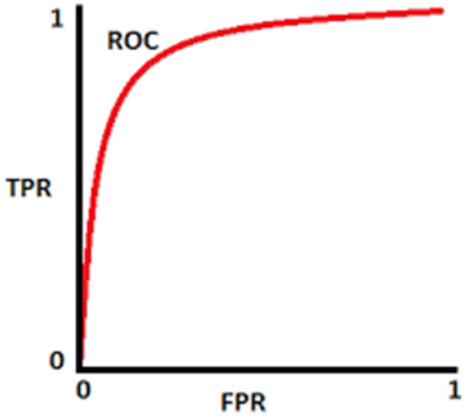


Figure 16: ROC curve [14]

An ROC curve plots true positive and false positive along y and x-axis respectively and lowering the classification threshold classifies more items as positive, thus increasing FP and TP.

The Area under the curve or AUC [14, Fig.17]is a measure of the accuracy of the model.

The more area is under the curve, the more is the accuracy.

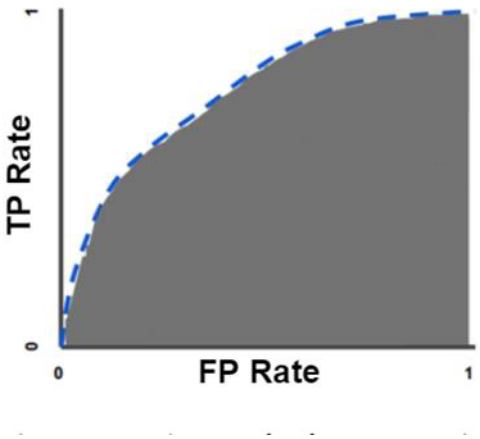


Figure 17: AUC model [14]

2.7.2 Confusion Matrix

Confusion matrix helps in determining the predictive capability of a model.

A: TP = True positive

B: FN = False negative

C: FP = False positive

D: TN = True negative

We should focus on getting a higher percentage on regions A and D as they determine the predicted result to be correct.

Table-2.3: Confusion Matrix 2-class representation

		Predicted Class	
		Class= YES	Class= NO
Actual Class	Class= YES	A TP	B FN
	Class= NO	C FP	D TN

2.7.4 Accuracy

Accuracy of a model can be determined from the confusion matrix using the following formula [15, eq. (2.6)]:

$$\text{Accuracy} = \frac{A+D}{A+B+C+D} = \frac{TP+TN}{TP+TN+FP+FN} \quad (2.6)$$

Accuracy lies between 0 to 100 percentage. Although Accuracy is used in many branches, it is not a correct way of evaluation.

Let's consider a 2-class problem.

- Number of class 0 examples = 990
- Number of class 1 examples = 10

Here the accuracy is 990/1000 = 99.9%

Accuracy is misleading because the model does not detect any class 1 example. It predicts everything to be class 0.

2.7.5 Cost of Prediction

If the prediction of the model is weighted against a cost matrix and compared, we can determine the cost of the classification problem.

Cost Matrix	Predicted Class		
	C(i/j)	+	-
Actual Class	+	-1	100
	-	1	0

Model M1	Predicted Class		
	C(i/j)	+	-
Actual Class	+	150	40
	-	60	250

Model M2	Predicted Class		
	C(i/j)	+	-
Actual Class	+	250	45
	-	2	200

Accuracy = 80%

Accuracy = 90%

Cost = 150*-1+40100+60*1+250*0

Cost = 250*-1+45*100+2*1 +200*0

Cost = 3910

Cost = 4255

2.7.6 Cost-sensitive Measures

All the parameters under this are given below [15, eq. (2.7), eq. (2.8), eq. (2.9), eq. (2.10), eq. (2.11), eq. (2.12), eq. (2.13), eq. (2.14)]:

$$\text{Precision}(p) = \frac{A}{A+C} \quad \ldots\ldots (2.7)$$

$$\text{Recall}(r) = \frac{A}{A+B} \quad \ldots\ldots (2.8)$$

F-measure (F)= $\frac{2rp}{r+p} = \frac{2a}{2a+b+c}$…… (2.9)

Weighted Accuracy = $\frac{w1a+w4d}{w1a+w2b+w3c+w4d}$…….....(2.10)

True positive rate (TPR) = $\frac{A}{A+B}$ [Sensitivity]…… (2.11)

True Negative rate (TNR) = $\frac{D}{C+D}$ [Specificity]…… (2.12)

False positive rate (FPR) = $\frac{C}{C+D}$……..... (2.13)

False Negative rate (FNR) = $\frac{B}{A+B}$…… (2.14)

2.8 Flask development

To demonstrate how robust and compatible our models are we developed an app using Flask.

[54, Fig. 18] shows how flask interacts with a ML model and to demonstrate how robust and compatible our models are we developed an app using Flask.

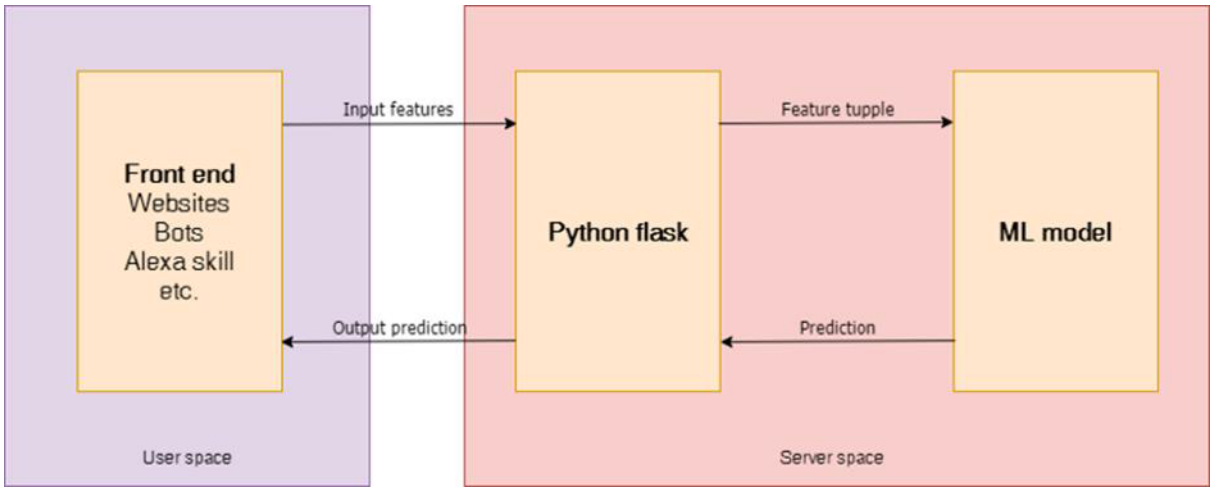


Figure 18: Model Deployment [54]

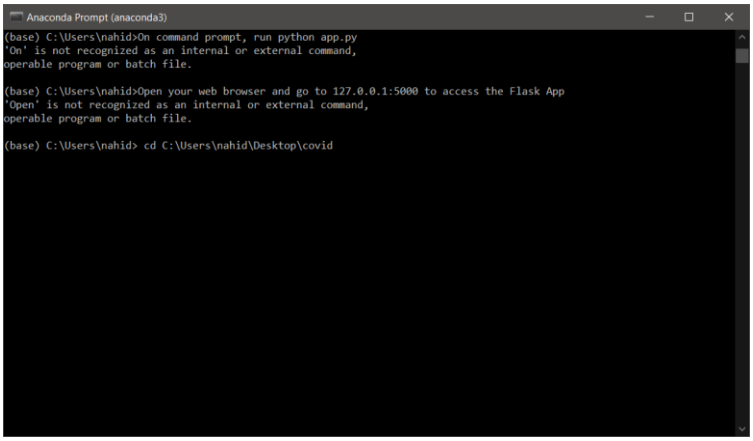
Figure (18) shows how flask interacts with a ML model and Front-end Website bots.

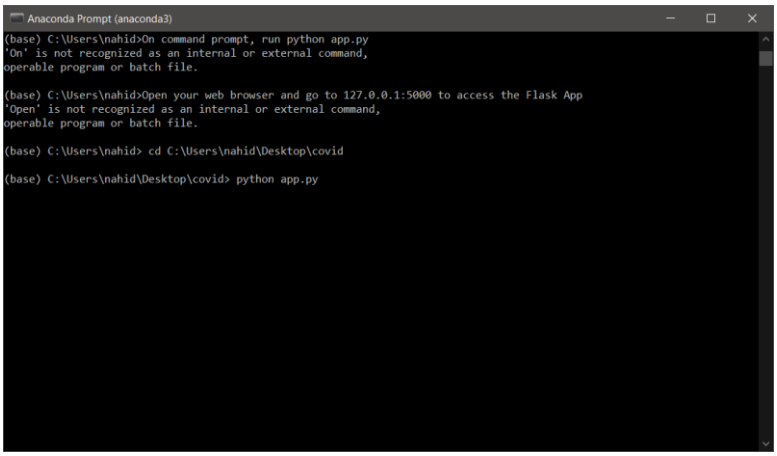
Our model sends predictions to the flask app. Since it is a research work, we ran the service in a local machine instead of deploying online. Figure below shows the predictions of the model in our local environment.

Our model sends predictions to the flask app. Since it is a research work, we ran the service in a local machine instead of deploying online. Figure below shows the predictions of the model in our local environment.

How to use Flask App

- Download the entire folder that contains the static files and the html files too. The base.html and index.html are needed for the webpage. They are present in the template folder. app.py and the uploads folder should be present in the root folder too. The uploads folder is where the images are stored once anyone uploads for prediction.
- On command prompt, run **cd** followed by the path of the root folder location
- Then type **python app.py** and press enter.
- Open your web browser and go to **127.0.0.1:5000** to access the Flask App





Chapter 3

Case-1 Deep Learning Algorithm in Malaria Detection

In this chapter we see how we worked on several deep learning models for predicting malaria from a blood cell image. In the subsequent subchapters we will explain in detail how we compiled, trained our model, evaluated the model, compared the results and deployed it in a more user-friendly system using flask app.

3.1 About Malaria

Malaria is a major epidemic infectious disease caused by a protozoa Plasmodium transmitted by female Anopheles mosquitoes. It is one of the deadliest diseases in Bangladesh and many other developing countries. As a result of this, developing countries have to face a lot of hardship in dealing with the infected people. Around 400,000 people die from malaria per year. According to World Health Organization (WHO), in 2019, approximately 229 million cases of malaria were detected and 409,000 people lost their lives due to malaria worldwide.[1] Malaria is curable and can be prevented if proper initiatives are taken. In this era of advanced technology, biomedical research plays a vital role to reduce the mortality rate by malaria.

One of the obstacles faced in fighting the disease is the insufficient malaria diagnosis. There is a huge amount of data in medical sectors. Most are computerized. However, they are not put to any use. If studied and analyzed the data could be really useful. For instance, if we could diagnose and treat malaria in the early stage it can reduce the disease and prevent deaths.

3.2 Detecting Malaria

There are many techniques to detect malarial host in a patient. For example, clinical diagnosis, microscopic diagnosis, rapid diagnosis test (RDT) and polymerase chain reaction (PCR).
The most efficient way of malaria diagnosis is RDT and microscopic diagnosis methods. RDT is effective as it proves that no trained professional is required or microscope and may provide diagnosis within 15 min. However, drawbacks of RDT are quite a few. WHO and others claim that RDT is not sensitive enough, cannot quantify parasite density? RDT is also more expensive than light microscope and easily damaged by heat. These limitations are not present in microscopic systems but microscopic systems need a trained professional.
Usually, clinical diagnosis and PCR are done in laboratories where human judgment is needed for accurate results. However, in villages and remote areas laboratories and skilled professional are not so readily available. As a result, the efficiency and accuracy are very low.

Modern systems can use deep learning algorithms for image analysis. Automatic malaria parasite detection which only needs an image of the blood sample of the patient to detect malaria would be a great solution.

In this study we trained and implemented multiple accurate and computationally efficient Convolution Neural Network (CNN) models for malaria parasite detection in cell images using a publicly available dataset.

3.3 Relevant work

Scientists all over the world research malaria since it is such a deadly disease. Previously malaria was diagnosed in labs but the concept of deep learning has become one of the most sophisticated approaches to computer vision and machine learning.[43] Machine learning automatic systems were initially researched like [42.43] used SVM and Principal Component Analysis (PCA) for classification. However, Deep learning models have produced more accurate results than the Machine learning models. Researchers liked to use CNN for automatic detection of malaria from microscopic images of the blood cells.

Recently, Dong et al. found the performances of 3 famous convolution neural network and compared CNN with SVM. They found that CNN is better because of its ability to learn images features automatically.

We propose several deep learning models whose performance is comparable to the previously reported highly accurate deep learning models. Moreover, our models are efficient in terms of computational resources and can be easily used in smart phones in areas that are remote at a very low cost.

3.4 Dataset

In order to conduct a series of experiments we used a publicly available dataset. We used 4 convolution networks to find out the model with the maximum accuracy possible. The data collection, data processing, techniques, are discussed in the following subsections. The dataset used in this work is taken from National Institute of Health [20]. There are 535 cell images, with 321 parasitized and 214 uninfected segmented red blood cell images. Positive samples contained plasmodium and negative samples contained no plasmodium. The negative samples however could contain impurities or other staining artifacts. Giemsa-stained thin blood smear slides from 150 P. falciparum infected and 50 healthy patients were collected and photographed

at Chittagong Medical College Hospital, Bangladesh. The images were manually annotated by an expert slide reader at the Mahidol-Oxford Tropical Medicine Research unit.

The cell images are of 3-channels (RGB). Their size varies between 110-150 pixels which we later re-sampled to 124 x 124 output dimension, a channel depth of 3 to suit the input requirements of different classification algorithms we used. Various pre-processing techniques are also applied. This helped us achieve faster convergence which has been discussed in [Fig. 19(a), Fig.19(b)] show some samples from the dataset containing uninfected and parasitized segmented red blood cells.

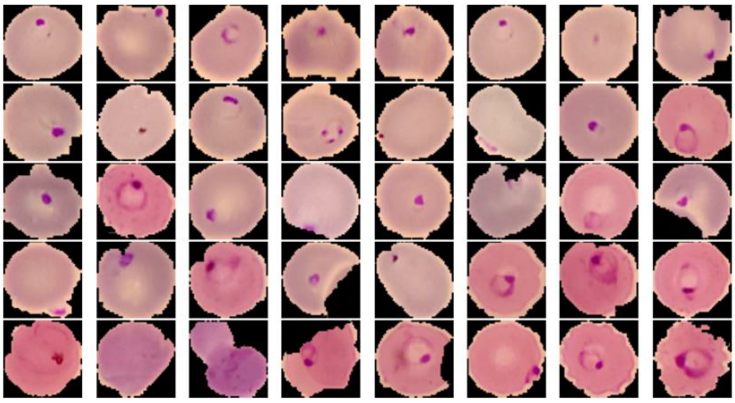


Figure 19(a): Positive Malaria Sample [20]

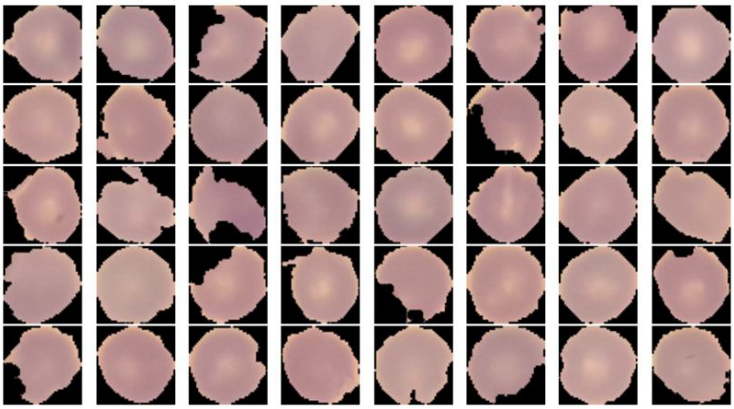


Figure 19(b): Negative Malaria Sample [20]

Figure 19: Malaria Sample Images

3.5 General Procedure

The subsequent sub-sections under 3.5 describe the steps we used in running our models. We used the similar process using InceptionV3, ResNet50, VGG19 and Xception.

3.5.1 Training and Test split

We used a ratio of 80 20 that is we used 80% of the data for Training and 20% for Test. We ran 500 epochs, each with a batch size of 32 images.

3.5.2 Building the model

Image size are resized to 224 by 224 pixels before being fed into the model because the images in our dataset were of different sizes. Then we added the tensor shape as 224 by 224 by 3 px where 3 is the number of channels.

Next, we used a Flatten layer and a Dropout layer to flatten al our features and to overcome overfitting respectively. In the last layer, we used a Dense output layer using a SoftMax Activation Function. This ensures that we have 2 classes in the output as either parasitized or uninfected.

We made the trainable attribute of the previous layer as False since first part of the model is already trained. In the end we used Adam optimizer to compile the model using categorical crossentropy as the loss function.

The code for adding custom layers, for example, to the Xception model is shown below. The code for the rest of the models is the same. Only the Xception in the first line needs to be changed into the name of the desired model.

```
xception = Xception(weights="imagenet", include_top=False,
    input_tensor=Input(shape=(224, 224, 3)))

outputs = xception.output
outputs = Flatten(name="flatten")(outputs)
outputs = Dropout(0.5)(outputs)
outputs = Dense(2, activation="softmax")(outputs)

model = Model(inputs=xception.input, outputs=outputs)

for layer in xception.layers:
    layer.trainable = False

model.compile(
        loss='categorical_crossentropy',
        optimizer='adam',
        metrics=['accuracy']
)
```

As we used the SoftMax function (2.3.2) for the final layer, you can see the last layer has 2 outputs.

```
__________________________________________________________________________________________________
concatenate_1 (Concatenate)     (None, 5, 5, 768)    0           activation_91[0][0]
                                                                 activation_92[0][0]
__________________________________________________________________________________________________
activation_93 (Activation)      (None, 5, 5, 192)    0           batch_normalization_93[0][0]
__________________________________________________________________________________________________
mixed10 (Concatenate)           (None, 5, 5, 2048)   0           activation_85[0][0]
                                                                 mixed9_1[0][0]
                                                                 concatenate_1[0][0]
                                                                 activation_93[0][0]
__________________________________________________________________________________________________
flatten (Flatten)               (None, 51200)        0           mixed10[0][0]
__________________________________________________________________________________________________
dropout (Dropout)               (None, 51200)        0           flatten[0][0]
__________________________________________________________________________________________________
dense (Dense)                   (None, 2)            102402      dropout[0][0]
==================================================================================================
Total params: 21,905,186
Trainable params: 102,402
Non-trainable params: 21,802,784
```

3.5.3 Training the model

We trained our model using Xception had a training accuracy of 98.38% after 500 epochs. We did the same using InceptionV3, Vgg19 and ResNet50.

```
.3/13 [==============================] - 5s 391ms/step - loss: 0.6825 - accuracy: 0.9894 - val_loss: 0.4649 - val_accuracy: 0.
:poch 491/500
.3/13 [==============================] - 5s 394ms/step - loss: 0.6841 - accuracy: 0.9798 - val_loss: 0.3937 - val_accuracy: 0.
:poch 492/500
.3/13 [==============================] - 5s 396ms/step - loss: 0.7882 - accuracy: 0.9686 - val_loss: 1.1974 - val_accuracy: 0.
:poch 493/500
.3/13 [==============================] - 5s 393ms/step - loss: 0.9324 - accuracy: 0.9521 - val_loss: 0.7084 - val_accuracy: 0.
:poch 494/500
.3/13 [==============================] - 5s 401ms/step - loss: 0.6698 - accuracy: 0.9583 - val_loss: 0.3587 - val_accuracy: 0.
:poch 495/500
.3/13 [==============================] - 5s 392ms/step - loss: 0.2520 - accuracy: 0.9897 - val_loss: 0.3227 - val_accuracy: 0.
:poch 496/500
.3/13 [==============================] - 5s 392ms/step - loss: 0.2192 - accuracy: 0.9867 - val_loss: 0.9379 - val_accuracy: 0.
:poch 497/500
.3/13 [==============================] - 5s 393ms/step - loss: 0.6253 - accuracy: 0.9753 - val_loss: 0.4623 - val_accuracy: 0.
:poch 498/500
.3/13 [==============================] - 5s 396ms/step - loss: 0.2468 - accuracy: 0.9835 - val_loss: 1.0893 - val_accuracy: 0.
:poch 499/500
.3/13 [==============================] - 5s 400ms/step - loss: 0.4542 - accuracy: 0.9747 - val_loss: 0.6666 - val_accuracy: 0.
:poch 500/500
.3/13 [==============================] - 5s 398ms/step - loss: 0.4186 - accuracy: 0.9838 - val_loss: 0.5900 - val_accuracy: 0.
```

3.6 Results and Discussion

We ran the trained models on the test set and plotted the confusion matrices, ROC curves, classification reports, Accuracy and Loss curves. These are discussed in detail in the following parts.

3.6.1 Confusion matrix

A confusion matrix is a table that is often used to describe the performance of a classification model as we have learnt from chapter 2. The rows in a confusion matrix corresponds to what the machine learning algorithm has predicted. In our case one will be the patient has malaria and the other does not have malaria. The top left corner contains true positives. These are patients who have malaria and detected correctly by the algorithm. And the bottom right corner has the 'True Negatives'. These are patients who did not have malaria and the algorithm correctly identified them for not having malaria. The left-hand corner contains the 'False negatives'. False Negatives are when a patient has malaria, but the algorithm says they don’t. And the top right corner has the False positives. the false positives are patients that do not have malaria but the algorithm says they do.

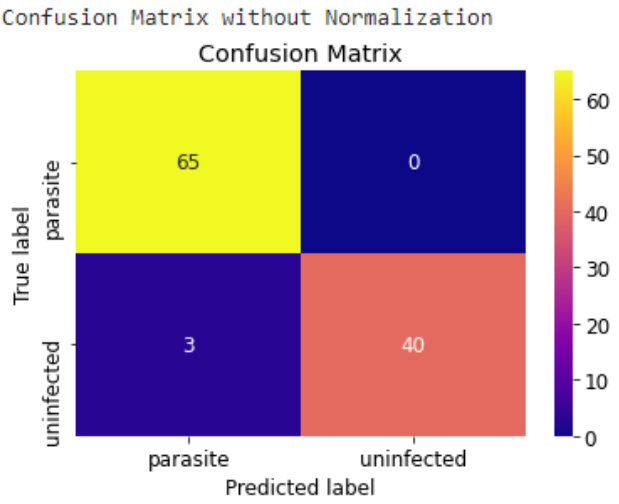


Figure 20a: Confusion Matrix of InceptionV3

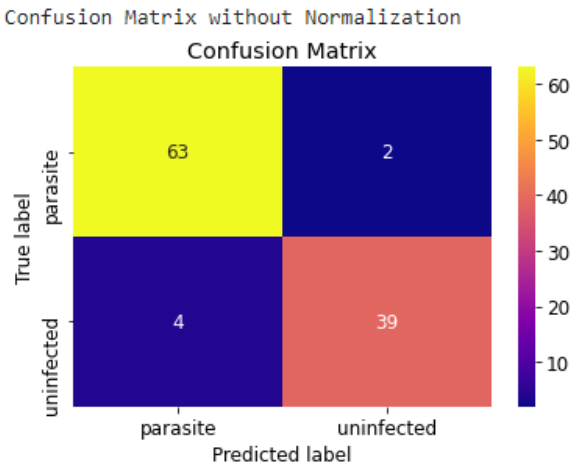


Figure 20b: Confusion Matrix of Vgg19

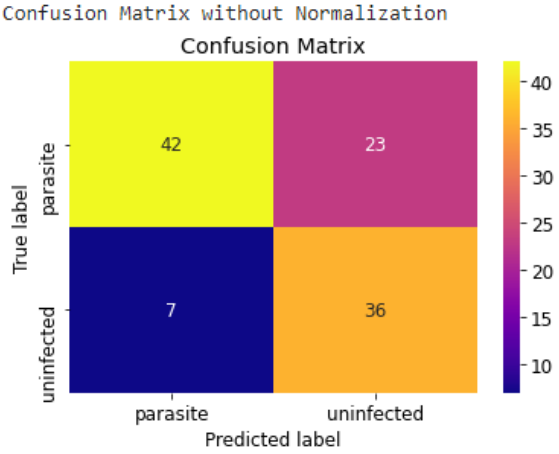


Figure 20c: Confusion Matrix of Resnet50

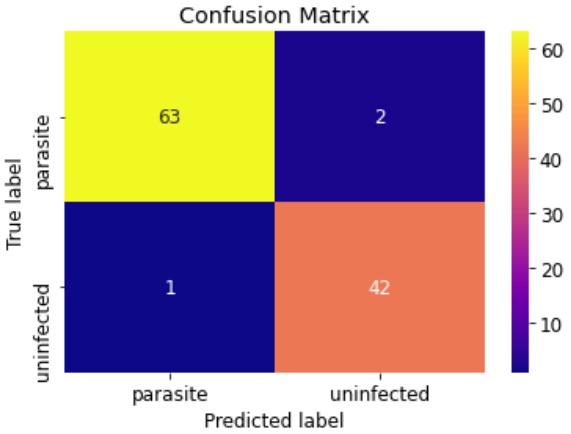


Figure 20d: Confusion Matrix of Xception

Figure 20: Comparison of Confusion Matrix without normalization

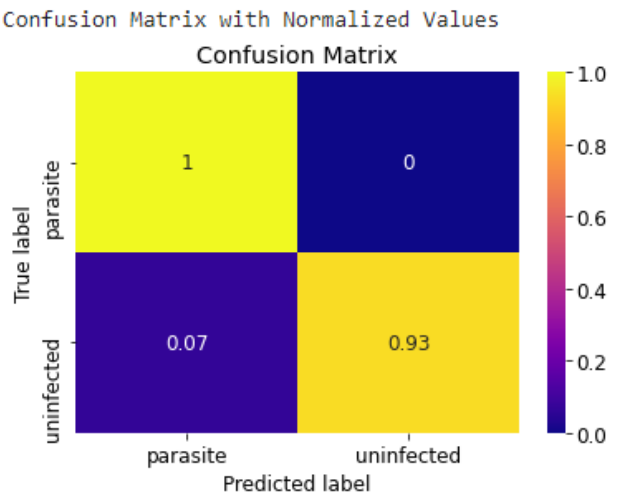


Figure 21a: Confusion Matrix of InceptionV3

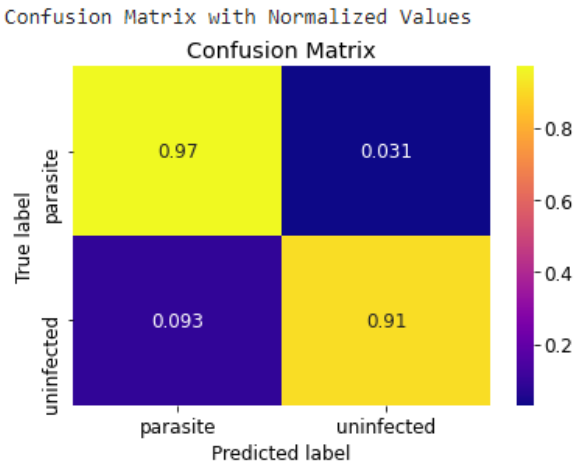


Figure 21b: Confusion Matrix of Vgg19

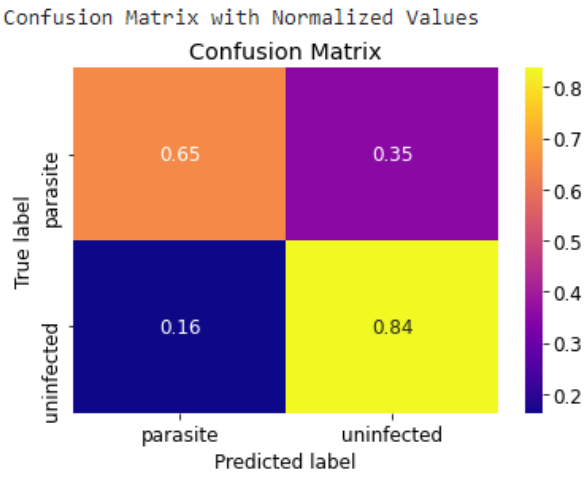


Figure 21c: Confusion Matrix of Resnet50

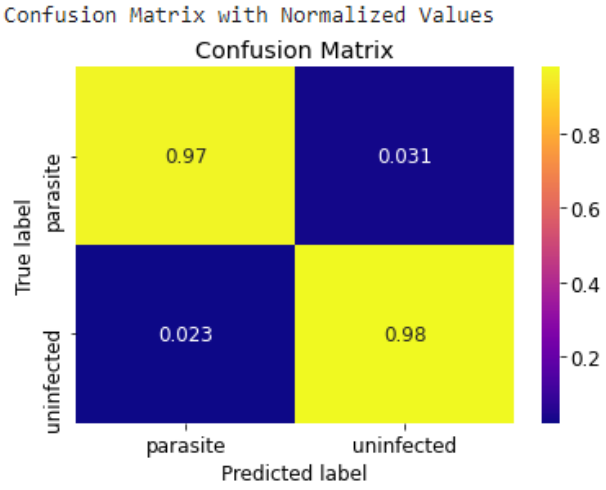


Figure 21d: Confusion Matrix of Xception

Figure 21: Comparison of Confusion Matrix with normalization

Here we can see in Figure 20a which is confusion matrix without normalization using Inception V3, we took 108 samples to test. Among them, our trained model could detect 65 of the affected patients correctly. And 40 unaffected patients were also detected correctly by our model. And our model misclassified 0 patients having malaria as unaffected. Only 3 patients who were uninfected were identified as affected by our algorithm. For normalized values, our trained model could detect 100% of the affected patients correctly. And 93% of the unaffected patients were also detected correctly by our model. And our model misclassified 0.0% of the patients having malaria as unaffected. Only 7% of the patients who were uninfected were identified as affected by our algorithm. This is shown in figure 21a.

In the same way if we interpret the other matrices, we can see that InceptionV3 has the best confusion matrix as it has 0 false positives. The Y axis represents True positive rate and X axis represents Predicted label. Similar to the Previous article, Xception and Inception have the highest true positive ratio whereas ResNet50 has the lowest. We can see that Inception and Xception have similar accuracy but Inception has 0 false positives, that is, no Parasitized patients are considered to be unparasitized which is the most important factor as many people may die by not getting treatment for the disease.

3.6.2 ROC curve

From chapter 2, we have learnt about the commonly used evaluation metrices. An ROC curve or Receiver Operating Characteristic Curve is a graph which displays the performance of a classification model at all classification thresholds. ROC curve plots two parameters- True Positive Rate and False Positive Rate. True Positive Rate or TPR is also known as recall and False Positive Rate or FPR; True Negative or TN

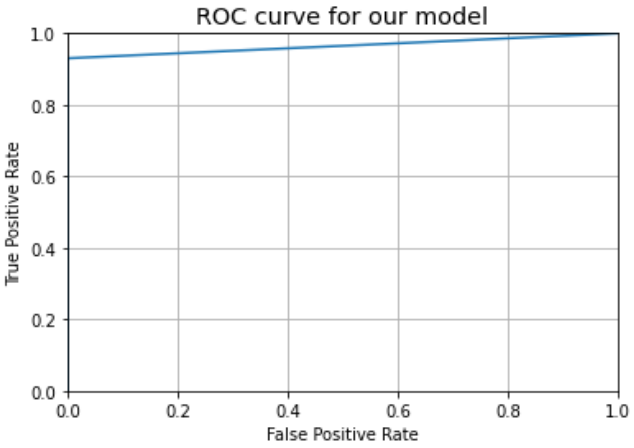


Figure 22a: ROC curve of InceptionV3

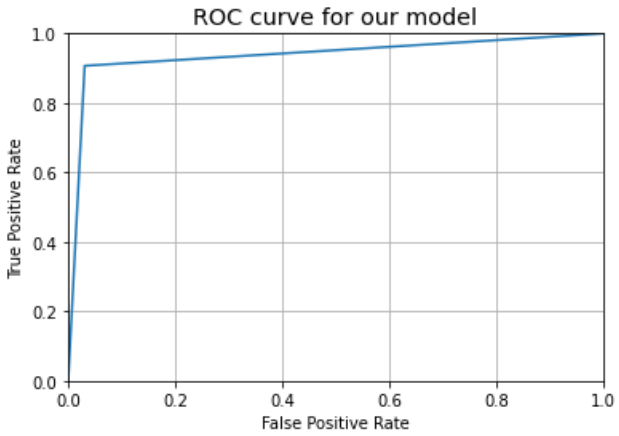


Figure 22b: ROC curve of Vgg19

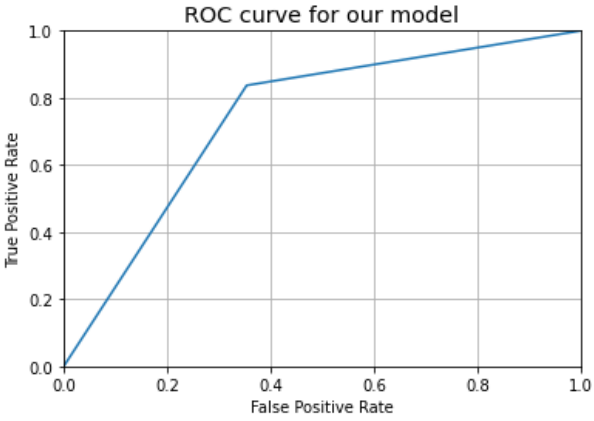


Figure 22c: ROC curve of Resnet50

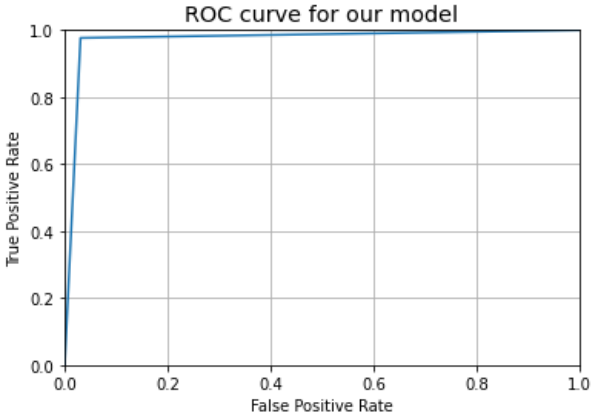


Figure 22d: ROC curve of Xception

Figure 22: Comparison of ROC Curves

We plotted the ROC curve. Along the X axis of ROC curve, False Positive Rate or FPR is represented and along Y axis True Positive Rate or TPR is represented. The ROC curve shows the trade-off between sensitivity (or TPR) and specificity (1 – FPR). Classifiers that give curves closer to the top-left corner indicate a better performance. As a baseline, a random classifier is expected to give points lying along the diagonal (FPR = TPR). The closer the curve comes to the 45-degree diagonal of the ROC space, the less accurate the test.

In the above ROC curve of InceptionV3 model that is Figure 22a we can see that the curve is very far away from the 45-degree diagonal of the ROC space which indicates that the curve is very accurate. The Area Under Curve or AUC is used as predictive accuracy measure for ROC curves and in our case, it is very high which indicates that our model is very accurate.

From the ROC curves in Figure 22, we can see that the Xception model has the highest Area under curve as the True positive rate starts at almost 1 for false positive rate of 0.01. We can compare that to VGG19 and Inceptionv3 where the AUC is quite similar. ResNet50 model has low training time and also has the lowest accuracy in our case as the AUC is the least among the 4 models.

3.6.3 Model Accuracy

Accuracy is the number of correct predictions. The basic accuracy function is $A = (TP + TN)/(TP + TN + FP + FN)$. In our algorithm the accuracy curve is formed by taking both test and train data and calculating the accuracy after each epoch. The blue line describes the training accuracy and the yellow line describes the test accuracy.

In the model accuracy curves. The test curve follows the train curve which happens in all cases therefore showing us that the model is working properly. Although there are some spikes in the curve, it is not to be considered and if we find the mean of the curve, we can see that test curve is closer to the train curve in Xception model.

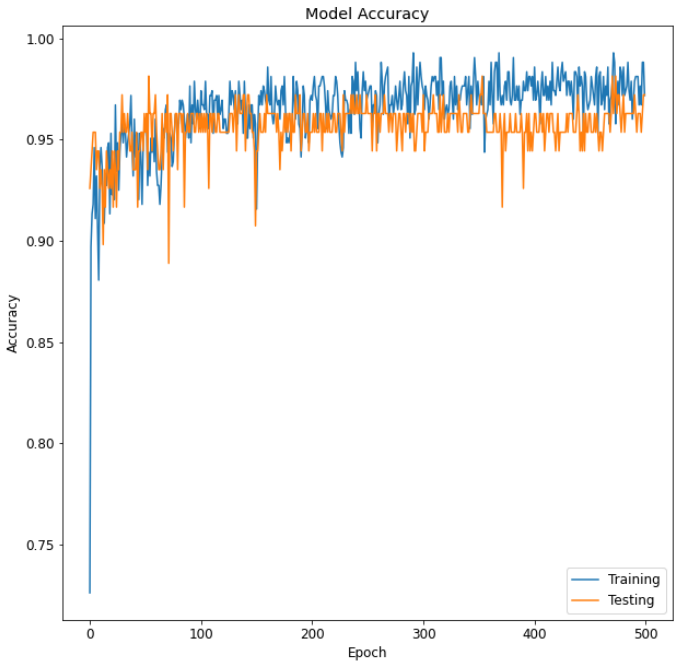


Figure 23a: Model Accuracy of InceptionV3

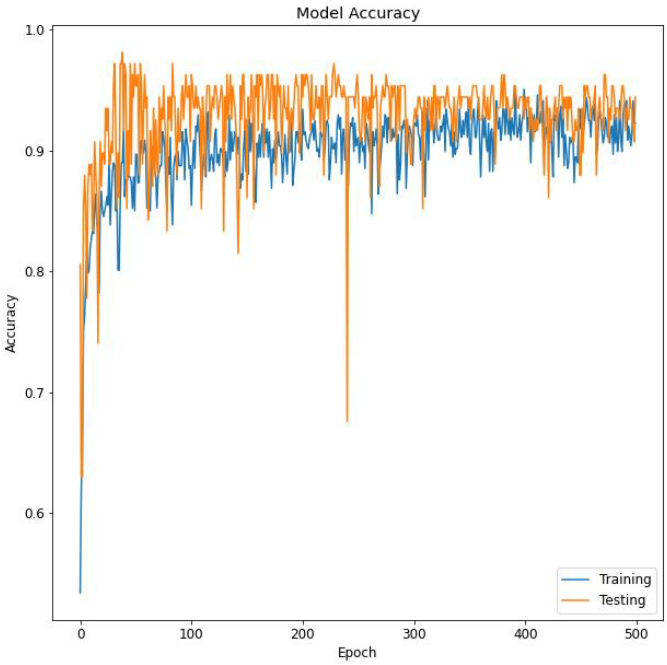


Figure 23b: Model Accuracy of VGG19

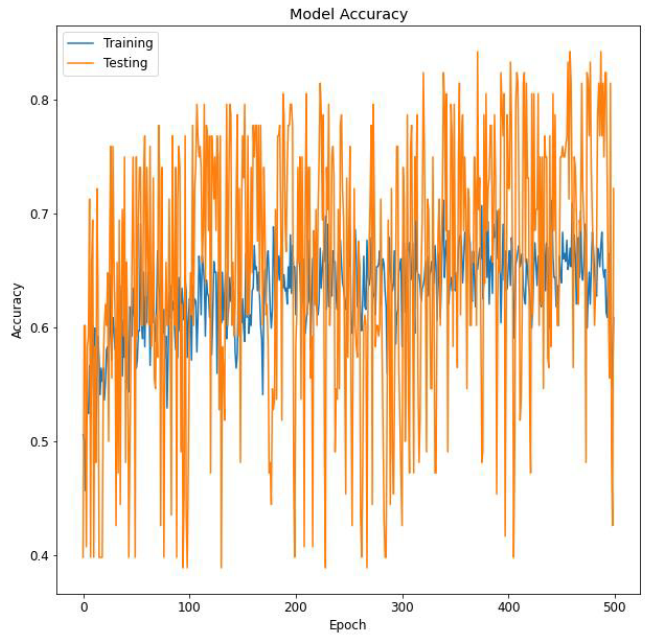


Figure 23c: Model Accuracy of Resnet50

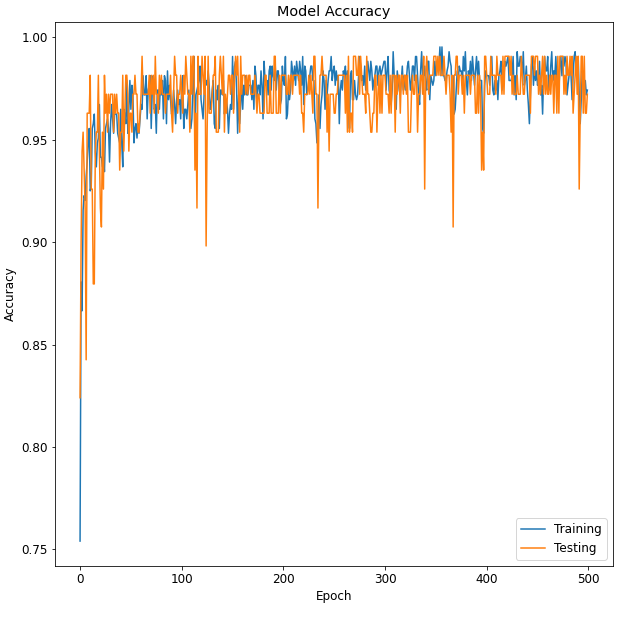


Figure 23d: Model Accuracy of Xception

Figure 23: Comparison of Accuracy curve

3.6.4 Model Loss Comparison

Loss is the penalty for a bad prediction. That is, loss is a number indicating how bad the model's prediction was on a single example. If the model's prediction is perfect, the loss is zero; otherwise, the loss is greater. In the model loss curves, we can find more about how the testing and training process is taking place.

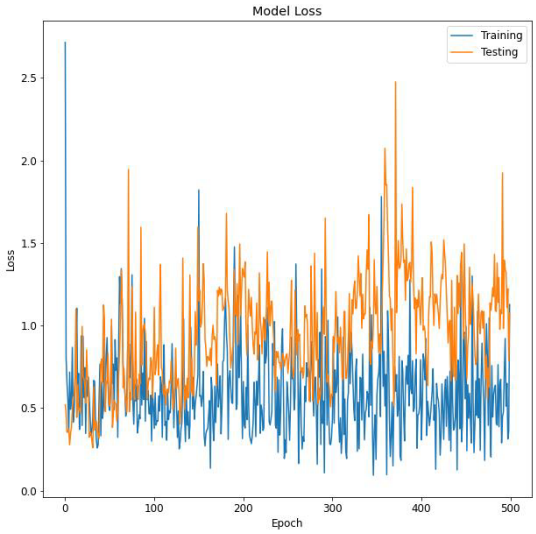


Figure 24a: Model Loss of Inception

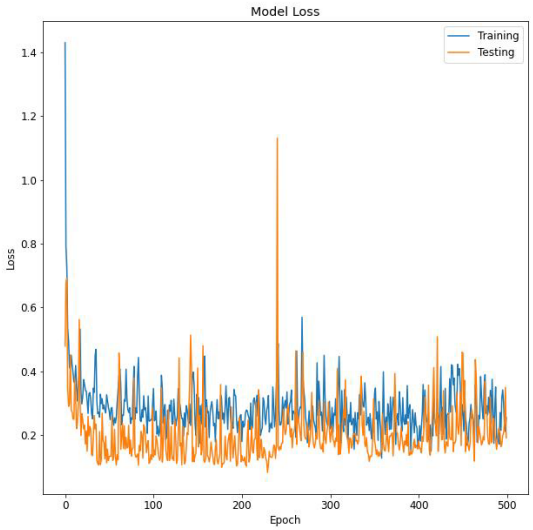


Figure 24b: Model Loss of VGG19

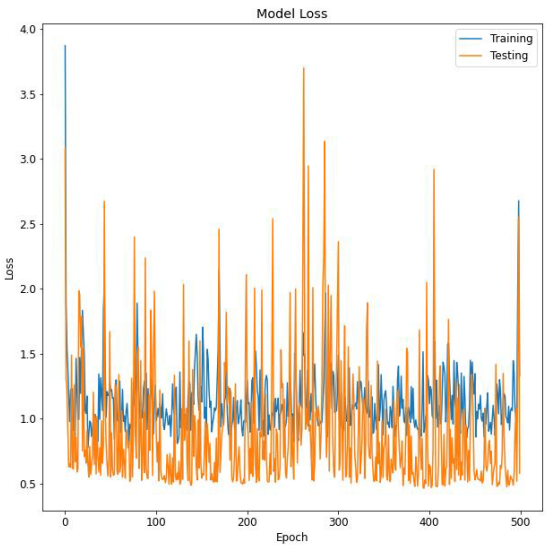


Figure 24c: Model Loss of Resnet50

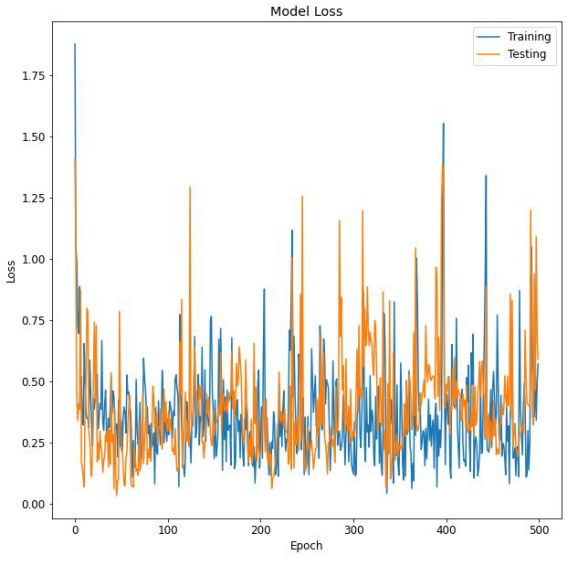


Figure 24d: Model Loss of Xception

Figure 24: Comparison of Loss curve

If we look at Figure 24b, we can see the Training loss is going down and the Test loss curve is also going down. In the end there is an unrepresentative split between train and test data. The curve is jumping up and down because we have limited data.

The test curve should follow the train curve which happens in all cases therefore showing us that the model is working properly. Although there are some spikes in the curve, it is not to be considered and if we find the mean of the curve, we can see that test curve is closer to the train curve in Inception and Xception model.

3.6.5 Classification Report

From the classification report, we can find out the accuracy of our model. Classification report shows accuracy, precision, recall, f1 score and support. We will use Accuracy of the model to evaluate and compare our algorithm using different models.

	precision	recall	f1-score	support
0	0.96	1.00	0.98	65
1	1.00	0.93	0.96	43
accuracy			0.97	108
macro avg	0.98	0.97	0.97	108
weighted avg	0.97	0.97	0.97	108

Figure 25a: Classification Report of Inception

	precision	recall	f1-score	support
0	0.94	0.97	0.95	65
1	0.95	0.91	0.93	43
accuracy			0.94	108
macro avg	0.95	0.94	0.94	108
weighted avg	0.94	0.94	0.94	108

Figure 25b: Classification Report of VGG19

	precision	recall	f1-score	support
0	0.86	0.65	0.74	65
1	0.61	0.84	0.71	43
accuracy			0.72	108
macro avg	0.73	0.74	0.72	108
weighted avg	0.76	0.72	0.72	108

Figure 25c: Classification Report of Resnet50

	precision	recall	f1-score	support
0	0.98	0.97	0.98	65
1	0.95	0.98	0.97	43
accuracy			0.97	108
macro avg	0.97	0.97	0.97	108
weighted avg	0.97	0.97	0.97	108

Figure 25d: Classification Report of Xception

Figure 25: Comparison of Classification Reports

In Figure 25a, we can see that the accuracy of the predictions using InceptionV3 is 0.97 or 97 percent. That means our model can make predictions which are true 97 out of 100 times. It has a precision of 97 percent, recall 97 percent and f1 score of 97 percent. We can visualize these parameters for the other models too in Figure 25. As a result, we can observe that Inception V3 and Xception has the better accuracy of the predictions.

3.7 Comparison between the Results of Malaria

From the given table, we can get to know about some of metrics of four models we have tested on our dataset. For our purpose, the model Inceptionv3 and Xception are giving us the best metrics, VGG19 becoming third and ResNet50 lagging at the fourth place. All four models with respect to different Evaluation metrices are compared below:

Table-3.1: Comparison of performance of different models

Model	Overall Accuracy	Weighted Average				Training Time per epoch seconds
		Precision	Sensitivity	F1 Score	Specificity	
ResNet50	0.72	0.76	0.84	0.72	0.65	5
InceptionV3	0.97	0.97	0.93	0.97	1	5
Vgg19	0.94	0.94	0.91	0.94	0.97	7
Xception	0.97	0.97	0.98	0.97	0.97	5

Sensitivity is the Recall of positive true values.

Specificity is the Recall of negative true values.

F1 score is the harmonic mean of Precision and recall

3.8 Flask Implementation

Initially we define the flask app. Inside the flask app we have:

1. Loading the model

We import our trained model path and give it in MODEL_PATH

MODEL_PATH will read the h5 file directly from our models' folder

Then we load the model

2. Defining the app route

The first app route is for our root folder. This app route will only show us our home page present in index.html. Index.html is present inside the template folder.

By default, render template will be index.html for our root path

3. Defining the predict path

A function upload is created. As soon as the person uploads the picture in the web app, firstly it is stored inside a path called as uploads in our folder.

As soon as it gets stored model predict function is hit which has our file path and our loaded model.

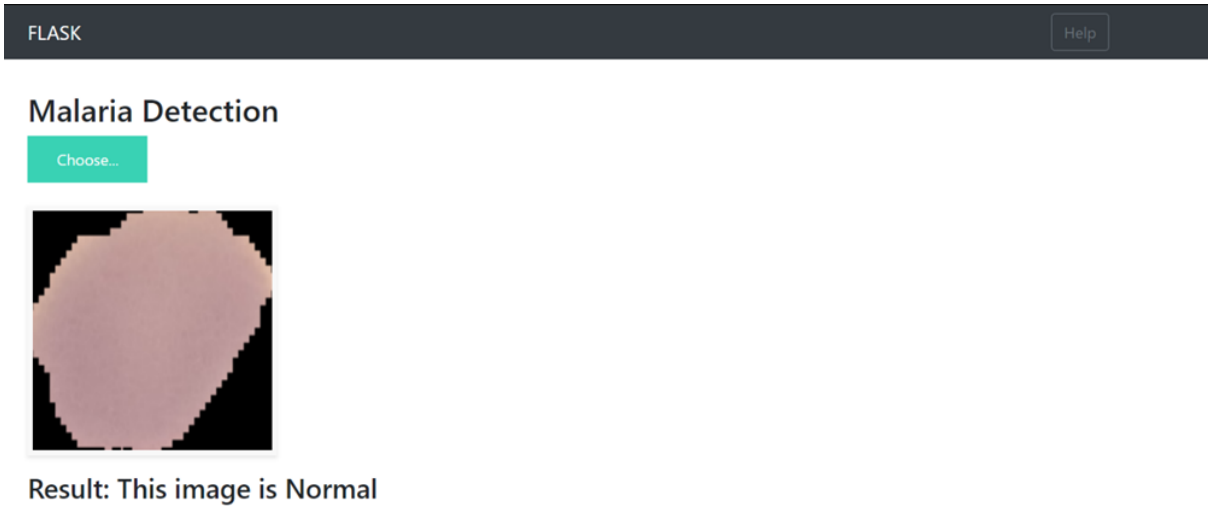
4. Defining model predict function

The image is loaded from the image path, converted into an array, dimensions expanded and pre-processed input.

Then we predict using model. Predict model: predict returns 0 or 1

if 0 then it is parasitized image and if 1then it is a normal image.

In the images below you can see a snapshot of our results.



In the end, we have worked on malaria detection of cell images using 4 different deep learning models. We trained our dataset using Inception V3, Vgg19, Resnet50 and Xception for 500 epochs with a batch size of 32 images in each epoch. Then we evaluated our model by plotting the ROC curves, Confusion Matrices, Classification Reports, Accuracy Graphs and the Loss graphs. We compared the results in section 3.7 and found that the model trained with Xception and Inception gave the best results with an overall accuracy of 97%. In section 3.8, we deployed the models in our local environment using a flask app where we can upload an image and see the prediction by our model.

Chapter 4

Case-2 Deep Learning Algorithm in COVID Detection

In this chapter we see how we worked on several deep learning models for predicting COVID-19 from a CT image and Chest Xray images. In the subsequent subchapters we will explain in detail how we compiled, trained our model, evaluated the model, compared the results and deployed it in a more user-friendly system using flask app.

4.1 Coronavirus and its background

One of the most searched words on google today is 'coronavirus' and 'coronavirus vaccine'. The coronavirus pandemic has had e deep effect on the whole world for the last one year starting officially from 31st December 2019. The virus was first detected in the city of Wuhan in China. Wuhan is one of the most important economic hubs of China. World Health organization was first informed about a widespread cluster of pneumonia case in Wuhan. This later spread to other parts of the city and by 5th January there were 59 people affected in Wuhan, but none of them died [44]. By the 15th of January the virus had started to be detected in the other countries like Japan, south Korea and Thailand [45]. The genome of the virus was officially introduced and named COVID-19 on the 11-12th of January 2020. Another important issue that came up with the up rise of the COVID-19 virus was that many claimed it to be a biological weapon made by powerful nations for war use and so on. Many claimed that it was laboratory made for weapon purpose. But later it was labelled as fake as many scientists around the world researched with its genomic structure and found nothing common of any known element. Another close case of Coronavirus was seen in 2003, which was labelled as the SARS-CoV-1. This virus is suspected to be originated from bats. The SARS-CoV-1 was seen to be contaminated among the animals too that time. But the Covid-19 is suspected to be originated from bats and transmitted to other animals and lastly to humans in Wuhan [46][47]. The Wuhan wet market is suspected to be the main place of contamination of the coronavirus in early stage of the pandemic. Alongside bats there is another animal, the nocturnal ant eater which is consumed illegally by people known to carry another type of coronavirus but is different from the SARS-CoV-2. Like most coronaviruses the SARS-2 can survive for hours in aerosols, steel surfaces and card boards. [48]. The virus spreads in the asymptomatic incubation phase and can stay up to 2 weeks of infection [49]. The major symptoms of coronavirus are fever, dry cough and tiredness. The alarming symptoms are breathing difficulties, chest pain, pressure and loss of speech or movement. As up to today the total cases of coronavirus is 88.1 M and

among these 49.1M recovered. There have been 1.9M deaths among these people. Testing is an important part of the coronavirus challenge for the whole world. The nations which have been ahead in testing more are the ones succeeding more in preventing the pandemic. Usually, two methods are used to detect the COVID-19 virus. Diagnostic Test and antibody test. Both of these tests are quite expensive and requires some time to give results. Mostly nasal swabs are used to take samples to detect the virus. Nowadays scientists are coming up with faster and budget efficient methods to detect coronavirus like the use of deep learning and X-ray image samples and CT scan images samples to detect corona virus.

4.2 Coronavirus Detection (COVID-19)

The whole world is suffering due to the COVID-19 statewide and most of our regular activities got hampered due to the month's long lockdown period. But COVID-19 is just more than a health crisis and it continued stressing every sector of the countries it touched, created devastating economic, social and political crises that left deep scars. It's still spreading slowly in different places around the world. The testing of the Covid-19 virus in human body is an urgent part of the crisis management. Basically, two branches are there to test the presence of the virus. One is to find the antibody another is to find the virus. If there is presence of antibody, it refers to a prior infection. And detection of virus means the patient is COVID positive. Also, there are basic tests that prove a patient positive or negative. For example, high temperature, low oxygen level, breathing difficulties and so on. The major methods for testing the virus are Reverse transcription polymerase chain reaction (RT-PCR). This method converts RNA to DNA and follows up with PCR to analyze the DNA, which later can be used to detect SARS-CoV-2. This process takes a 4-6 hours to detect. Also sometimes takes the whole day to give result. But the problem that arises for the RT-PCR is that it gives false results and the rate is pretty high which is almost 5%. Also, there are other disadvantages of this test like time taken to give result, false positive test and shortage of kits make it much inefficient.

On the other hand, X-Ray images and Ct scan images are a traditional form of detection of a number of diseases. Places where there are thousands of affected people with less testing kits, these methods of detecting COVID-19 with X-ray and CT scan images comes effective. Densely populated regions can use COVID-19 detection using X-Ray and Ct-scan images. Previously the images of effected and non-effected samples were not available, but now there are good number of samples available to test and train the data for COVID-19 detection using Deep learning. In this research we have used deep learning and python to train our program to

detect COVID and Non-COVID patients by imputing their Xray images. We used inception_v3 to train our program and the accuracy was 94 % for X-ray images.

4.3 Datasets

The datasets we have conduct to feed our algorithm is a public dataset from GitHub repository; [50,55]. During the first of January 2021, the dataset weights of 746 unique images of CT scans of patients.

The dataset contains meta information of patients details, patient other diseases information etc. COVID19-related papers like medRxiv, bioRxiv etc. are being used for gathering of images for the work. CTs containing COVID19 the picture titles in those papers provides the information of patient's abnormal behavior. All copyrights of the information associated to the authors and publishers of these papers. It is also mentioned that this dataset is constantly updated with new images of patients. More details of the dataset can be found here [51].

The appropriateness of this dataset has been performed by a senior radiologist in Tongji Hospital, Wuhan, China, who has been assigned in diagnosis and conducting of a bigger number of covid19 patients in the eve of the prevalence of this disease between January and April [52].

The classical method for medical and healthcare image classification basically follows a two-step of actions (hand-crafted feature extraction + recognition). Here, we use the latest boundary of DL skeleton that can directly forecast the COVID-19 disease from raw images without any further of feature education. Deep learning-based models (and more specifical (CNN)) have been shown to outperform the classical AI approaches in most of computer vision and health sector image analysis tasks in current years, and have been used in a greater range of problems from classification, segmentation, face recognition, to super-resolution and image incrementation. Since we have penetration to shortage number of images for our deep learning convolutional network, we require to perform some data augmentation to increase our training and testing images. We also used ImageNet within the final step of the pretrained model which will make the algorithm less expensive. There were some concerts regarding the operation of this dataset. First of all, when the central images of the Ct scans are kept into paper, the image loses quality and gets degraded. The degraded images will bring up the diagnosis arguments inaccurate. The dataset was later tested and verified by different radiologists and the issues raised in these concerns do not significantly affect the accuracy formation of diagnosis low quality. Any experienced radiologist is capable of making less error diagnosis from low quality

Ct images. Similarly, the quality gap between Ct images collected from the papers and the actual CT images will not be that effective to accuracy. It is considering that there is variation in the accomplishment of images in this dataset. There are some low-resolution images in Covid-19 class (below 400 × 400), and some high-resolution ones (more than 1900 × 1400). This is a positive point for the models that can achieve a reasonable high accuracy on this dataset. The dataset was last penetrated in January 2021. Also, this dataset which is not intended to be for competitive programming in Kaggle rather to be used for prediction and detection of COVID19. Our CT scan dataset comprised of 349 COVID-19 affected images and 407 non-COVID-19 images and was collected from Kaggle repository. And for X-Ray images, we had 930 COVID-19 affected samples and 880 non-COVID-19 samples.

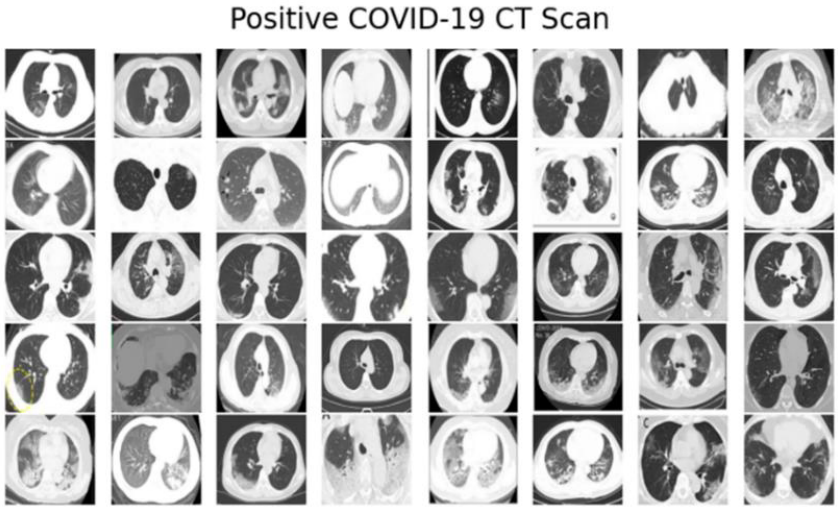


Figure 26 (a): Positive COVID19 CT Scan Sample

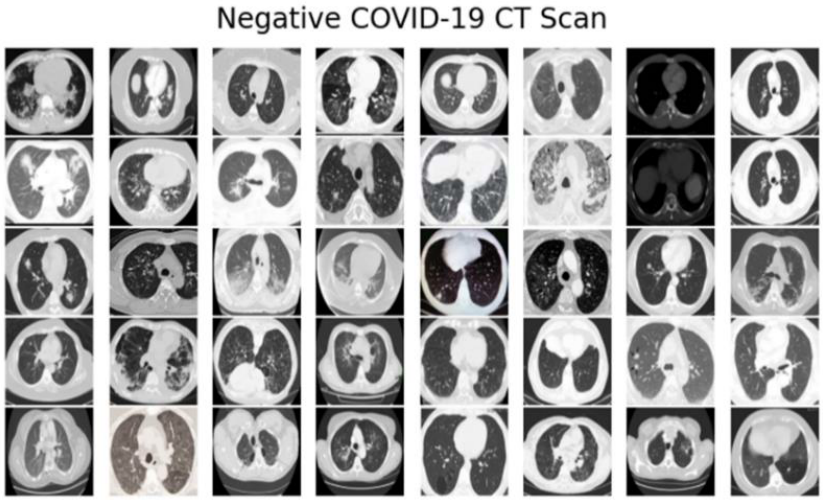


Figure 26(b): Negative COVID19 CT Scan Sample

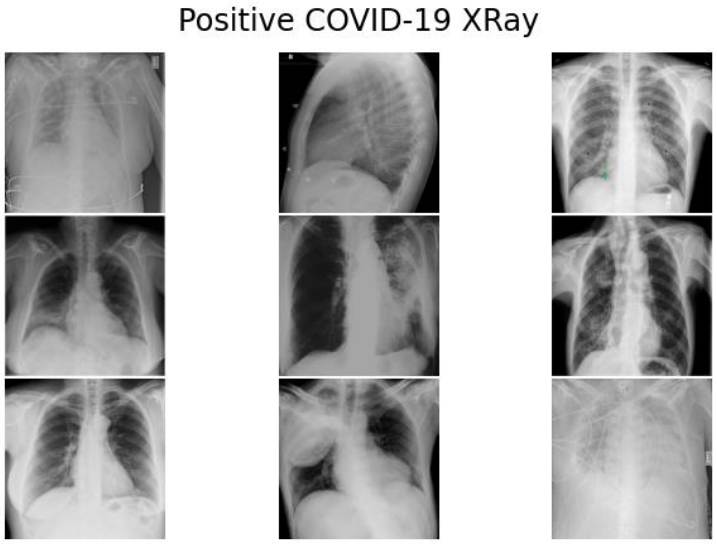


Figure 26(c): Positive COVID19 Chest Xray Sample

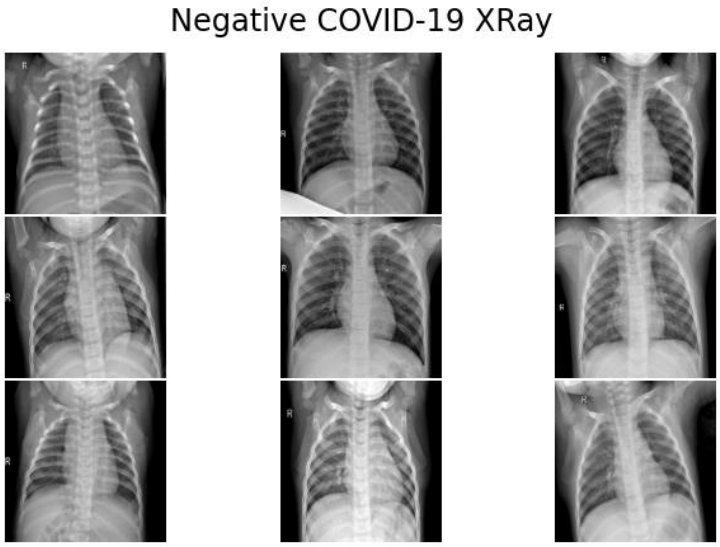


Figure 26(d): Negative COVID19 Chest Xray Sample

Fig. 26(a) and Fig.26(b) show some samples from the dataset containing positive and negative Covid19 Ct Scan images while Fig. 26(c) and Fig.26(d) show some samples of Xray images.

4.4 General Procedure

We used codes similar to that we used in Malaria detection. The following sub-sections under 4.4 describe the steps we used in running our models using 2 types of COVID19 images (CT scan and Chest Xray images). We used the similar process using InceptionV3, ResNet50, VGG19 and Xception

4.4.1 Data Preprocessing

- Data Splitting

The dataset we have used is split into two sets, namely Train and test. However, we should keep in mind that medical images are comparatively more diverse and subjective and so similar cases are observed in our dataset.

It is expected that the deployment of the algorithm may not initially match with other images of real-world scenarios and thus we can receive images drastically different from those used in

the training. Our dataset is split in the ratio of 80:20 for training and testing sets. This was randomly sampled from the dataset into the subsets.

- Data Augmentation

Data Augmentation is the enhancing process of data points to increase data points by adding slightly modified copies of already existing data. It has been used in most of the datasets to improve classification performance [53][54].

Data augmentation is necessary when training data becomes complex because of shortage of samples. In our case, as the dataset contains almost 700 images, we can use augmentation techniques to increase the data points.

We have used data augmentation to improve the performance of our dataset. As our dataset contains almost equal numbers of images for the different classes, it is a balanced dataset.

In our case, Data Augmentation includes horizontal and vertical flips and feature wise standardization. The techniques we have used and the values of change in data augmentation are given: [53,54, Tab. 4.1]

Table 4.1: Augmentation types and parameters

Augmentation Type	**Parameters**
rotation range	20(rotates by 20degree)
Width_shift_range	0.2(take the percentage of total width as range)
height_shift_range	0.2(take the percentage of total height as range)
Horizontal flip	true (flips image)

The newly created images can be used to pre-train the given neural network in order to improve the training process efficiency. Besides, Augmentation can overcome the problem of overfitting and enhance the accuracy of the proposed model. After applying augmentation, we have used the ReLu activation function to speed up the training process.

4.4.2 Building the model

Similar to the malaria dataset, our COVID-19 images were of different sizes. That is why we resized the input images into 224 by 224 pixels which is ideal for our models for training. Then we added the tensor shape as 224 by 224 by 3 px where 3 is the number of channels.

We used a Flatten layer to flatten our input features and a Dropout layer to overcome overfitting. We used a Dense output layer using a SoftMax Activation Function. This is so that we have 2 classes in the output as either parasitized or uninfected.

We made the trainable attribute of the previous layer as False since first part of the model is already trained. In the end we used Adam optimizer to compile the model using categorical crossentropy as the loss function.

The code for adding custom layers, for example, to the Xception model is shown below. The code for the rest of the models is the same. Only the Xception in the first line needs to be changed into the name of the desired model.

```
xception = Xception(weights="imagenet", include_top=False,
    input_tensor=Input(shape=(224, 224, 3)))

outputs = xception.output
outputs = Flatten(name="flatten")(outputs)
outputs = Dropout(0.5)(outputs)
outputs = Dense(2, activation="softmax")(outputs)

model = Model(inputs=xception.input, outputs=outputs)

for layer in xception.layers:
    layer.trainable = False

model.compile(
        loss='categorical_crossentropy',
        optimizer='adam',
        metrics=['accuracy']
)
```

We can see the last layer has 2 outputs as we used SoftMax activation function.

```
[ ]
activation_88 (Activation)      (None, 5, 5, 384)    0           batch_normalization_88[0][0]
__________________________________________________________________________________________________
activation_91 (Activation)      (None, 5, 5, 384)    0           batch_normalization_91[0][0]
__________________________________________________________________________________________________
activation_92 (Activation)      (None, 5, 5, 384)    0           batch_normalization_92[0][0]
__________________________________________________________________________________________________
batch_normalization_93 (BatchNo (None, 5, 5, 192)    576         conv2d_93[0][0]
__________________________________________________________________________________________________
activation_85 (Activation)      (None, 5, 5, 320)    0           batch_normalization_85[0][0]
__________________________________________________________________________________________________
mixed9_1 (Concatenate)          (None, 5, 5, 768)    0           activation_87[0][0]
                                                                 activation_88[0][0]
__________________________________________________________________________________________________
concatenate_1 (Concatenate)     (None, 5, 5, 768)    0           activation_91[0][0]
                                                                 activation_92[0][0]
__________________________________________________________________________________________________
activation_93 (Activation)      (None, 5, 5, 192)    0           batch_normalization_93[0][0]
__________________________________________________________________________________________________
mixed10 (Concatenate)           (None, 5, 5, 2048)   0           activation_85[0][0]
                                                                 mixed9_1[0][0]
                                                                 concatenate_1[0][0]
                                                                 activation_93[0][0]
__________________________________________________________________________________________________
flatten (Flatten)               (None, 51200)        0           mixed10[0][0]
__________________________________________________________________________________________________
dropout (Dropout)               (None, 51200)        0           flatten[0][0]
__________________________________________________________________________________________________
dense (Dense)                   (None, 2)            102402      dropout[0][0]
==================================================================================================
Total params: 21,905,186
Trainable params: 102,402
Non-trainable params: 21,802,784
__________________________________________________________________________________________________
```

4.5 Results and Discussions using CT scan images

We again ran the trained models on the test set and plotted the confusion matrices, ROC curves, classification reports, Accuracy and Loss curves.

There are two parts of the COVID19. Firstly, let us look at the results using the CT scan images of COVID-19 and non-COVID-19 infected people.

4.5.1 Confusion matrix

A confusion matrix describes the performance of a classification model as we have learnt from chapter 2 and 3. In this chapter the True Positives are the patients who has COVID19 and detected correctly by the algorithm. The 'True Negatives' are the patients who did not have COVID19 and the algorithm correctly identified them for not having COVID19. The 'False negatives' are when a patient has COVID19, but the algorithm says they don’t. And finally, the False positives are patients that do not have COVID19 but the algorithm says they do.

We took 152 samples to test. Among them, our trained model could detect 67 of the affected patients correctly. And 59 unaffected patients were also detected correctly by our model. And our model misclassified 3 patients having malaria as unaffected. Only 23 patients who were uninfected were identified as affected by our algorithm. For normalized values, our trained model could detect 54% of the affected patients correctly. And 71% of the unaffected patients were also detected correctly by our model. And our model misclassified 4.3% of the patients having malaria as unaffected. Only 28% of the patients who were uninfected were identified as affected by our algorithm. This is shown in figure 27a.

In the same way if we interpret the other matrices, we can see that InceptionV3 has the best confusion matrix as it has the least false positives.

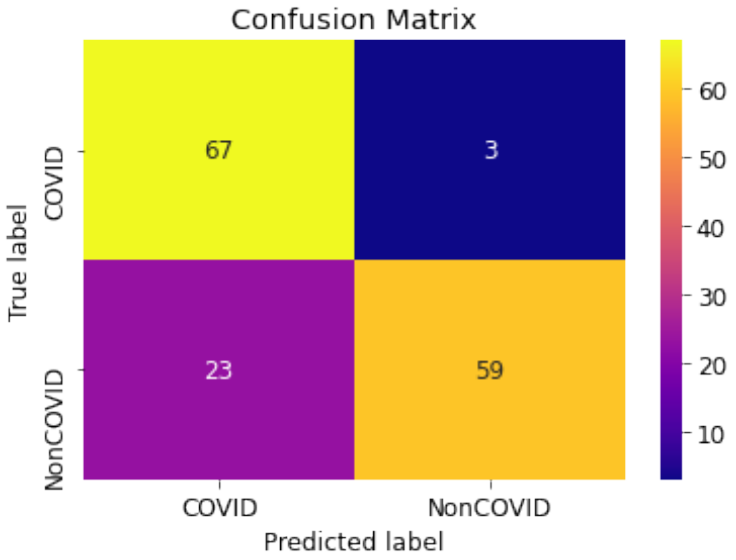


Figure 27a: Confusion Matrix of InceptionV3

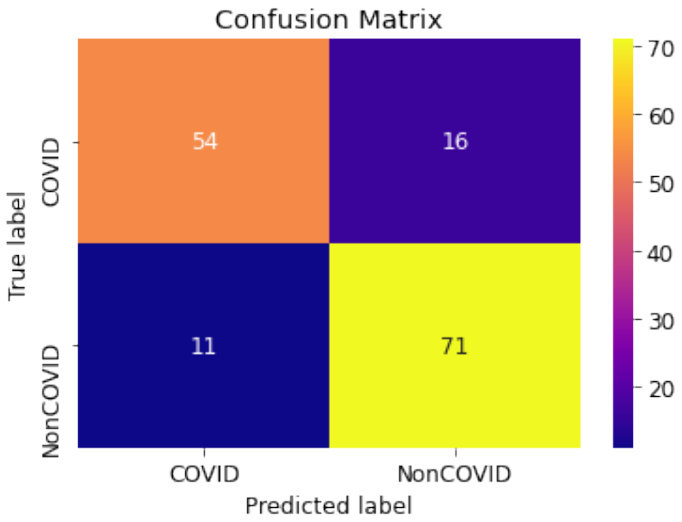


Figure 27b: Confusion Matrix of Vgg19

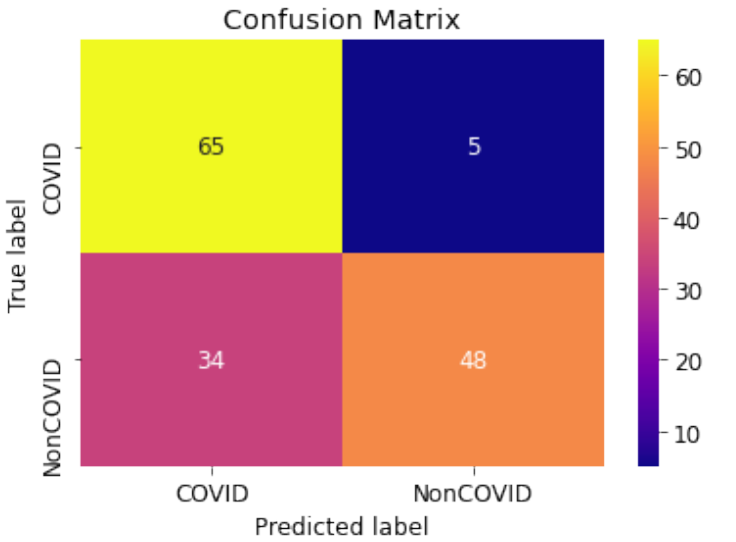


Figure 27c: Confusion Matrix of Resnet50

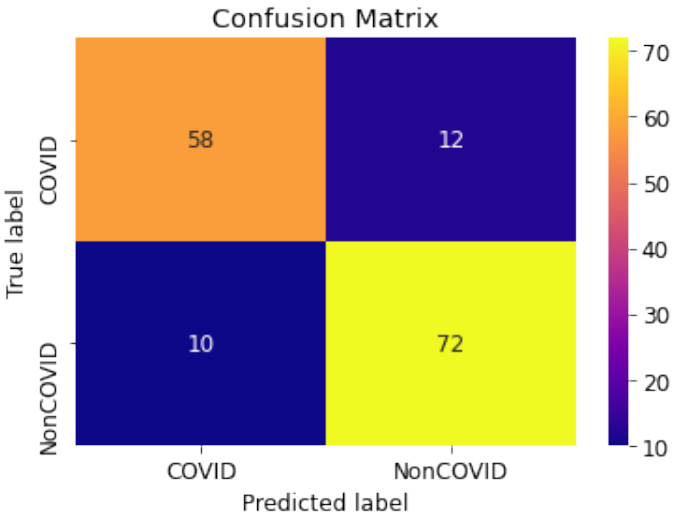


Figure 27d: Confusion Matrix of Xception

Figure 27: Comparison of Confusion Matrix without normalization

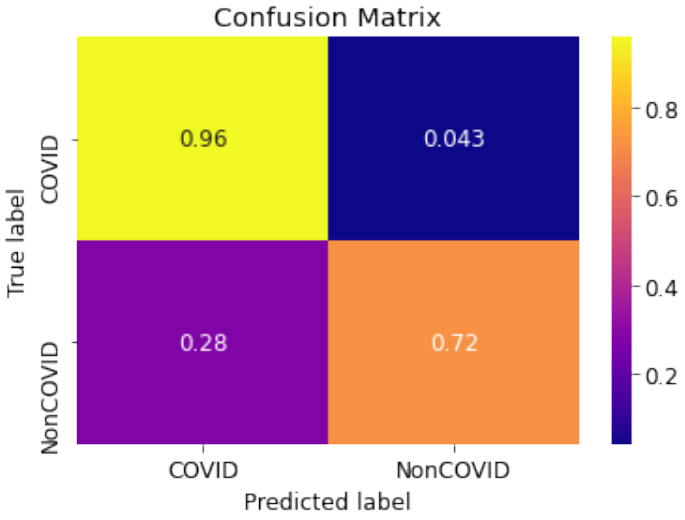


Figure 28a: Confusion Matrix of InceptionV3

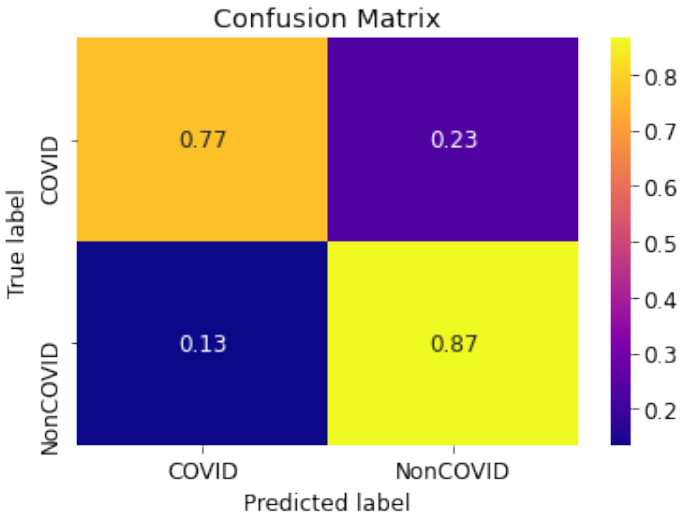


Figure 28b: Confusion Matrix of Vgg19

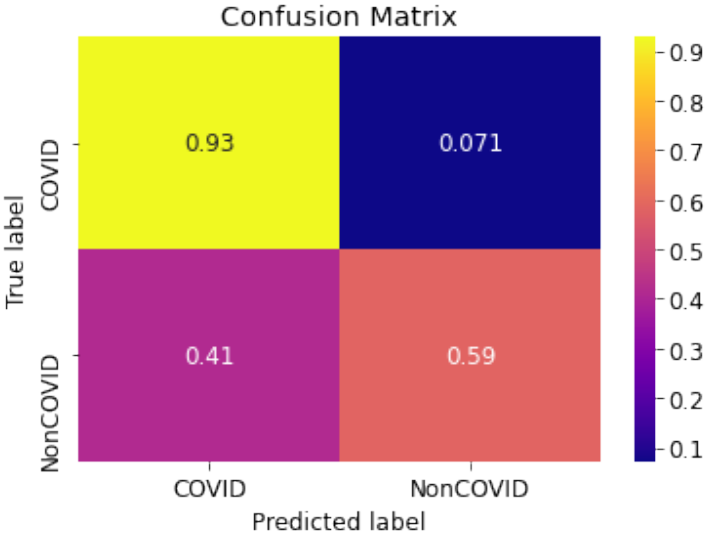


Figure 28c: Confusion Matrix of Resnet50

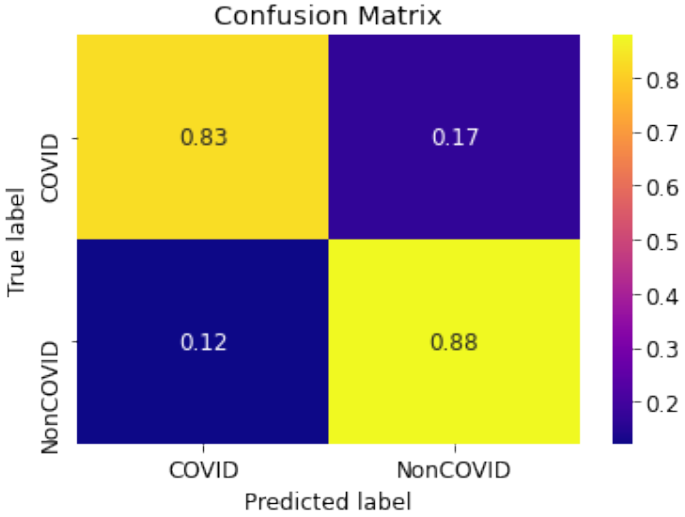


Figure 28d: Confusion Matrix of Xception

Figure 28: Comparison of Confusion Matrix with normalization

4.5.2 ROC curve

We already know from chapters 2 and 3 that a ROC curve or receiver operating characteristic curve is a graph that shows the efficiency of a classification model at all classification thresholds.

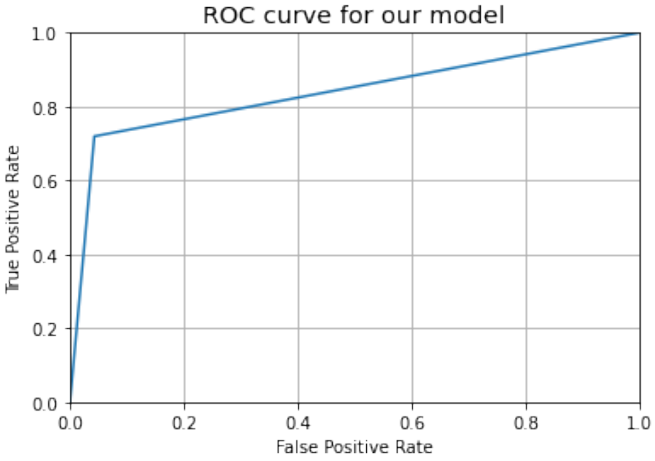


Figure 29a: ROC curve of InceptionV3

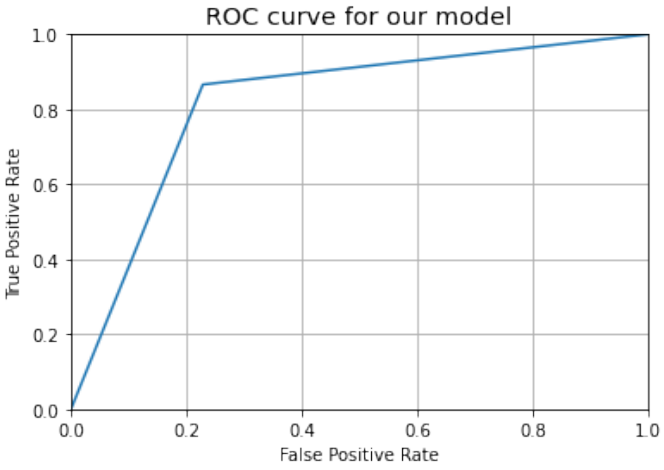


Figure 29b: ROC curve of Vgg19

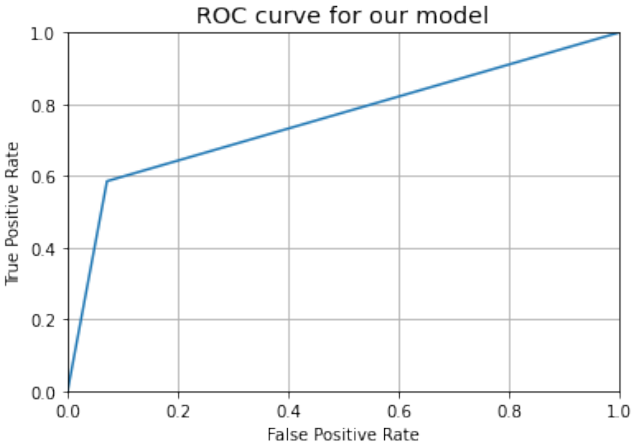


Figure 29c: ROC curve of Resnet50

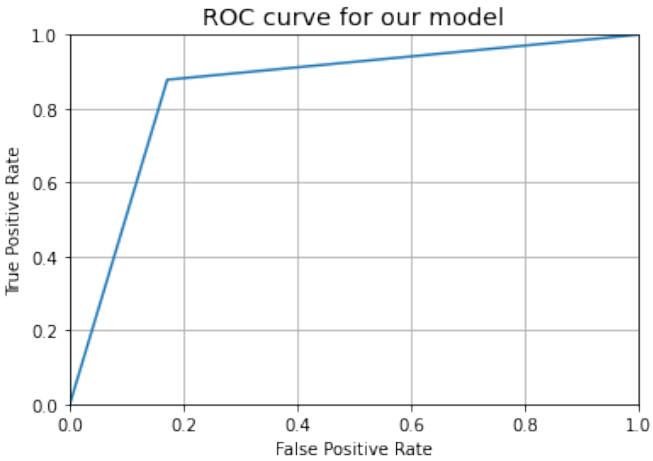


Figure 29d: ROC curve of Xception

Figure 29: Comparison of ROC Curves

Figure 29 shows the ROC curve. Along the X axis of ROC curve, False Positive Rate or FPR is represented and along Y axis True Positive Rate or TPR is represented.

In the above ROC curve of Xception model, that is Figure 29d we can see that the curve is very far away from the 45-degree diagonal of the ROC space which indicates that the curve is very accurate. The AUC indicates also confirms our model is very accurate.

Xception model has the highest Area under curve as the True positive rate starts at almost 1 for false positive rate of 0.01. We can compare that to VGG19 and Inceptionv3 where the AUC is quite similar. ResNet50 model has low training time and also has the lowest accuracy in our case as the AUC is the least among the 4 models.

4.5.3 Model Accuracy

Accuracy is the number of correct predictions. We used both test and train data and calculating the accuracy after each epoch to form the accuracy curve. This is shown in Figure 30. The blue line describes the training accuracy and the yellow line describes the test accuracy.

In the model accuracy curves. The test curve follows the train curve which happens in all cases therefore showing us that the model is working properly. The fluctuation in the line is due to the limited amount of data we had. However, we can see that test curve is closer to the train curve in Xception and VGG19 model.

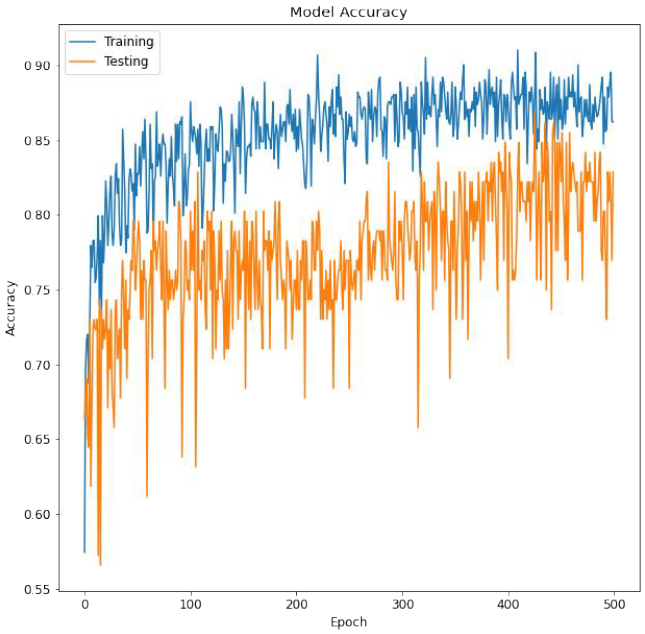


Figure 30a: Model Accuracy of InceptionV3

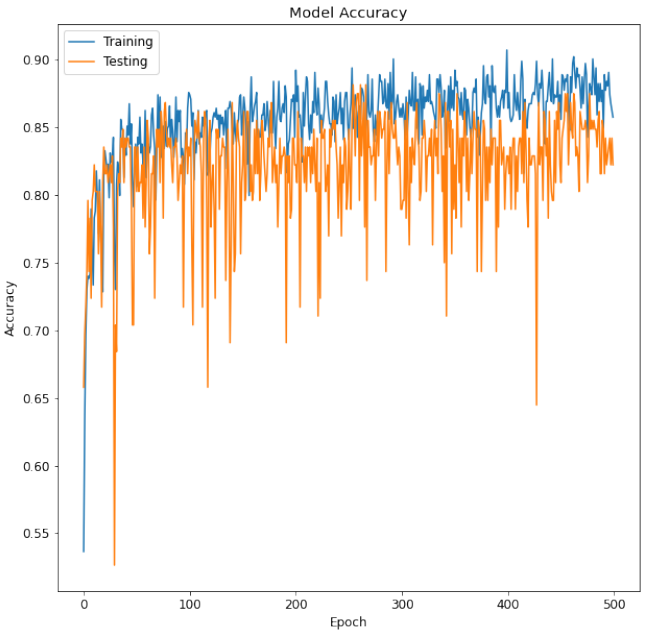


Figure 30b: Model Accuracy of VGG19

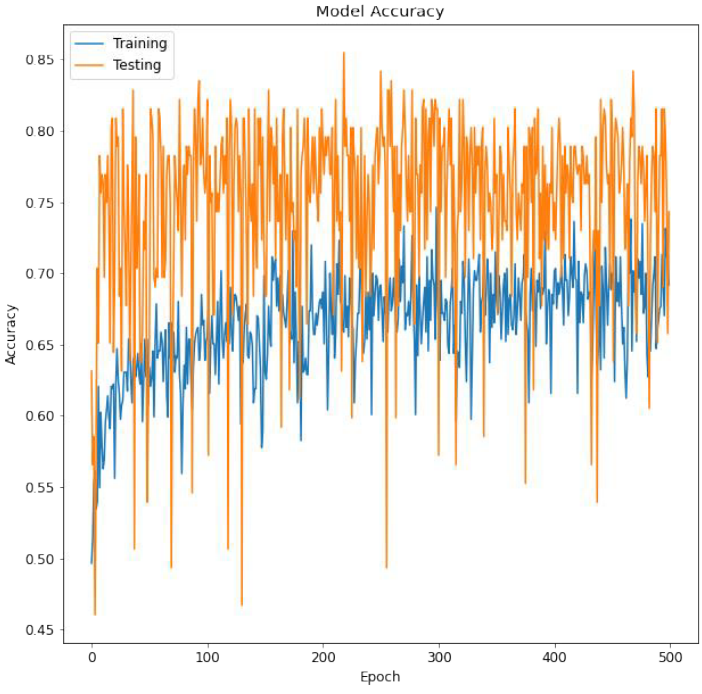


Figure 30c: Model Accuracy of Resnet50

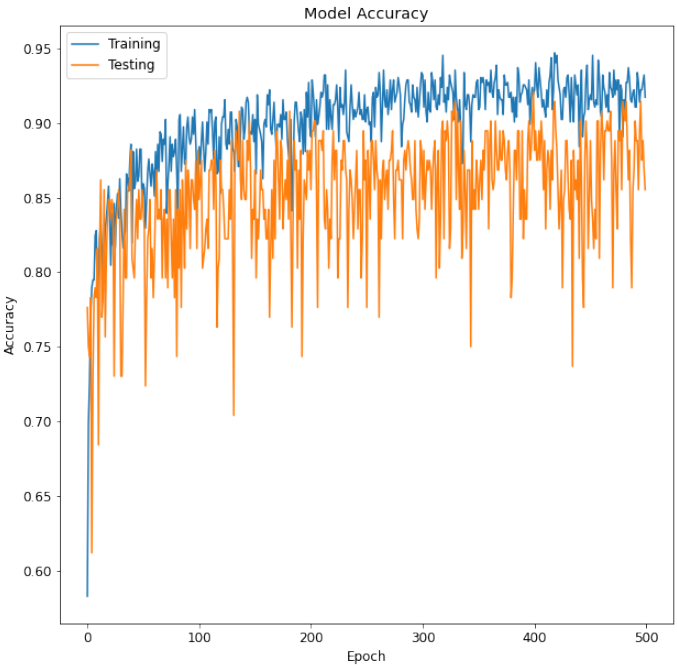


Figure 30d: Model Accuracy of Xception

Figure 30: Comparison of Accuracy curve

4.5.4 Model Loss Comparison

In the model loss curves, we can find more about how the testing and training process is taking place.

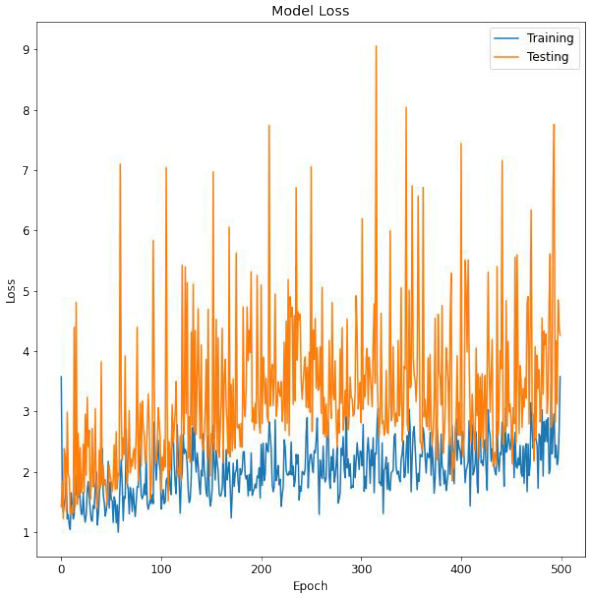


Figure 31a: Model Loss of Inception

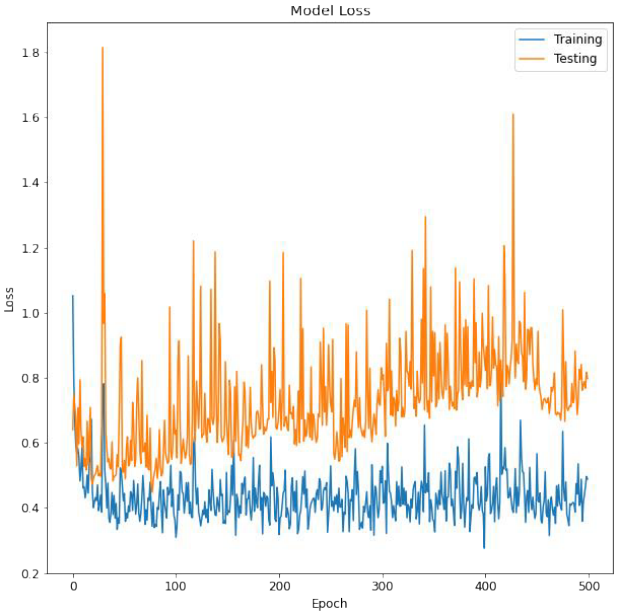


Figure 31b: Model Loss of VGG19

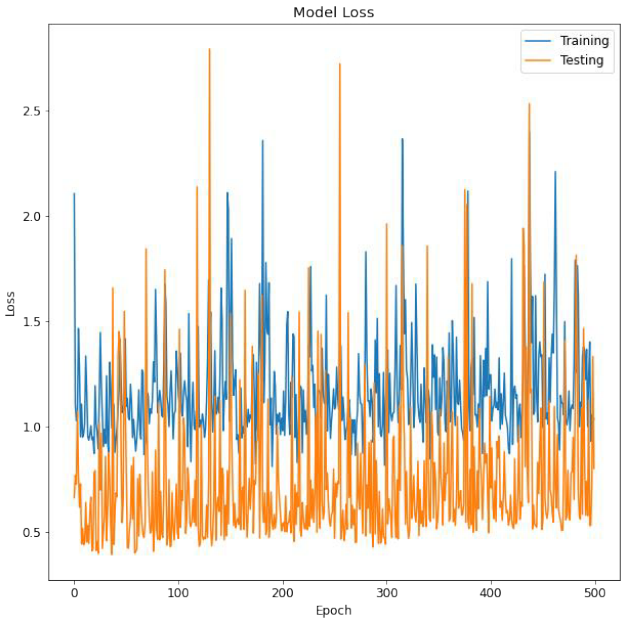


Figure 31c: Model Loss of Resnet50

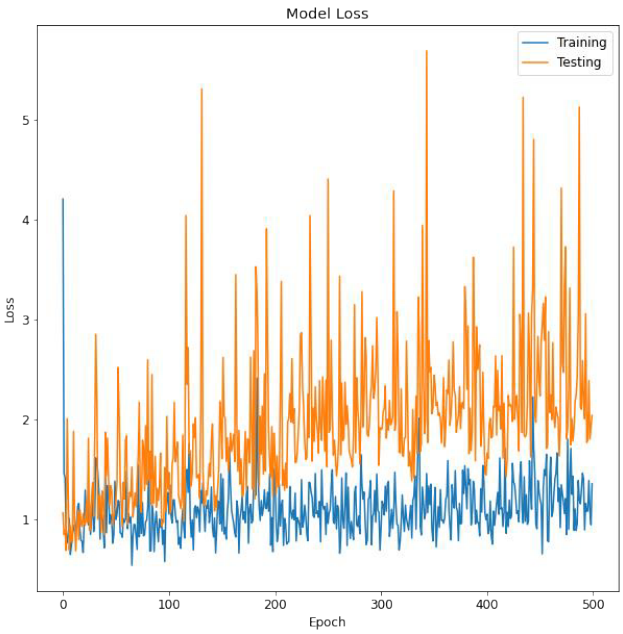


Figure 31d: Model Loss of Xception

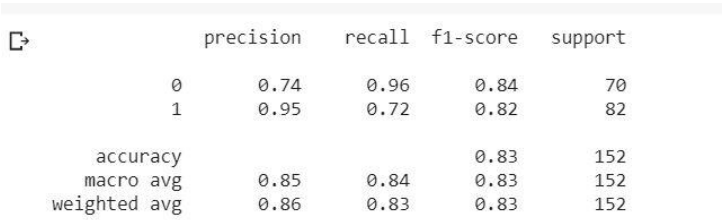
Figure 31: Comparison of Loss Curves

In the Figure 31, we can see both the Training loss and the Test loss curve is going down. In the end there is an unrepresentative split between train and test data. The curve is jumping up and down because we have limited data.

The test curve should follow the train curve which happens in all cases therefore showing us that the model is working properly. Although there are some spikes in the curve, it is not to be considered and if we find the mean of the curve, we can see that test curve is closer to the train curve in Inception and Xception model.

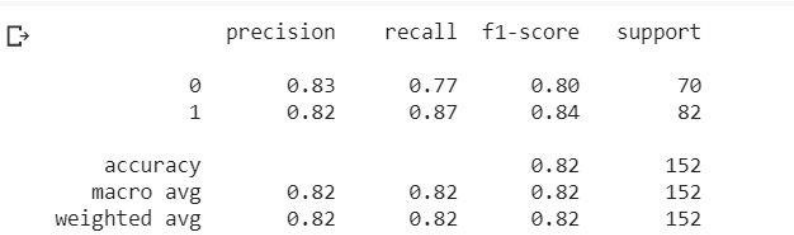
4.5.5 Classification Report

From the classification report, we can find accuracy, precision, recall, f1 score and support of our model.



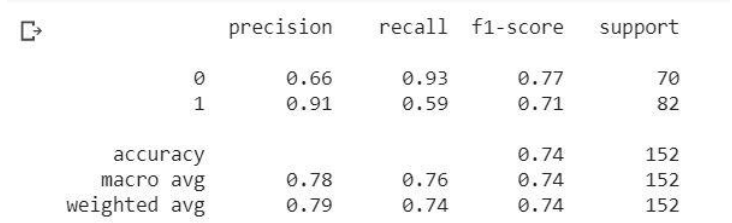
	precision	recall	f1-score	support
0	0.74	0.96	0.84	70
1	0.95	0.72	0.82	82
accuracy			0.83	152
macro avg	0.85	0.84	0.83	152
weighted avg	0.86	0.83	0.83	152

Figure 32a: Classification Report of Inception



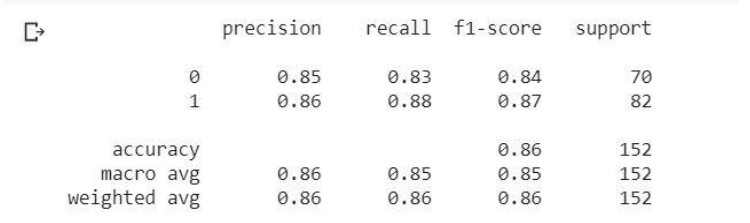
	precision	recall	f1-score	support
0	0.83	0.77	0.80	70
1	0.82	0.87	0.84	82
accuracy			0.82	152
macro avg	0.82	0.82	0.82	152
weighted avg	0.82	0.82	0.82	152

Figure 32b: Classification Report of VGG19



	precision	recall	f1-score	support
0	0.66	0.93	0.77	70
1	0.91	0.59	0.71	82
accuracy			0.74	152
macro avg	0.78	0.76	0.74	152
weighted avg	0.79	0.74	0.74	152

Figure 32c: Classification Report of Resnet50



	precision	recall	f1-score	support
0	0.85	0.83	0.84	70
1	0.86	0.88	0.87	82
accuracy			0.86	152
macro avg	0.86	0.85	0.85	152
weighted avg	0.86	0.86	0.86	152

Figure 32d: Classification Report of Xception

Figure 32: Comparison of Classification Reports

In Figure 32c, we can see that the accuracy of the predictions using ResNet50 is 74 percent. It has a precision of 79 percent, recall 74 percent and f1 score of 74 percent. We can visualize these parameters for the other models too in Figure 25. As a result, we can observe that Xception has the better accuracy of the predictions. This is confirmed by the confusion matrices and the ROC curves we have seen.

4.5.6 Comparison between the Results using CT scan images

From the given table, we can get to know about some of metrics of four models we have tested on our dataset. The model Xception gives us the best metrics, Next comes Inception V3 and VGG19. And ResNet50 lagging at the fourth place. All four models with respect to different Evaluation metrices are compared below:

Table-4.2: Comparison of performance of different models

Model	Overall Accuracy	Weighted Average				Training Time per epoch (seconds)
		Precision	Sensitivity	F1 Score	Specificity	
ResNet50	0.74	0.79	0.59	0.74	0.93	5
InceptionV3	0.83	0.86	0.72	0.83	0.96	5
Vgg19	0.82	0.82	0.87	0.82	0.77	7
Xception	0.86	0.86	0.88	0.86	0.83	5

Sensitivity is the Recall of positive true values.

Specificity is the Recall of negative true values.

F1 score is the harmonic mean of Precision and recall

4.6 Results and Discussions using Chest Xray images

We have seen in section 4.5 the results using CT scan images. In this section we will see the results using the Chest Xray images.

4.6.1 Confusion matrix

A confusion matrix describes the performance of a classification model as we have learnt from chapter 2 and 3. Moreover, from section 4.5.1 we already learnt what he True Positives, True Negatives, False Positives and False Negatives are.

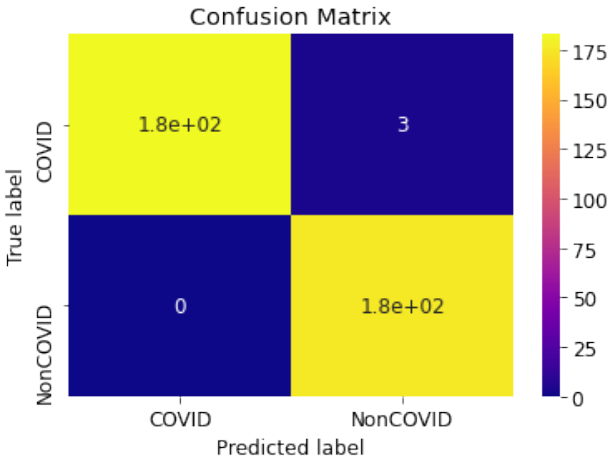


Figure 33a: Confusion Matrix of InceptionV3

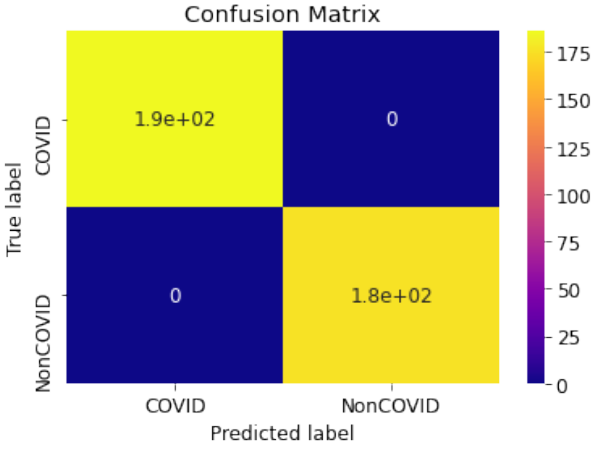


Figure 33b: Confusion Matrix of Vgg19

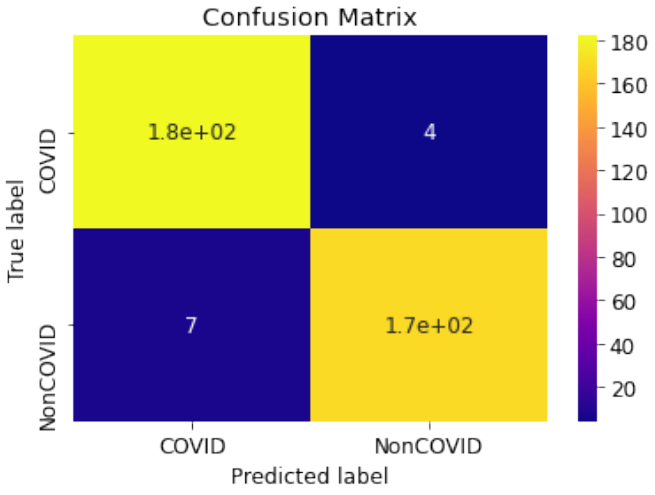


Figure 33c: Confusion Matrix of Resnet50

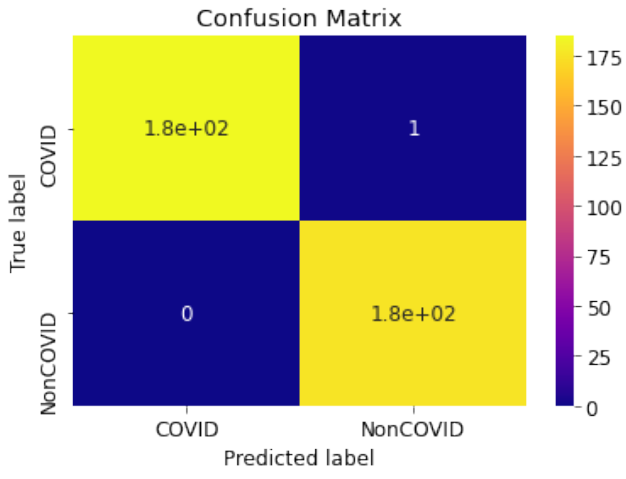


Figure 33d: Confusion Matrix of Xception

Figure 33: Comparison of Confusion Matrix without normalization

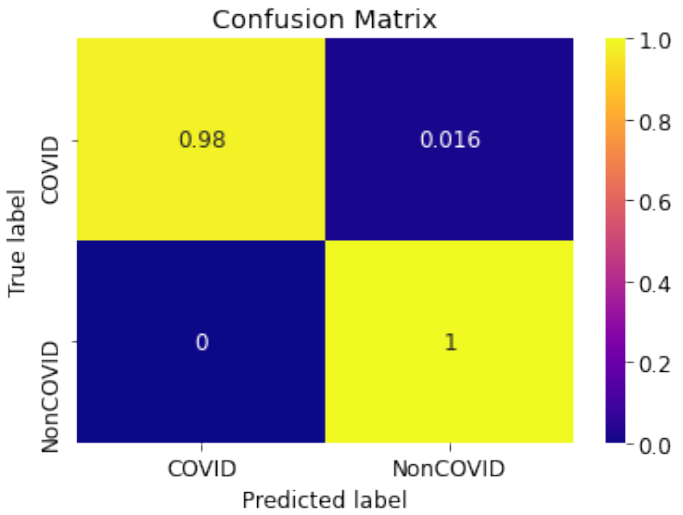


Figure 34a: Confusion Matrix of InceptionV3

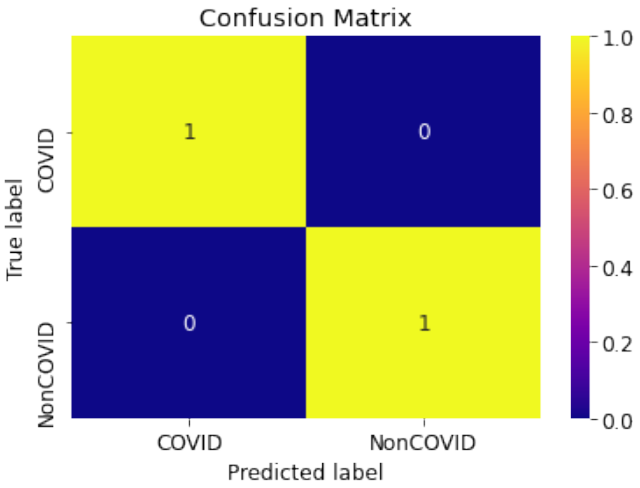


Figure 34b: Confusion Matrix of Vgg19

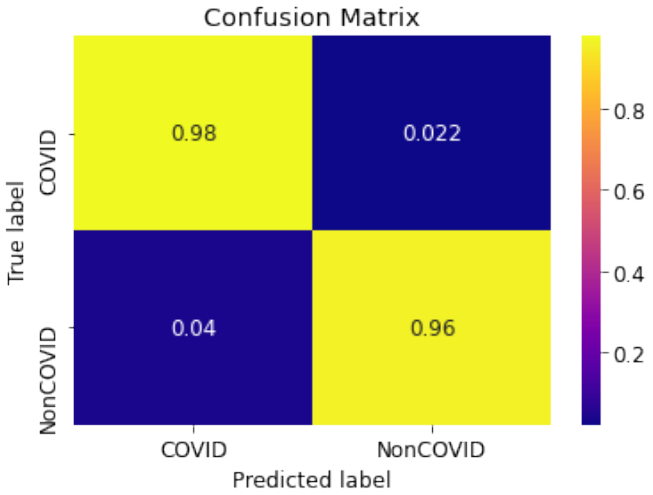


Figure 34c: Confusion Matrix of Resnet50

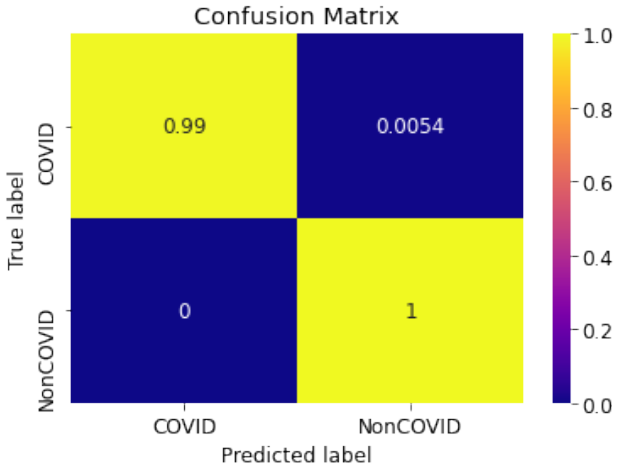


Figure 34d: Confusion Matrix of Xception

Figure 34: Comparison of Confusion Matrix with normalization

Figure 33 shows the confusion matrices of the different models without normalization and Figure 34 shows the confusion matrices with normalization.

If we interpret the other matrices in the same way as we did in section 4.5.1, we can see that all 4 models give really good results. VGG19 has the best confusion matrix as it has 0 False Positives and False Negatives and 100% True Positive and True Negatives.

4.6.2 ROC curve

From chapter 2, we already know that an ROC curve displays the performance of a classification model at all classification thresholds. Figure 35 shows the ROC curves of our models using Xray images dataset.

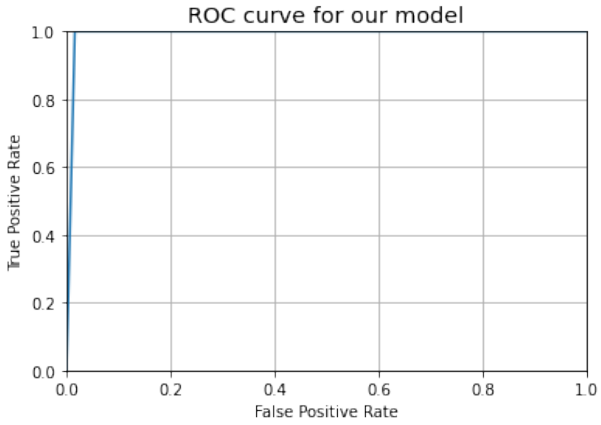


Figure 35a: ROC curve of InceptionV3

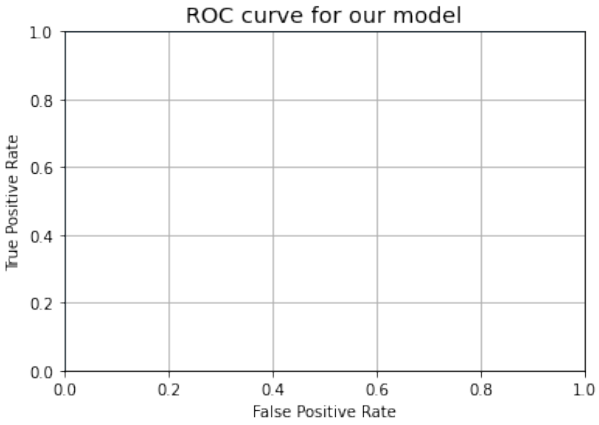


Figure 35b: ROC curve of Vgg19

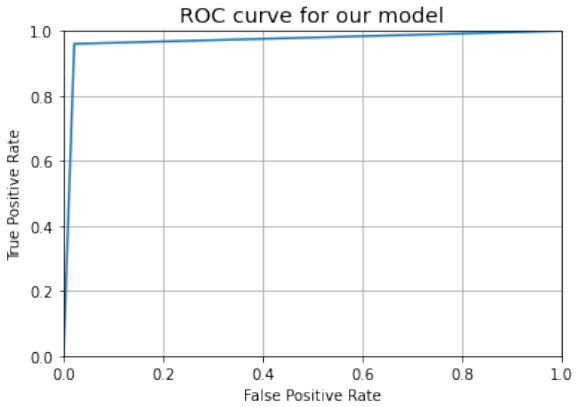


Figure 35c: ROC curve of Resnet50

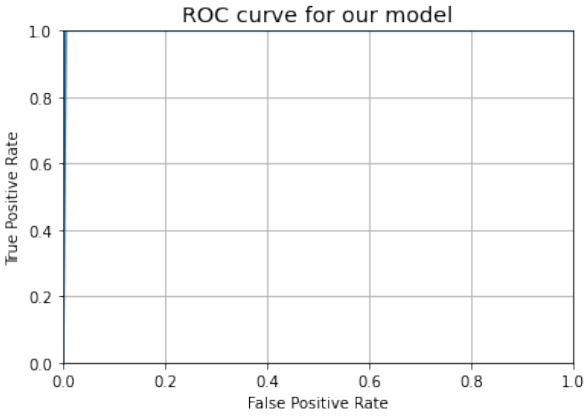


Figure 35d: ROC curve of Xception

Figure 35: Comparison of ROC Curves

Classifiers that give curves closer to the top-left corner indicate a better performance. The Area Under Curve or AUC is used as predictive accuracy measure for ROC curves. We can see that the Area under the Curve is quite high in all 4 models. This indicates that all our 4 models are giving accurate results. The VGG19 model has the highest Area under curve as the True positive rate starts at almost 1 for false positive rate of 0.01. From the ROC curves in Figure 35, we can see that ResNet50 model has low training time and also has the lowest accuracy in our case as the AUC is the least among the 4 models.

4.6.3 Model Accuracy

Accuracy is one metric for evaluating classification models. Informally, accuracy is the fraction of predictions our model got right. We plotted a curve for the Accuracy vs Number of Epochs. This can be seen in Figure 36. The blue line describes the training accuracy and the yellow line describes the test accuracy.

In the model accuracy curves. The test curve follows the train curve which happens in all cases therefore showing us that the model is working properly. Although there are some spikes in the curve, it is not to be considered and if we find the mean of the curve. The curves are closes together in InceptionV3.

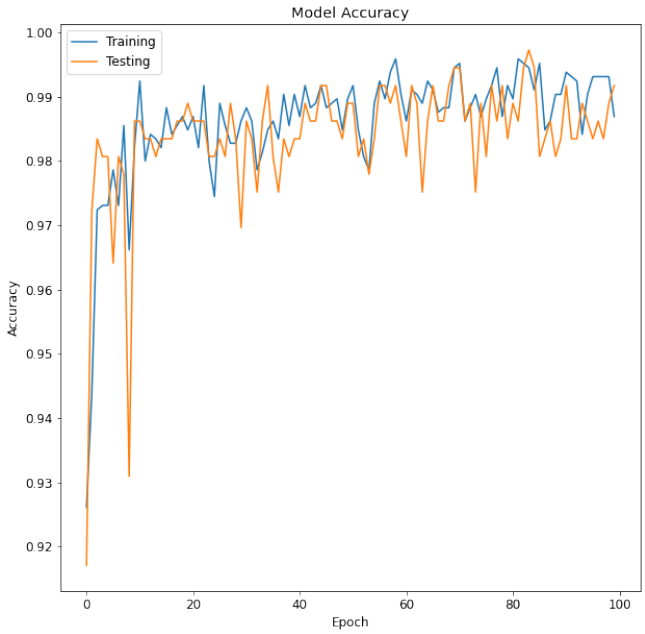


Figure 36a: Model Accuracy of InceptionV3

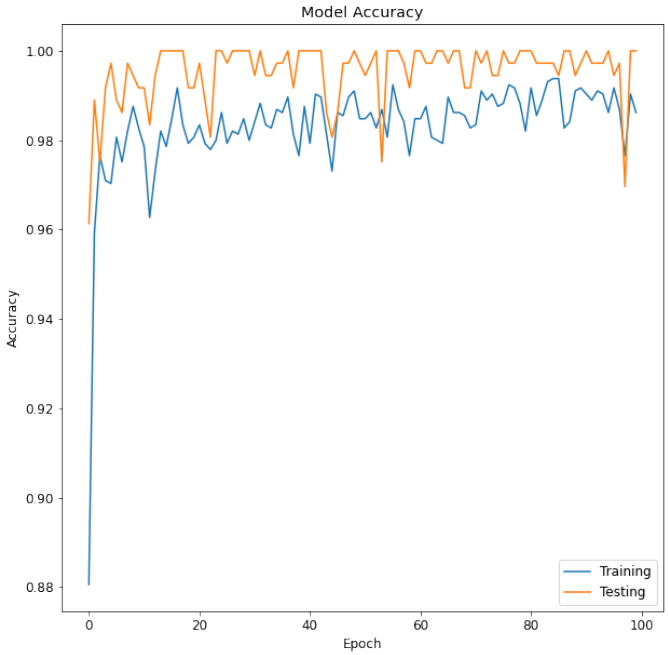


Figure 36b: Model Accuracy of VGG19

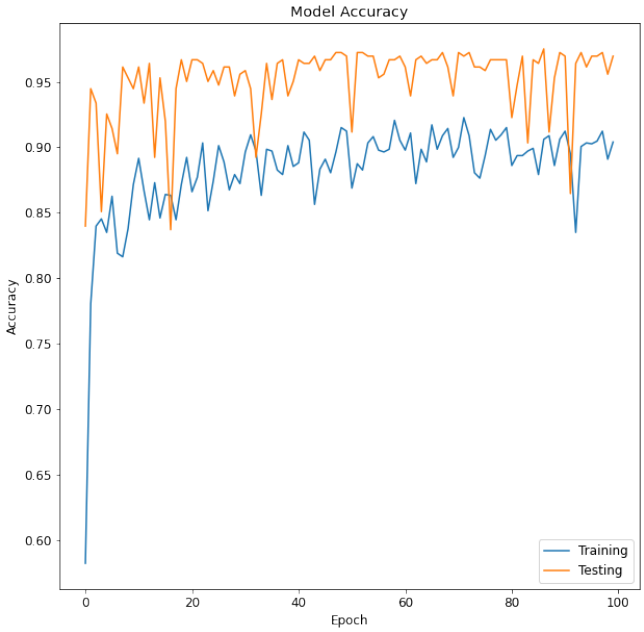


Figure 36c: Model Accuracy of Resnet50

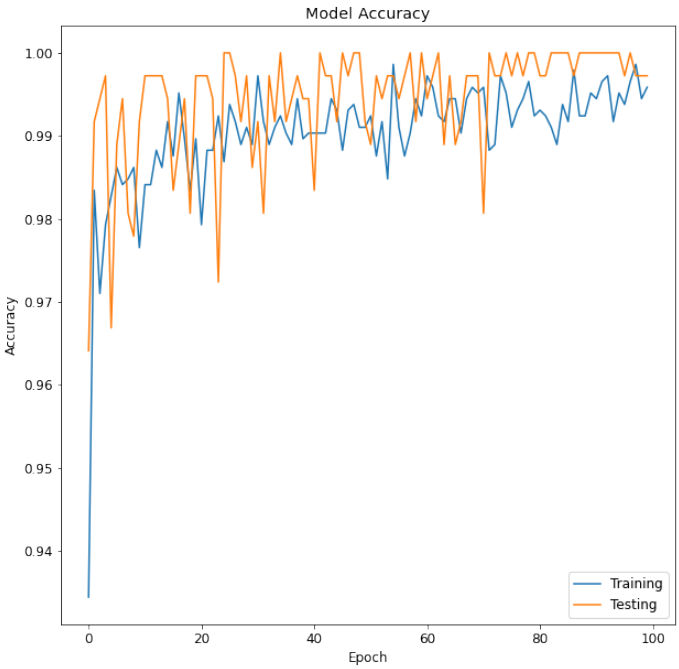


Figure 36d: Model Accuracy of Xception

Figure 36: Comparison of Accuracy curve

4.6.4 Model Loss Comparison

The most popular example of a learning curve is *loss over time*. Loss (or cost) measures our model error, or "how bad our model is doing". A loss curve is plotted as Loss vs Epochs. From Figure 37, using the model loss curves, we can find more about how the testing and training process is taking place.

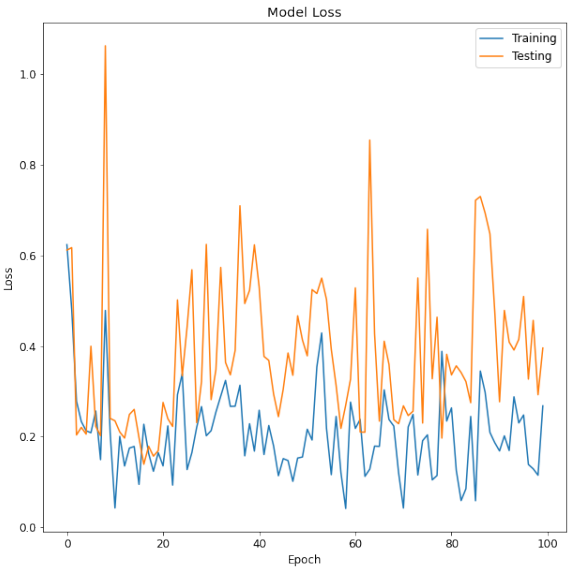


Figure 37a: Model Loss of Inception

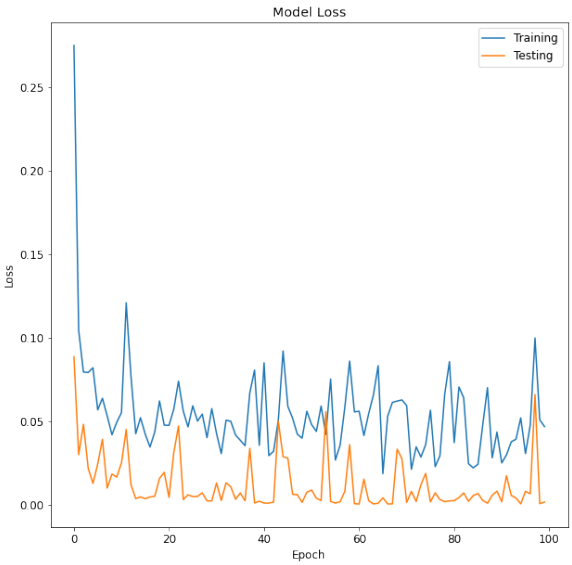


Figure 37b: Model Loss of VGG19

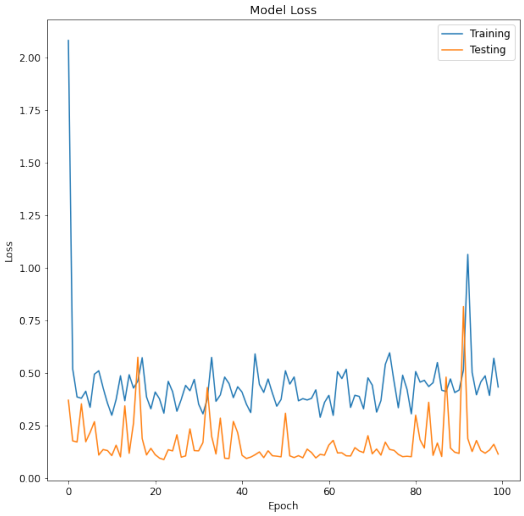


Figure 37c: Model Loss of Resnet50

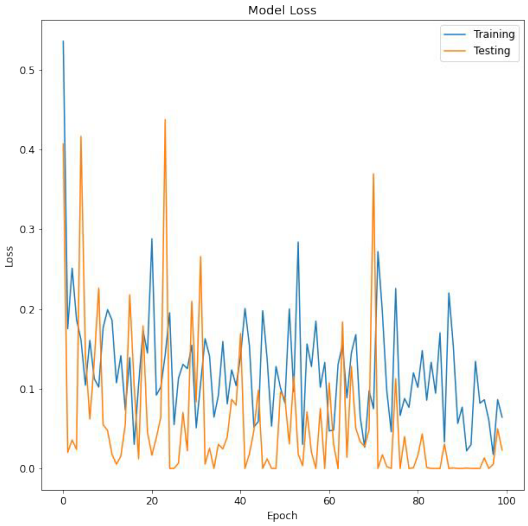


Figure 37d: Model Loss of Xception

Figure 37: Comparison of Loss curves

If we look at Figure 37b, we can see the Training loss is going down and the Test loss curve is also going down. In the end there is an unrepresentative split between train and test data. The curve is jumping up and down because we have limited data. Despite the fact it has slight ups and downs, in the long term, the loss decreases over time, so the model is learning.

The test curve should follow the train curve which happens in all cases therefore showing us that the model is working properly. we can see that test curve is closest to the train curve in Inception and Xception model.

4.6.5 Classification Report

The classification report shows us the accuracy of our model. Classification report also gives us the accuracy, precision, recall, f1 score and support.

	precision	recall	f1-score	support
0	1.00	0.98	0.99	186
1	0.98	1.00	0.99	176
accuracy			0.99	362
macro avg	0.99	0.99	0.99	362
weighted avg	0.99	0.99	0.99	362

Figure 38a: Classification Report of Inception

	precision	recall	f1-score	support
0	1.00	1.00	1.00	186
1	1.00	1.00	1.00	176
accuracy			1.00	362
macro avg	1.00	1.00	1.00	362
weighted avg	1.00	1.00	1.00	362

Figure 38b: Classification Report of VGG19

	precision	recall	f1-score	support
0	0.96	0.98	0.97	186
1	0.98	0.96	0.97	176
accuracy			0.97	362
macro avg	0.97	0.97	0.97	362
weighted avg	0.97	0.97	0.97	362

Figure 38c: Classification Report of Resnet50

	precision	recall	f1-score	support
0	1.00	0.99	1.00	186
1	0.99	1.00	1.00	176
accuracy			1.00	362
macro avg	1.00	1.00	1.00	362
weighted avg	1.00	1.00	1.00	362

Figure 38d: Classification Report of Xception

Figure 38: Comparison of Classification Reports

In Figure 38a, we can see that the accuracy of the predictions using InceptionV3 is 99 percent. That means our model can make predictions which are true 99 out of 100 times. The precision, recall and F1 score are 99% each too. In the same way we can visualize these parameters for the other models too. As a result, we can observe that VGG19 has the best accuracy of the predictions.

4.6.6 Comparison between the Results using X-Ray images

From the given Table 4.3, we can get to know about some of metrics of four models we have tested on our dataset. For our purpose, the model VGG19 and Xception are giving us the best metrics. However, Inception and Resnet50 are not far behind. All four models with respect to different Evaluation metrices are compared below:

Table-4.3: Comparison of performance of different models

Model	Overall Accuracy	Weighted Average				Training Time per epoch seconds
		Precision	Sensitivity	F1 Score	Specificity	
ResNet50	0.97	0.97	0.96	0.97	0.98	15
InceptionV3	0.99	0.99	0.99	0.99	0.99	15
Vgg19	1.00	1.00	1.00	1.00	100	17
Xception	1.00	1.00	1.00	1.00	1.00	15

4.7 Flask Implementation in our model

Initially we define the flask app. Inside the flask app we have:

1. Loading the model

We import our trained model path and give it in MODEL_PATH

MODEL_PATH will read the h5 file directly from our models' folder

Then we load the model

2. Defining the app route

The first app route is for our root folder. This app route will only show us our home page present in index.html. Index.html is present inside the template folder.

By default, render template will be index.html for our root path

3. Defining the predict path

A function upload is created. As soon as the person uploads the picture in the web app, firstly it is stored inside a path called as uploads in our folder.

As soon as it gets stored model_predict function is hit which has our file path and our loaded model.

4. Defining model_predict function

The image is loaded from the image path, converted into an array, dimensions expanded and pre-processed input.

Then we predict using model. Predict

model_predict returns 0 or 1

If 0 then it is COVID-19 image and if 1 then it is a normal image

In this chapter, we have worked on Covid19 detection of CT scan images and Xray mages using 4 different deep learning models. We trained our dataset using Inception V3, Vgg19, Resnet50 and Xception for 500 epochs for CT and 100 epochs for X-ray with a batch size of 32. Then we evaluated our model by plotting the ROC curves, Confusion Matrices, Classification Reports, Accuracy Graphs and the Loss graphs. We compared the results and concluded that for CT scan images, the model trained with Xception and Inception V3 gave the best results and for Xray images, the model trained with Xception and VGG19 gave the best results. Finally, in section 4.7, we deployed the models in our local environment using a flask app where we can upload an image and see the prediction by our model.

Chapter 5

Conclusion & Future work

5.1 Concluding Remarks

Diseases are part of living beings. Every animal on the world will be affected by diseases at some point of their life. Diseases can either be deadly or not. But without proper care and treatment small diseases can turn up to be life threatening disease too. The most fatal diseases among humans in the current world are Cancer, malaria, tuberculosis, hepatitis and so on. And also, not to forget the deadliest of the current times, COVID-19. The detection of diseases is important part of disease treatment. We can go for cure only when we will know that one is affected with some disease. The detection of disease for mass number people is too costly and time consuming too. In recent times we have seen the problems we have to face to detect COVID-19 in over populated areas like Bangladesh. Testing kits and instruments were also sent in limited numbers. From this experience the scientists and programmers came up with the idea to detect diseases like Malaria, Tuberculosis and COVID-19 using images samples of X-ray and CT-scan to detect the presence of the viruses. In our report we worked with detecting malaria and COVID-19 using various CNN models and flask method. Our results were pretty decent and acceptable. We used the datasets obtained from public Kaggle repositories. The objective of our report was to compare the models to find out which gave the best results using the same samples. We used the test train ratio of 80-20 for all the simulations in our report. Mostly we used Google Collab to run the simulations but also used anaconda software in the beginning. However, there were some problems faced during the simulation. We initially used Spyder to simulate but it took much longer time as the software depended on power supply of the computer. And in Bangladesh load shedding is a very normal thing so we had to run one code for 2 to 3 times minimum to get seamless results. On the other hand, Google Collaboratory was much better as it required very less time than Spyder and was totally online. We could save our files any time and resume from the same point whenever we wanted without any problem. Lastly, we could conduct our simulations properly and evaluate our models. We used the Flask App to detect the presence of virus in samples too.

5.2 Future Work

Disease detection using deep learning has advanced a lot in the recent years. The flask app is one of the outcomes of the process. The high accuracy and speed these models have make us hopeful of having web-based application where you can input an image and our model will be able to predict if the image is of an infected or uninfected sample. Moreover, we plan on developing mobile applications based on deep learning to detect diseases remotely from any parts of the country using mobile devices. We feel this will be a revolution for medical technology and for the whole world as well. Also, we hope that our research will help in the future for anyone who will work with the models and save their time and efforts to choose the best model for disease detection.

References

[1] R. Yamashita, M. Nishio, R. K. G. Do, and K. Togashi, “Convolutional neural networks: an overview and application in radiology,” *Insights Imaging*, vol. 9, no. 4, pp. 611–629, 2018.

[2] A. Khan, A. Sohail, U. Zahoora, and A. S. Qureshi, “A survey of the recent architectures of deep convolutional neural networks,” Artif. Intell. Rev., vol. 53, no. 8, pp. 5455–5516, 2020.

[3] D. Upadhyay, “How a computer looks at pictures: Image classification,” Data Driven Investor, 17-Feb-2019. [Online]. Available: https://medium.com/datadriveninvestor/how-a-computer-looks-at-pictures-image-classification-a4992a83f46b. [Accessed: 05-Jan-2021].

[4] M. Mahmud, M. S. Kaiser, A. Hussain, and S. Vassanelli, “Applications of deep learning and reinforcement learning to biological data,” *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 29, no. 6, pp. 2063–2079, 2018.

[5] D. H. Hubel and T. N. Wiesel, “Receptive fields, binocular interaction and functional architecture in the cat’s visual cortex,” J. Physiol., vol. 160, no. 1, pp. 106–154, 1962.

[6] D. C. Ciresan and A. Giusti, “Deep neural networks segment neuronal membranes in electron microscopy images,” *Nips.cc*.

[7] L. Deng, “Deep learning: Methods and applications,” Found. Trends® Signal Process., vol. 7, no. 3–4, pp. 197–387, 2014.

[8] J. Günther, P. M. Pilarski, G. Helfrich, H. Shen, and K. Diepold, “First steps towards an intelligent laser welding architecture using deep neural networks and reinforcement learning,” Procedia technol., vol. 15, pp. 474–483, 2014.

[9] U. Udofia, “Basic overview of convolutional Neural Network (CNN),” DataSeries, 13-Feb-2018. [Online]. Available: https://medium.com/dataseries/basic-overview-of-convolutional-neural-network-cnn-4fcc7dbb4f17. [Accessed: 06-Jan-2021].

[10] A. Deshpande, “A beginner’s guide to understanding convolutional neural networks,” Github.io. [Online]. Available: https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks/. [Accessed: 07-Jan-2021].

[11] Chris, “What is weight initialization?” *Machinecurve.com*, 22-Aug-2019. [Online]. Available: https://www.machinecurve.com/index.php/2019/08/22/what-is-weight-initialization/. [Accessed: 07-Jan-2021].

[12] S. Ruder, “An overview of gradient descent optimization algorithms,” *arXiv [cs. LG]*, 2016.

[13] J. Brownlee, “Gentle introduction to the Adam optimization algorithm for deep learning,” Machinelearningmastery.com, 02-Jul-2017. [Online]. Available: https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/. [Accessed: 07-Jan-2021].

[14] J. Brownlee, “How to use ROC curves and precision-recall curves for classification in python,” Machinelearningmastery.com, 30-Aug-2018. [Online]. Available: https://machinelearningmastery.com/roc-curves-and-precision-recall-curves-for-classification-in-python/. [Accessed: 07-Jan-2021].

[15] “Confusion Matrix,” Sciencedirect.com. [Online]. Available: https://www.sciencedirect.com/topics/engineering/confusion-matrix. [Accessed: 07-Jan-2021].

[16] F. Jiang et al., “Artificial intelligence in healthcare: past, present and future,” Stroke Vasc. Neurol., vol. 2, no. 4, pp. 230–243, 2017.

[17] A. Esteva et al., “Dermatologist-level classification of skin cancer with deep neural networks,” Nature, vol. 542, no. 7639, pp. 115–118, 2017.

[18] “CNN Architectures over a timeline (1998-2019),” Aismartz.com, 24-Oct-2019. [Online]. Available: https://www.aismartz.com/blog/cnn-architectures/. [Accessed: 08-Jan-2021]

[19] L. Li and X. Zhang, “Malaria,” in Radiology of Infectious Diseases: Volume 2, Dordrecht: Springer Netherlands, 2015, pp. 385–406.

[20] “Dengue and severe dengue,” Who.int. [Online]. Available: https://www.who.int/news-room/fact-sheets/detail/dengue-and-severe-dengue. [Accessed: 09-Jan-2021].

[21] J. R. Schellenberg, T. Smith, P. L. Alonso, and R. J. Hayes, “What is clinical malaria? Finding case definitions for field research in highly endemic areas,” Parasitol. Today, vol. 10, no. 11, pp. 439–442, 1994.

[22] D. S. Tarimo, J. N. Minjas, and I. C. Bygbjerg, “Malaria diagnosis and treatment under the strategy of the integrated management of childhood illness (IMCI): relevance of laboratory support from the rapid immunochromatographic tests of ICT Malaria P.f/P.v and OptiMal,” Ann. Trop. Med. Parasitol., vol. 95, no. 5, pp. 437–444, 2001.

[23] G. Díaz, F. A. González, and E. Romero, “A semi-automatic method for quantification and classification of erythrocytes infected with malaria parasites in microscopic images,” J. Biomed. Inform., vol. 42, no. 2, pp. 296–307, 2009.

[24] C. Ohrt, Purnomo, M. A. Sutamihardja, D. Tang, and K. C. Kain, “Impact of microscopy error on estimates of protective efficacy in malaria-prevention trials,” J. Infect. Dis., vol. 186, no. 4, pp. 540–546, 2002.

[25] D. Anggraini, A. S. Nugroho, C. Pratama, I. E. Rozi, V. Pragesjvara, and M. Gunawan, “Automated Status Identification of Microscopic Images Obtained from Malaria Thin Blood Smears using Bayes Decision: A study case in Plasmodium Falciparum,” Asnugroho.net. [Online]. Available: http://www.asnugroho.net/papers/icacsis2011-anggraini.pdf. [Accessed: 09-Jan-2021].

[26] C. Di Rubeto, A. Dempster, S. Khan, and B. Jarra, “Segmentation of blood images using morphological operators,” in Proceedings 15th International Conference on Pattern Recognition. ICPR-2000, 2002.

[27] K. M. F. Fuhad, J. F. Tuba, M. R. A. Sarker, S. Momen, N. Mohammed, and T. Rahman, “Deep learning based automatic malaria parasite detection from blood smear and its smartphone-based application,” Diagnostics (Basel), vol. 10, no. 5, p. 329, 2020.

[28] J. Williams, “Automatic Malaria Diagnosis system,” Mpg.de. [Online]. Available: https://ei.is.mpg.de/publications/mehabbiza13. [Accessed: 09-Jan-2021].

[29] M. S. Alam et al., “Real-time PCR assay and rapid diagnostic tests for the diagnosis of clinically suspected malaria patients in Bangladesh,” Malar. J., vol. 10, no. 1, p. 175, 2011.

[30] Europepmc.org. [Online]. Available: https://europepmc.org/article/med/25889613. [Accessed: 09-Jan-2021].

[31] C. Wongsrichanalai, M. J. Barcus, S. Muth, A. Sutamihardja, and W. H. Wernsdorfer, “A review of malaria diagnostic tools: microscopy and rapid diagnostic test (RDT),” Am. J. Trop. Med. Hyg., vol. 77, no. 6 Suppl, pp. 119–127, 2007.

[32] K. M. F. Fuhad, J. F. Tuba, M. R. A. Sarker, S. Momen, N. Mohammed, and T. Rahman, “Deep learning based automatic malaria parasite detection from blood smear and its smartphone-based application,” *Diagnostics (Basel)*, vol. 10, no. 5, p. 329, 2020.

[33] C. Ramakers, J. M. Ruijter, R. H. L. Deprez, and A. F. M. Moorman, “Assumption-free analysis of quantitative real-time polymerase chain reaction (PCR) data,” Neurosci. Lett., vol. 339, no. 1, pp. 62–66, 2003.

[34] G. Snounou, S. Viriyakosol, W. Jarra, S. Thaithong, and K. N. Brown, “Identification of the four human malaria parasite species in field samples by the polymerase chain reaction and detection of a high prevalence of mixed infections,” Mol. Biochem. Parasitol., vol. 58, no. 2, pp. 283–292, 1993.

[35] World Health Organization. Roll Back Malaria and United States. Agency for International Development, “New perspectives: malaria diagnosis: report of a joint WHO/USAID informal consultation, 25-27 October 1999,” World Health Organization, 2000.

[36] Y. Dong et al., “Evaluations of deep convolutional neural networks for automatic identification of malaria infected cells,” in 2017 IEEE EMBS International Conference on Biomedical & Health Informatics (BHI), 2017, pp. 101–104.

[37] S. Wold, K. Esbensen, and P. Geladi, “Principal component analysis,” Chemometr. Intell. Lab. Syst., vol. 2, no. 1–3, pp. 37–52, 1987.

[38] J. A. Quinn, R. Nakasi, P. K. B. Mugagga, P. Byanyima, W. Lubega, and A. Andama, “Deep convolutional neural networks for microscopy-based point of care diagnostics,” arXiv [cs.CV], pp. 271–281, 2016.

[39] E. I. Obeagu, C. Uo, and E. Is, “Malaria Rapid Diagnostic Test (RDTs),” *Ann. Clin. Lab. Res.*, vol. 06, no. 04, 2018.

[40] clinic level, “How to use a rapid diagnostic test (RDT),” Who.int. [Online]. Available: https://www.who.int/malaria/areas/diagnosis/rapid-diagnostic-tests/generic_PfPan_training_manual_web.pdf. [Accessed: 09-Jan-2021].

[41] “WHO | New perspectives: malaria diagnosis. Report of a joint WHO/USAID informal consultation (archived),” 2015.

[42] N. Linder et al., “A malaria diagnostic tool based on computer vision screening and visualization of Plasmodium falciparum candidate areas in digitized blood smears,” PLoS One, vol. 9, no. 8, p. e104855, 2014.

[43] J. O. -Ansah, M. J. Eghan, B. Anderson, and J. N. Boampong, “Wavelength markers for malaria (Plasmodium falciparum) infected and uninfected red blood cells for ring and trophozoite stages,” *Appl. Phys. Res.*, vol. 6, no. 2, 2014.

[44] Who.int. [Online]. Available: https://www.who.int/blueprint/10-01-2020-nfr-gcm.pdf. [Accessed: 09-Jan-2021].

[45] S. Chaplin, “COVID -19: a brief history and treatments in development,” Prescriber, vol. 31, no. 5, pp. 23–28, 2020.

[46] S. Chaplin, “COVID -19: a brief history and treatments in development,” *Prescriber*, vol. 31, no. 5, pp. 23–28, 2020

[47] X. Yang, X. He, J. Zhao, Y. Zhang, S. Zhang, and P. Xie, “COVID-CT-Dataset: A CT Scan Dataset about COVID-19,” arXiv [cs.LG], 2020.

[48] T. Araújo et al., “Classification of breast cancer histology images using Convolutional Neural Networks,” PLoS One, vol. 12, no. 6, p. e0177544, 2017.

[49] A. Mikolajczyk and M. Grochowski, “Data augmentation for improving deep learning in image classification problem,” in 2018 International Interdisciplinary PhD Workshop (IIPhDW), 2018, pp. 117–122.

[50] R. Jain, M. Gupta, S. Taneja, and D. J. Hemanth, “Deep learning-based detection and analysis of COVID-19 on chest X-ray images,” Appl. Intell., 2020.

[51] R. Thakur, “Step by step VGG16 implementation in Keras for beginners,” Towards Data Science, 06-Aug-2019. [Online]. Available: https://towardsdatascience.com/step-by-step-vgg16-implementation-in-keras-for-beginners-a833c686ae6c. [Accessed: 10-Jan-2021].

[52] C. Fernandez-Labrador, J. M. Facil, A. Perez-Yus, C. Demonceaux, and J. J. Guerrero, “PanoRoom: From the sphere to the 3D layout,” *arXiv [cs.CV]*, 2018.

[53] P. Dwivedi, “Understanding and Coding a ResNet in Keras,” Towards Data Science, 04-Jan-2019. [Online]. Available: https://towardsdatascience.com/understanding-and-coding-a-resnet-in-keras-446d7ff84d33. [Accessed: 10-Jan-2021].

[54] S. Varangaonkar, “Deploying your Machine Learning Project,” AI Graduate, 07-May-2019. [Online]. Available: https://medium.com/x8-the-ai-community/machine-learning-deployment-final-crucial-step-in-ml-pipeline-16ade930b578. [Accessed: 10-Jan-2021].

[55] J. P. Cohen, P. Morrison, L. Dao, K. Roth, T. Q. Duong, and M. Ghassemi, “COVID-19 Image Data Collection: Prospective Predictions Are the Future,” *arXiv [q-bio.QM]*, 2020.

Appendix A.

An Example of an Appendix

Appendices should be used for supplemental information that does not form part of the main research. Remember that figures and tables in appendices should not be listed in the List of Figures or List of Tables. Refer to the Thesis Template Instructions for more information.