Skin Cancer Detection using Convolutional Neural Network Architechtures

by

Md. Moshiur Rahman
15301103
Ashiqur Rahman
15301070
Anuradha Choudhury
15301124

A thesis submitted to the Department of Computer Science and Engineering
in partial fulfillment of the requirements for the degree of
B.Sc. in Computer Science

Department of Computer Science and Engineering
Brac University
January 2020

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:

Ashiqur Rahman
15301070

Md. Moshiur Rahman
15301103

Anuradha Choudhury
15301124

Approval

The thesis titled "Skin Cancer Detection using Comvolutional Neural Network Architectures" submitted by

1. Md. Moshiur Rahman (15301103)
2. Ashiqur Rahman (15301070)
3. Anuradha Choudhury (15301124)

Of Fall, 2019 has been accepted as satisfactory in partial fulfillment of the requirement for the degree of B.Sc. in Computer Science on December 10, 2019.

Examining Committee:

Supervisor:
(Member)

Amitabha Chakrabarty, PhD
Associate Professor
Department of Computer Science and Engineering
Brac University

Program Coordinator:
(Member)

Md. Golam Rabiul Alam, PhD
Associate Professor
Department of Computer Science and Engineering
Brac University

Head of Department:
(Chair)

Mahbubul Alam Majumdar, PhD
Professor and Chairperson
Department of Computer Science and Engineering
Brac University

Abstract

Skin cancer is one of the most common types of human malignancy in medical sector. Normally, it is being diagnosed visually starting with an initial clinical screening and then possibly followed by dermoscopic analysis, a biopsy and histopathological examination. Application of machine learning is continuously being used to determine the accuracy of detecting different medical problems more effectively. A lot of new techniques have been discovered to fast forward the procedure with having highest percentage of accuracy. In this thesis work, we have proposed a model to detect skin cancer more effectively using image processing with convolutional neural network, a part of deep learning concept under machine learning. The dataset contains almost 3000+ images of the patients having skin diseases classified into two classes, malignant and benign. We have introduced CNN along with its seven different architectures to find the accuracy of the images of skin cancer and performed a comparative analysis to find out the best architecture that suits this type of problem. Among all the architectures Xception performed the best results with having almost 85.303% accuracy to determine skin cancer.

Keywords: Skin cancer, Machine Learning, Image Processing, Convolutional Neural Network, Malignant, Benign, Different CNN architectures, Comparative Analysis, Accuracy.

Acknowledgement

Firstly, all praise to the Great Allah for whom our thesis have been completed without any major interruption. With His blessings we were able to put our best efforts and successfully complete it within time.
Secondly, we would like to convey our gratitude to our supervisor Amitabha Chakrabarty, PhD for his handful contribution throughout the whole phase of our thesis work as well as to write this report. From the very beginning to the end of the work he has guided and inspired us to move forward to our goal. He listened to every problems we faced during the work and pulled us out of that deadlock with his visionary ideas which worked brilliantly.
Thirdly, we are also very much thankful to our parents, friends, and well-wishers who have helped us directly or indirectly while conducting our research and continuing our work.
Furthermore, We would also like to acknowledge the assistance we received from a number of resources over the internet, journals and papers, specially from the works of our fellow researchers.
Finally, we would like to thank BRAC University for giving us the opportunity to conclude the thesis and for giving us the chance to complete our Bachelor degree.

Table of Contents

List of Figures

List of Tables

Nomenclature

The next list describes several symbols & abbreviation that will be later used within the body of the document

AVG Average

CNN Convolutional Neural Network

$DCNN$ Deep Convolutional Neural Network

DNN Deep Neural Network

$FlLpF$ Fast Local Laplacian Filtering

GPU Graphic Processing Unit

HD Hamming Distance

HDF Hierarchical Data Format

NN Neural Network

$ReLU$ Rectified Linear Unit

$ResNet$ Residual Network

RGB Red Green and Blue

SF Sigmoid Function

SGD Stochastic Gradient Descent

VGG Visual Geometry Group

$Xception$ Extreme Inception

Chapter 1

Introduction

This chapter provides an extensive overview of the research work and also about the entire thesis. Machine learning is the implementation of algorithms to analyze, learn and make determination or prediction about data. Deep learning is a subfield of machine learning which works based on the concept of brain's neural network. Artificial neural networks are deep learning models that are based on a collection of connected units called artificial neurons or neurons and each connection between neurons can transmit a signal from one neuron to another. CNN is a version of deep learning which develops a model that takes image as input and processes the image to extract the features of that image by recognizing the patterns of it. Moreover if a new image is given, what CNN does is it try to detect the patterns and similarities of an image as best and as accurately as possible. In this research work, we tried to determine from different types of the skin disease to predict the accuracy of whether it can lead to skin cancer or not by classifying images into benign and malignant using various architectures of CNN and also we tried to do a comparative analysis of those architectures.

1.1 Motivation

The advantages of machine learning have been comprehensively used in medical care. Different sectors of medical field have implemented different machine learning algorithms for disease prediction, classification, medicine and cure recommendations. Skin cancer is the nineteenth most common cancer in the worldwide. There were nearly 300,000 new cases in recent year. Non-melanoma skin cancer is the 5th most commonly occurring cancer in men and women, with over 1 million diagnoses worldwide in 2018.[22]. Usually, it is treated visually beginning with an initial medical examination and then possibly followed by dermoscopic analysis, a biopsy, and histopathology [7].CNN uses sophisticated algorithms to learn features from a large volume of health care data and then use the obtained insights to assist in diagnosis. The use of CNN has been increased enormously in the last 10 years. The area of CNN is rapidly increasing as the recent development of the processing power is increasing day by day as well. Therefore, CNN models show more improved accuracy and optimal use of computation resources. The aim of the research work is to find a process that can be trained and find extensive results using machine learning; more specifically different CNN architectures to detect skin cancer more efficiently. Hence we are doing a comparative analysis on the different CNN architectures which

already exists and proven to detect patterns and similarities of any images.

1.2 Aims and Objective

- Converting the digital images into a dataset for building training and testing models.
- Applying various Convolutional Neural Network (CNN) models to predict the types of a skin disease to find out the probability whether it would lead into skin cancer or not.
- Analyzing reasons why performance varies between models having tested with the same dataset.

1.3 Thesis Overview

- **Chapter 1** is the introduction of our thesis work. Our motivations and objectives to do this thesis are also mentioned here as well as a short overview of the methodology that has been followed by us.
- **Chapter 2** consists of the literature review where we have demonstrated the background study that we have done for this thesis.
- **Chapter 3** consists of workflow,pre-processing and compilation of the pre-trained models for our thesis.
- **Chapter 4** consists of pre-tained models and implementation of our overall thesis.
- **Chapter 5** contains the conclusion and future work.

Chapter 2

Related Work

In this chapter, seven similar works related to our used techniques have been discussed along with their processes of how they completed their work. A comprehensive study of each of the procedures and the techniques they used like image processing, image data extraction, training, testing have been discussed briefly so that we can understand our stand and what we have to do to achieve our goals. These works helped us to create and train our model and how we can apply our desired algorithms to complete a work that have not been done by anyone else. In the later part, a comprehensive background study have been discussed. In this part, as we completed our whole work using CNN and its different tools, we discussed what is CNN and how it works along with its different essential layers that are necessary to build a model.

2.1 Previous Work

In this progressive world of science and technology, skin cancer is considered as one of the most venturesome cancer to be found in the human body. Detecting skin cancer in abortive stage is fundamental and decisive. With the development of medical science and technology, researchers and scholars have already invented to detect skin cancer in various efficient techniques. To familiarize with the existing systems and works related to our purpose, we have inspected a large number of works and related papers. For subdivision of skin lesion in the input image, prevailing systems either use manual, semi-automatic or fully automatic border detection approaches. The features to execute skin lesion dissection used in several papers are: shape, color, texture, and luminance.

Shivangi Jain et al. represented a thoughtful procedure regarding to skin cancer detection[1]. They proposed a method to detect melanoma by using image processing. The images they used as input of their system were of skin lesions which was suspected to had skin cancer. At the first step, they pre-processed their images and enriched the image quality. For image segmentation they used edge detection and automatic thesholding tecnique. After that, they extracted the geometrical features of the images and performed lesion area analysis. The reason they used geomatrical features was as it was the most dominant features of the lesions. They used the extracted features to classify skin lesions whether it had cancer or not by comparing the features extracted with the predefined thesholds. They created skin lesion mask

and used it as input images to get the segmented images. The segmented images of skin lesions could detect melanoma quite precisely.

Vijayalakshmi M M et al. proposed a methodology of a fully automatic system to determine lesions using dermatological infection recognition, an approach by machine in opposed to detect the lesions visually by medical expert[19]. His model was designed in three phases. The first phase was to compromising of data collection and data augmentation. As the image pre-processing tecnique they performed shade removal, glare removal and hair removal and in that way they were able to identify features like shape, color, size and texture. The second phase was about to design their architecture and it consisted of feature extraction and segmentation where segmentation was done by three tecniques which are Otsu segmentation tecnique, modified Otsu segmentation technique and Water shed segmentation tecnique. The last and most important phase was to predict the lesions by designing and training the architectures. They used Back Propagation Algorithm, SVM and CNN to train their dataset.

E. Nasr-Esfahani et al. proposed an approach of deep learning system using a server with graphic processing unit (GPU), which was proposed for recognition of melanoma lesions [5]. In their proposed system, they used input images which have illumination and noise effects.They used their system to reduce those occurances. In that way, they increased the quality of the images and subsequently used those images in a pre-trained convolutional neural network architecture. Their proposed method performed better in finding accuracy than existing latest pre-trained CNN architectures. They built an automatic diagnosis system using deep learning methods to detect melanoma in skin. Since their dataset had low number of images, they performed data augmentation such as resizing, cropping and rotating the images. The output result of their proposed system increased exceptionally by it.

M.H. Jafari et al. proposed a method for precise extraction of lesion region which is based on deep learning methodologies [3] . To lessen noisy artifacts, the input image, after pre-processing, was applied to DCNN. CNN produced a segmentation mask which detected the area of the lesion. Using some post prossecing operations the quality image of the mask is being further improved. Their input images were produced by customary cameras; henceforth, those were pre-hprocessed in order to manage noisy artifacts. They used a filter to decrease the noise of the images. They created two patches, one is local patch and another is global patch. The local patch is illustrated as a window around each pixel. It shows the local texture around the middle pixel. Whereas, the global patch is used to show the global structure of the area. After that, both the local and global textures are sent into the proposed CNN architecture as the training input. The experimental outcomes showed that their suggested method can outpace other architectures to detect lesions.

Tanzila Saba et al. proposed a innovative automated methodology to detect and recognize skin lesions using DCNN[20]. They performed three steps. The first step was to inhance the contrast of the images using first local laplacian filtering with HSV color convertion. The second step was to extract lesion boundary using XOR operation of color CNN methodology. The final step was to extract the features by applying transfer learning by using InceptionV3 in order to feature fusion using hamming distance technique. They introduced a feature selection technique by

controlling entropy which was presented for the collection of the most discriminant types. Their proposed system was varified by different datasets. They achieved excellent accuracy using their proposed system.

V. Pomponiu et al. proposed a methodology that used a pre-trained DNN to spontaneously detect the features which can later be used to find out malignancy from any skin cancer dataset[6]. Their out put results succeded the current existing DNN based architectures. The main part of their system was a pre-trained CNN architecture to extract features from their dataset. Their aim was to create a CNN architecture which can be determined by various factors such as the type and the size of the dataset as an alternative fine tuning technique of CNN. Since their dataset was slightly small, they constucted a classifier model on above of the out result of the hidden layers. The classifier is built with the values of the activations from previous layers of the network. Their proposed technique performed better than employing hand tuned features. Additionally, it was very fascinating to observe that, even though the CNN has been trained with different datasets each time including the datasets that are not related to skin cancer performed very well to extract features. The reason it performed so extraordinary was that, it could change the skin tones because of the lighting conditions.

Jeremy Kawahara et al. established a linear classifier, trained on features extracted from a convolutional neural network pre-trained on natural images, differentiates between up to ten skin lesions with a greater accuracy [4]. Their methodology needs neither lesion segmentation nor complex pre-processing. They achieved a satisfactory accuracy over 5-classes with visible progresses in accuracy for underrepresented classes. Their technique attained a great accuracy outperforming the accuracy formerly reported. This was an effective methodology to calculating features over dissimilar spatial locations as it reprocesses the conjoint convolutions done previously in the network and can be used to calculate features over multiple scales. Combining features over the spatial dimensions has presented to develop predictive presentation and the resulting feature vectors simplify well to other natural object image tasks.

2.2 Background Study

2.2.1 Covolutional Neural Network

A Convolutional Neural Network is a type of artificial neural network that uses convolutional layers for extracting useful information by filtering inputs [11]. This involves combining feature map with a convolution filter to form a transformed feature map. The kernels are modified based on learned parameters to extract the most useful information from a specific task. Convolutional networks instinctively detect the best components of that task. Convolutional Neural Network has multiple applications in various sectors like image recognition, image classification, text analysis, speech recognition, natural language processing etc. A Neural Network is constructed with one input layer, one or more hidden layers and one output layer. These layers can be dense layer, convolutional layer, pooling Layer, normalization

layer, rectified linear unit layer, fully connected layer etc.Convolutional Neural Network must have at least one convolutional layer inside its layers.
Figure 2.1 shows the whole architecture of a Convolutional Neural Network.

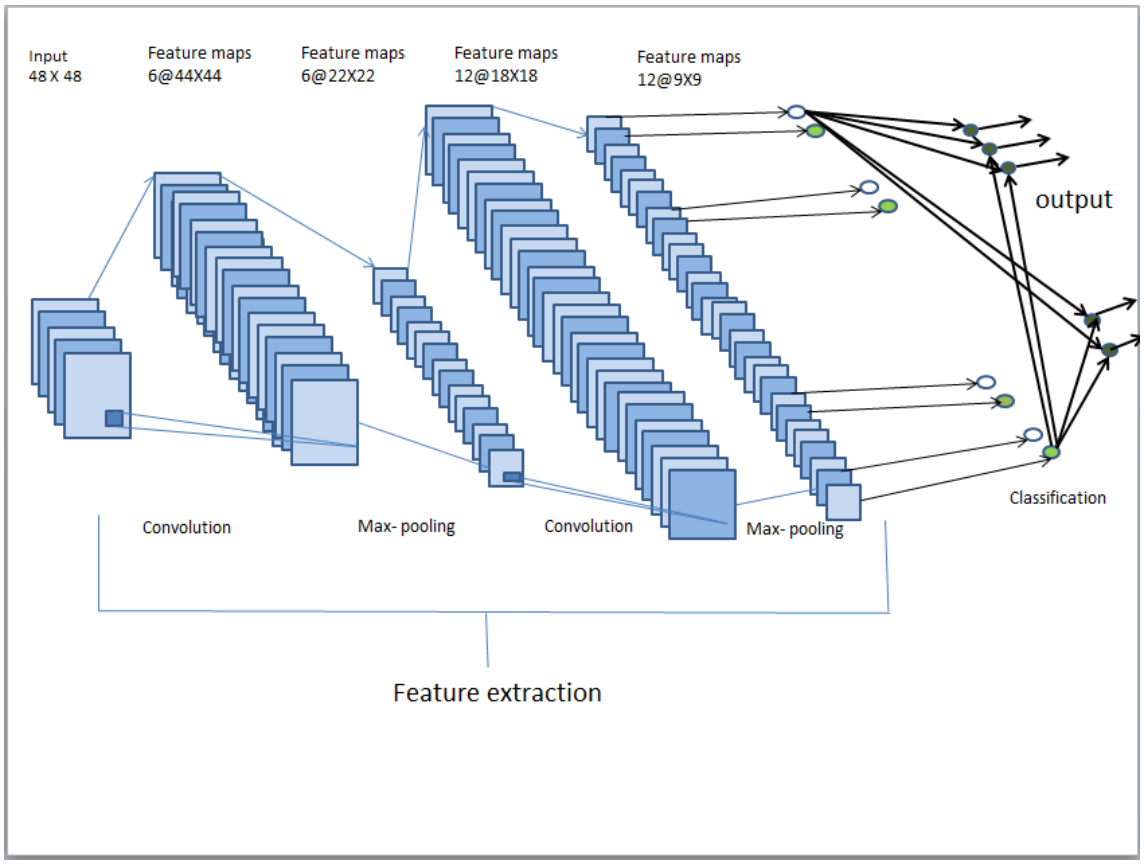

Figure 2.1: Architectures of Convolutional Neural Network

Input Layer

Input Layer in CNN works based on only image data. These images have three dimensions, [width x height x depth] which is a matrix of pixel values. For example, for a input value of (width=32, height=32, depth=3) or [32x32x3] where depth stands for Red green and blue channels or RGB channels. Also the input layer has to be divisible many times by 2. To work with input layer, at first the three dimensional matrix has to be reshaped into a single column. For example, if there is a image of dimension 32x32=1024, it needs to be converted into 1024x1 before passing it into the input. For 'x' training examples the dimension of input will be (1024,x)
Figure 2.2 shows the input layer of a CNN.

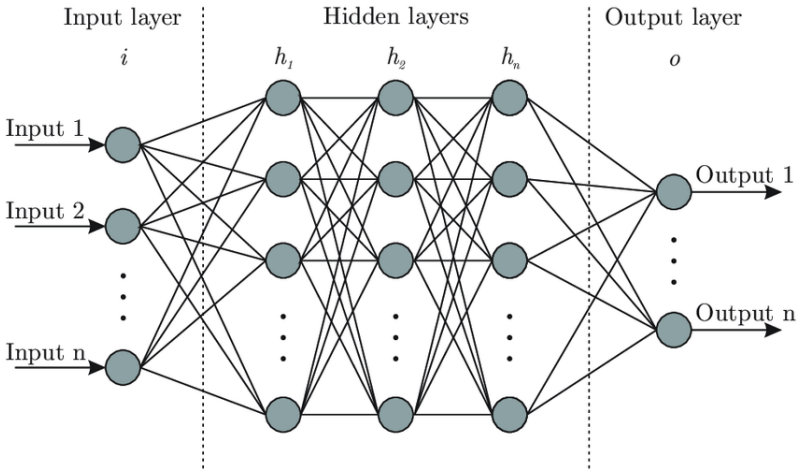

Figure 2.2: Input Layer

Dense Layer

Dense layer is the non-linear layer in neural network. These layers follows the exact formulas as linear layer but the difference is dense layer passes the end result through a non-linear function which is called Activation function. Dense layer can model any mathematical function which is considered as one of the unique properties of it. Although there are some limitations for example for any given input vector, the output will always be the same. Dense layer neither can produce different answers on the same input nor can detect the repetition in time.
Figure 2.3 shows the dense layer of a CNN.

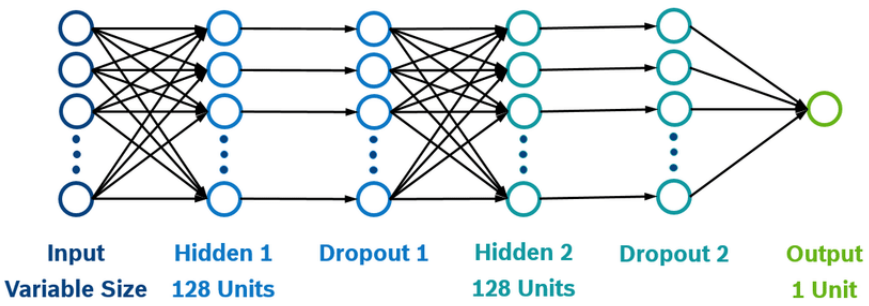

Figure 2.3: Dense Layer

Convolution Layer

Convolution layer is the core building block of a Convolutional Network that does most of the computational heavy lifting. Convolution layer has some parameters and hyperparameters made up with filters from which this layer extracts features and learns from it. In a sentence, this is a feature extractor layer. Input images are compared with segments to find out the changes. These segments are known as features. This layer selects one or more features from the input images and creates one or multiple matrix and dot product with the image matrix. It also calculates the output neurons connected to the local regions. Furthermore the whole computation

gives out a result known as the convolution layer. For a given I x I image size and F x F filter size, the convolution result will be:

$$(I \times I) \times (F \times F) = (I–F + 1) \times (I–F + 1) \tag{2.1}$$

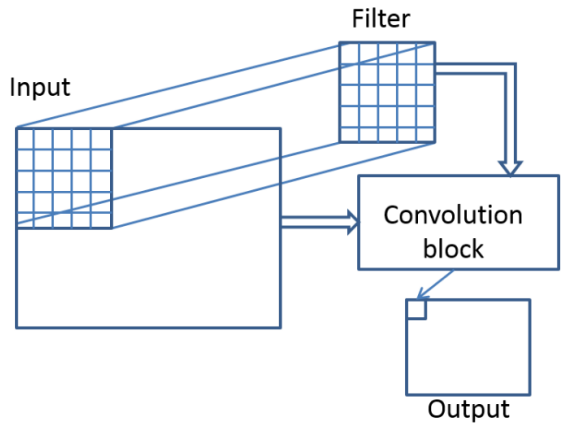

Figure 2.4: Convolution Layer

Rectified Linear Unit Layer

Rectified linear unit or ReLU layer has an activation function which is a piecewise linear function that will output the positive input directly, otherwise it will output zero. After convolution layer passes the result into this layer, it will change all the negative numbers to zero keeping the positive number unchanged. ReLU has become the default activation function of many types of neural networks. Model initiated with ReLU can be trained easily and also it achieves better performances. ReLU layer does not have any parameters or hyperparameters.
Figure2.5 shows the rectified linear unit layer of CNN model.

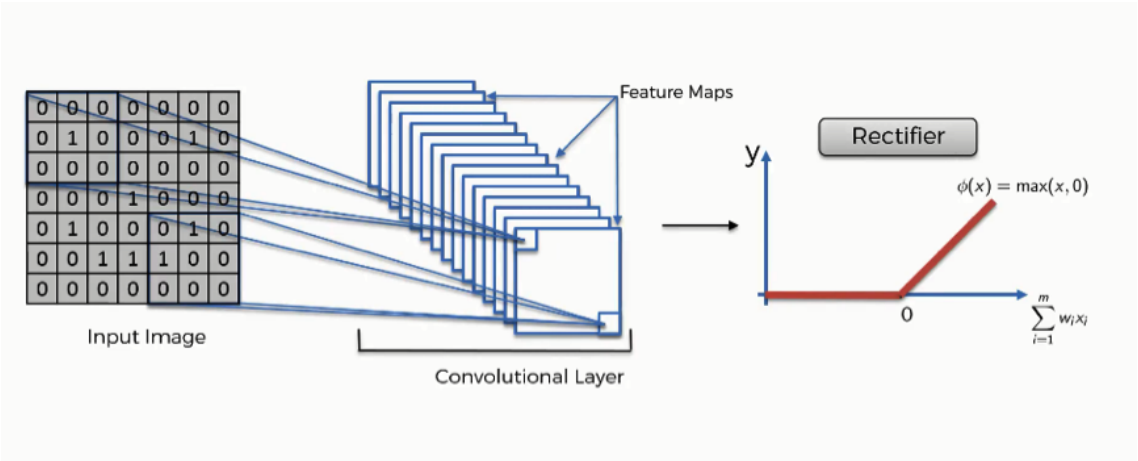

Figure 2.5: Rectified Linear Unit Layer

Pooling Layer

Pooling layer is a building block of a Convolutional Neural Network which works on reducing the spatial size of the representation to reduce the amount of parameters and computation in the network. Pooling layer is not dependent on the functionality or the quantity of features in the network means it can operate in each feature map independently. The most popular implementation used in pooling is known as max pooling. Pooling layer does not have any parameters but it has additional hyperparameters: Filter(F) and Stride(S). If the input dimension is X1 x Y1 x Z1,

$$X2 = \frac{X1 - F}{s + 1} \tag{2.2}$$

$$Y2 = \frac{Y1 - F}{s + 1} \tag{2.3}$$

$$Z2 = Z1 \tag{2.4}$$

Figure 2.6 shows the pooling layer of a CNN model and the max pooling mechanism.

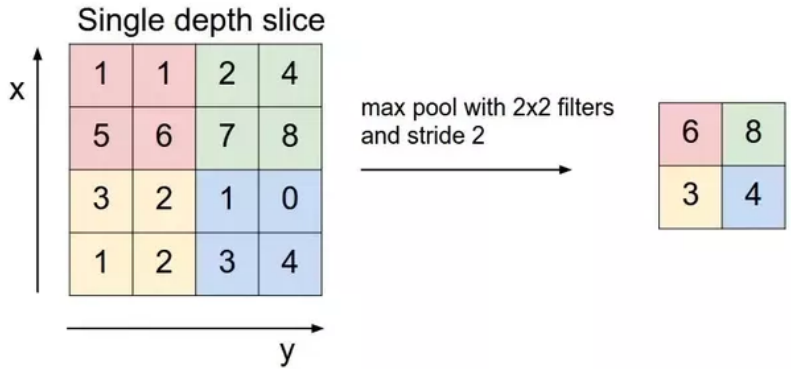

Figure 2.6: Pooling Layer

Normalization Layer

Normalization layer basically normalizes the activation outputs of previous layer which means it puts in a transformation that upholds the mean activation close to zero and the activation standard derivation close to 1. Batch normalization is mostly used here although there are many other alternatives like weight normalization, layer normalization, instance normalization, group normalization, spectral normalization etc.
Figure 2.7 shows the normalization layer and various normalization methods.

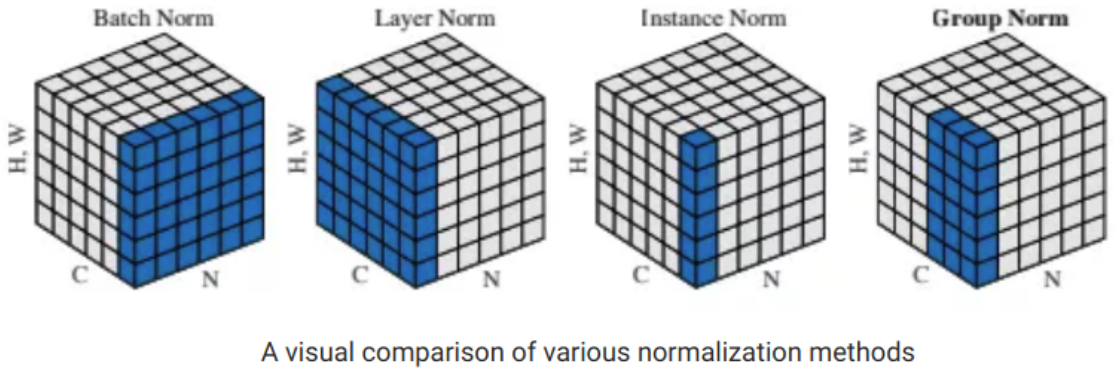

Figure 2.7: Normalization Layer

Fully Connected Layer

Fully connected layer normally takes an input volume and outputs an N dimensional vector where N is the number of classes that have been chosen from the program. The input volume comes from the output of convolution or relu or pool layer. After analyzing the outputs of the previous layer fully connected layer distinguishes the mostly matched features of a particular class. Fully connected layer works on features which are high level and have particular weights.In this way fully connected layer calculates the odds for the different classes as it calculates the product outputs of the weights and the previous layer.
Figure 2.8 shows the fully connected layer of a CNN model.

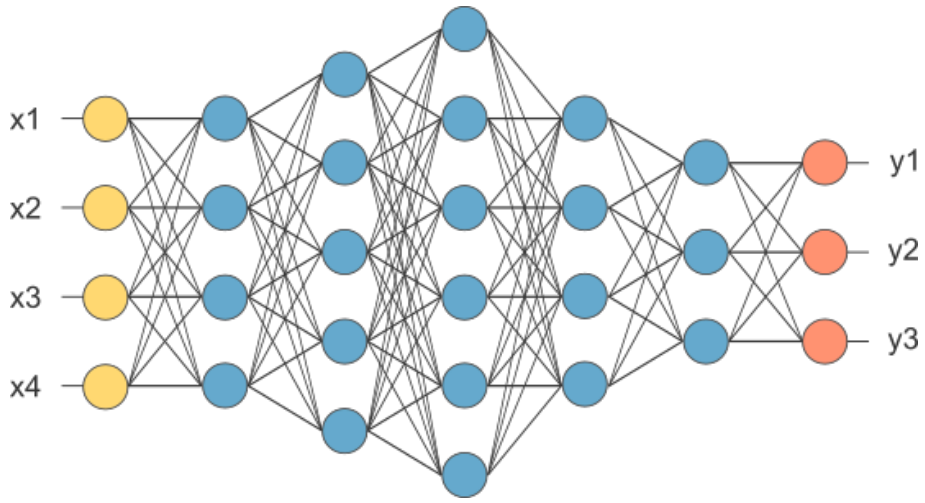

Figure 2.8: Fully Connected Layer

Output Layer

Output layer is the last layer in cnn which produces the final output for the program. Often the design of output layer neuron is modified to understand the result as well the whole program. After all the data is trained properly and the weights have been saved, for a new input it will check the similarities and determine whether the input

matches with the train data or not.In a sentence output layer coalesces and correctly produces the end result.
Figure 2.9 shows the output or the final layer of a CNN model.

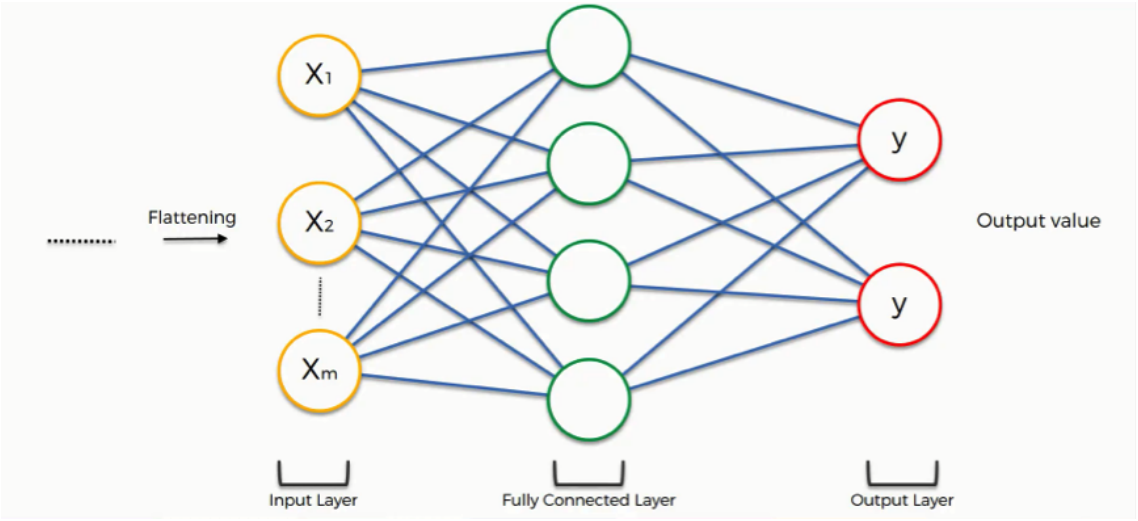

Figure 2.9: Output Layer

In this way, CNN takes, process, trains and thus predicts for the given inputs.

Chapter 3

Workflow and Feature Extraction

In this chapter, the whole procedure of our work have been discussed with how we solved the problem to build the model and then how we extracted our dataset which can be in a shape that can be used with our created model. Firstly an image of our proposed system have been shown so that a clear idea of what we are going to do can be understood. After that there is dataset description to describe our dataset and what it consists of. Then, the way of our dataset pre-processing techniques have been showed along with the compiling processes with describing all the parameters that have been used to compile our model.

3.1 Workflow

At the beginning of our work, we have collected our dataset at first. We completed our whole work on Jupyter Notebook. Thus, before doing anything we have imported the necessary libraries which will be needed to complete our work. We already had our train and test images in different folders. Then we pre-processed our dataset according to our needs. After that, we created and run a CNN model with our dataset and observed the accuracy. Then, we again created and run CNN model with KFold architectures by splitting the dataset into 3 folds and then run the CNN architectures 3 times considering one of the splits from total 3 splits as validation set each times differently. Then we predicted the accuracy for this model and saved it into disk. After that, by keeping the same variables, we run different CNN architectures to find out and predict the accuracies to compare the overall result. We ran ResNet50, VGG16, InceptionV3, VGG19, Xception, MobileNetV2, MobileNet architectures to find out the best model for that suits our type of dataset. Figure 3.1 shows our proposed system.

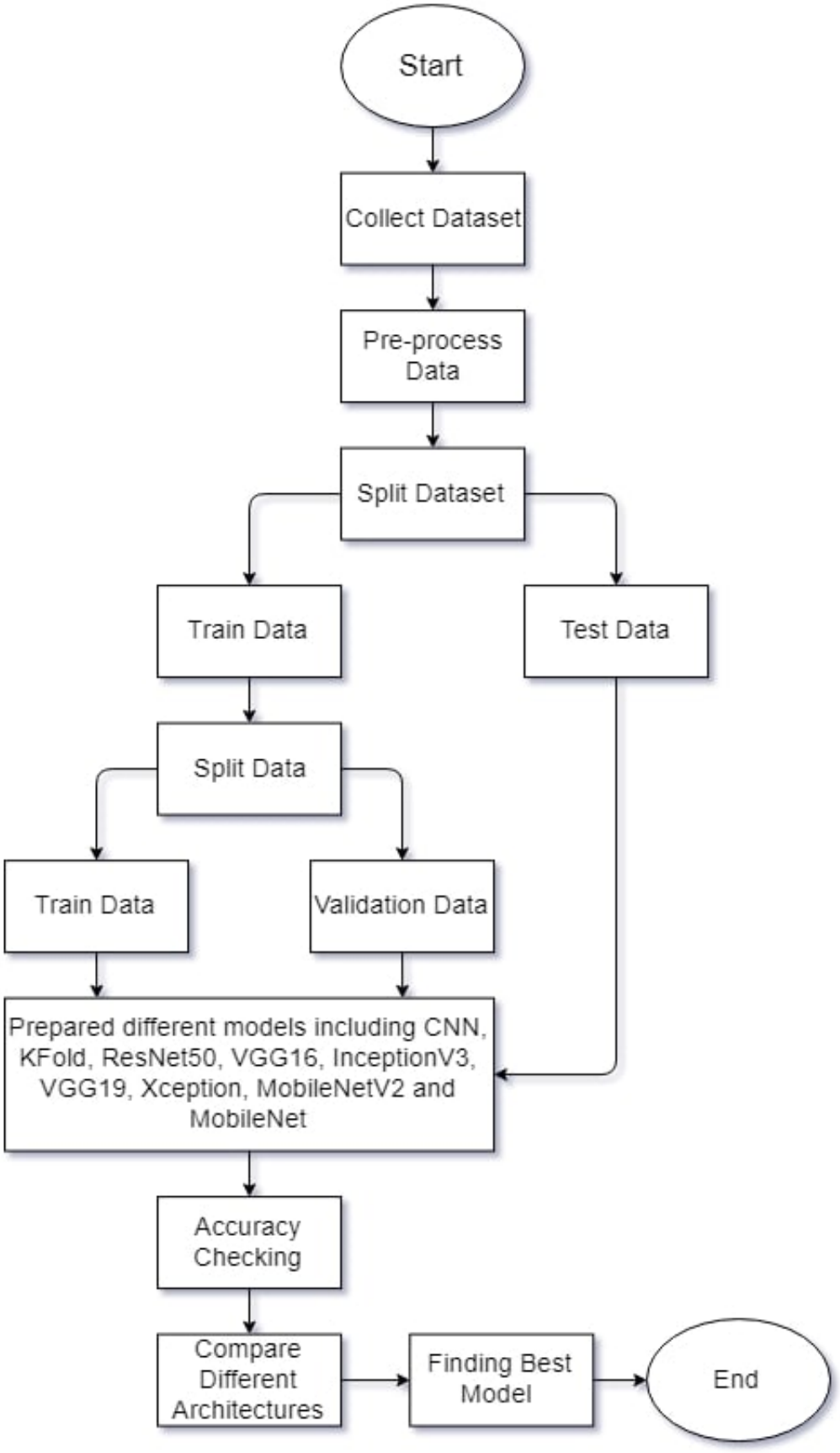

Figure 3.1: Proposed System

3.2 Dataset Description

The dataset we used for our thesis is a secondary dataset. We collected it from kaggle. The dataset is known as "Skin Cancer: Malignant vs Benign". The dataset contains a balanced dataset of images of benign skin moles and malignant skin moles. It consists of total 3300 pictures. This dataset was obtained from ISIC-Archive. The dataset is divided into two main folders named "test"(660 pictures) and "train"(2640 pictures) with each in the folder having two more sub folders named "malignant" and "benign". There are total 1440, 1200, 360, 300 pictures in the train>benign, train>malignant, test>benign , test>malignant folders respectively. In each of the folders above have the images with a unique id of each images. For the KFold architecture we split the whole train set into 3 splits and used 1 split as validation split differently 3 times.
Figure 3.2 shows sample images from our dataset

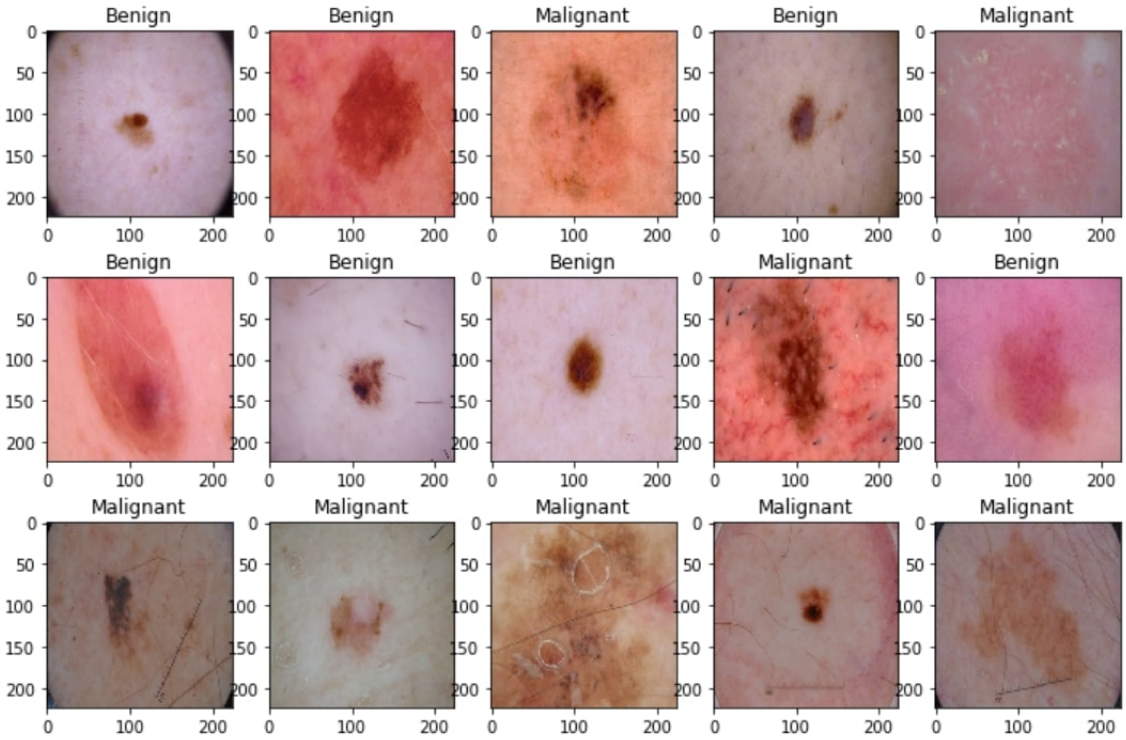

Figure 3.2: Sample Images from our Dataset

3.3 Pre-Processing

3.3.1 Image Pre-processing in Python

The images that we have in our dataset are all have the same resolution of 224x224 pixels. At first, we read all the images as list of arrays and then we converted the images into RGB images. After that, we converted the list of arrays to Numpy Arrays with having data type "uint8" to each one of them. We labeled the benign images as 0 and malignant images as 1. Furthermore, we concatenated the all benign and malignant images from test and train folders respectively to two different arrays

as test and train set. As Normalization process, we divided the train and test arrays by 255 so that every value would stay in a same measurement between 0 to 1, where closer to 1 would represent malignant and closer to 0 would represent benign.

3.4 Compiling Model

After completing the data pre-processing we worked on compiling the model. The input shape of the input images from the dataset is now 224X224X3 as we converted it in this way in the data pre-processing. We tried couple of values for the learning rate of our model and we found that $1/\,10^3$ as the learning rate value gives the best output. We choose 2 classes for our inputs. Moreover as activation function we choose the 'relu' function and as an optimizer we used 'adam'. Just before the end, while compiling the model we used 'binary-crossentropy' as our loss function and defined 'accuracy' as our metrics. At the end, we used another function named 'learning-rate reduction' if the value of learning rate keeps getting less than $1/\,10^7$ to keep balance of the learning rate. We did the compile model step while we run the different CNN architectures as well. A general description of the parameters that we used to compile the models are given below :

3.4.1 Activating Fully Connected Layer

The fully-connected layer at the top of the network has been activated by keeping the include top true. Fully connected layers at the end can only take fixed input sizes, but it works on the input image which is not ideal for images though. The fully connected network works well with all images, but if a cropped image is given as the actual object, it can not be easily classified with a fully connected network. In order to achieve a better result, the individual features of the object should be defined, such as the types of lines and curves in the image, so that the objects can be categorized with partial information. Activating the fully connected layers can result in over-fitting of the entire model which means that for known instances it will perform well, but for unknown instances it will perform badly. The specic size of images only can be given when the fully connected network is false. Otherwise, it will be converted into the size which is xed by the models. However, as we have enough number of images so over-fitting will not occur for this model, that's why we kept the include top as true.

3.4.2 Weight Selection

For our research we used the weights null. The other weights that can be used are any pre-trained weights or the path to the weight file to be loaded as the weights. We trained the model from nothing so that it can learn all by itself and then compare each of the models with one another and find out the best model for this project.

3.4.3 Pooling Layer selection

For the pooling layer in our model we used Max Pooling 2D. There are many types of pooling layers available but among them max pooling and average pooling are the most common and most widely used. If we need to detect patterns from all

over the image, in that case average pooling is a better option as it takes average value of the pixel from the defined matrix. But, as we are detecting infections from a body part of a patient which is only a portion of the whole image we are using for max pooling. Besides, max pooling is better choice here because it extracts the sharpest features of an image. It extracts the most important features like edges of patterns whereas, average pooling extracts features smoothly. If we don't define any parameters Keras takes pool size 2X2 and a stride value of 2 by default. However, for the simplicity of the code, we defined the pool size 2X2. Means, from a 2X2 matrix, max pooling will take the highest pixel value and replace it with one single pixel for the output image.

3.4.4 Dropout Selection

Dropout is a technique of regularization in CNN to prevent the over-fitting problem of the dataset in the training process. What dropout does is it reduces the complexity of the model by the given value of dropout. Dropout helps a neural network to extract more robust features while reducing the training time. In our model we used 0.25 as the value of dropout. Means, it will randomly ignore 25% of the neurons while training. The drop out value can change model to model depends on the approaches to solve a problem. We used different values for the dropout to check the best output and then we found that a value of 0.25 works the best.

3.4.5 Activation Function Selection

In an artificial neural network, what activation function does is, a input comes prom a previous neuron and the activation function processes the input and creates a output for that specific neuron which is being used for the input for the next neuron. There are several activation functions available. However, in our proposed system we only used two types of activation functions, the Sigmoid Function and the Relu. Function.

Sigmoid Function: In a sigmoid function, the more negative the input is, the closer to 0 the output will be. In a same way, the more positive the number is, the closer to 1 the output will be. The range will vary from 0 to 1.

$$S(x) = \frac{e^x}{e^x + 1} \tag{3.1}$$

Figure 3.3 shows the sigmoid curve for any input.

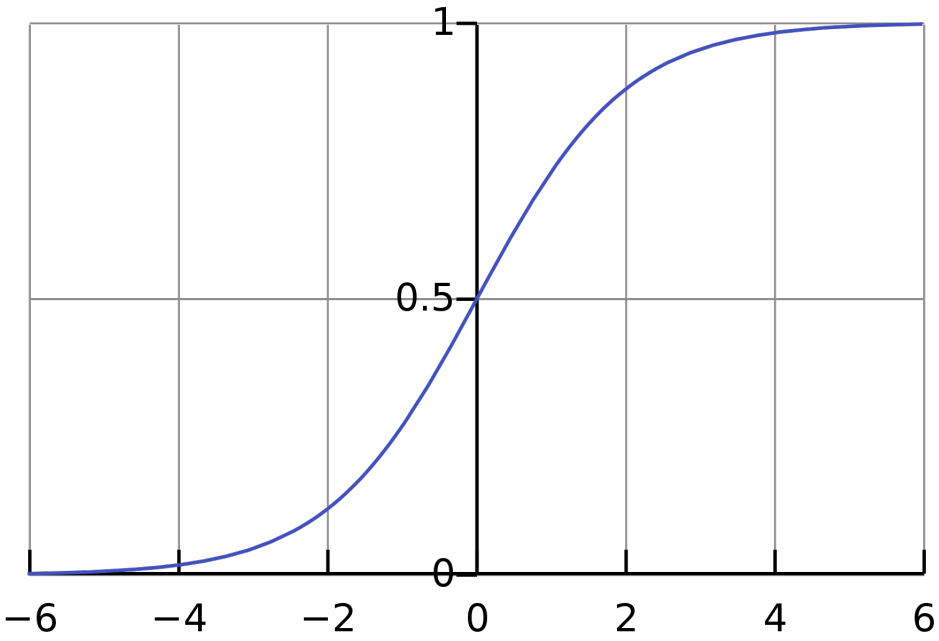

Figure 3.3: Sigmoid Function

Softmax Function: Softmax activation function is a type of sigmoid function which can be used in classification problems. Normal sigmoid function can handle two classes. But what if we need to handle multiple classes? Then the softmax function comes to save the day. The softmax function transforms the outputs of the each class into between 0 and 1.
Relu Function: In a Relu function, what it does is, it converts the value to a scale started from 0 to positive infinity. Means, if the input value is less than or equal to 0, then the output will be 0. Furthermore, if the input is greater than 0, the output will be the value itself.

$$f(x) = max(0, x) \tag{3.2}$$

Figure 3.4 shows the relu curve.

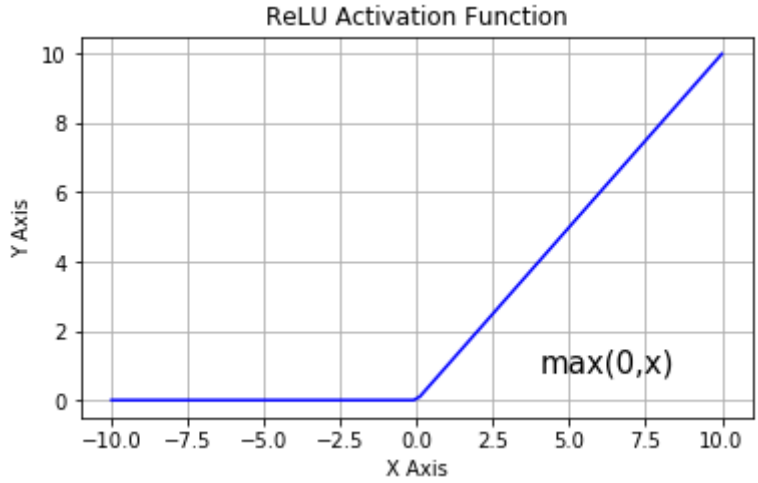

Figure 3.4: Relu Function

3.4.6 Flatten Layer Selection

After using the pooling layer we need to use a flatten layer to flatten the whole network. What flattening does is it transforms the whole pooled feature map matrix into a single column. Then this is being passed to the neural network for further processing.

3.4.7 Loss Function Selection:

For loss function, we have used Binary Cross-entropy.

Binary Cross-entrpy:

Binary classification is the task of classifying the elements of a given set into two groups and predicting which group each one belongs to on the basis of a classification rule. In Binary classification, the typical loss function is the Binary cross-entropy. The loss function returns high values for bad predictions and low values for good predictions. As we are determining the types of skin cancer whether is it benign or malignant, binary cross-entropy can help to calculate the loss and monitor if the architecture is learning properly.

3.4.8 Optimizer Selection:

Optimization algorithms helps us to minimize or maximize an Error function E(x) which is simply a mathematical function dependent on the Model's internal learnable parameters which are used in computing the target values(Y) from the set of predictors(X) used in the model.
We have used three optimizers in our work.

SGD:
Stochastic Gradient Descent (SGD) updates the parameters for each training instances .As it performs one update at a time it is a much fater approach. Because

of these frequent updates, parameters updates have high variance and cause the loss function to fluctuate to different intensities. In order to discover new and possibly better local minima this could be an optimal approach as Standard Gradient Descent will only converge to the minimum of the basin as mentioned above.

Adam:
Adaptive Moment Estimation (Adam) is an approach that calculates adaptive learning rates for each parameter. Adam also keeps a track of past gradient's exponentially decaying average. Adam works better than other adaptive learning-method algorithms since it converges very rapidly and the learning speed of the Model is fast and efficient and also it solves every problem that is found in other optimization methods for example - learning rate losing or at a value very low, lower convergence or extreme variance in the parameter updates. All these methods leads to the fluctuation of loss function.

RMSProp:
RMSProp is an optimization algorithm designed for neural networks which has not been published yet. It is a modified version of Rprop algorithm. Gradients may vary widely in magnitudes and Rprop tries to solve that. Some gradients may be small and others may be huge, which result in a critical problem as it becomes harder to find a single global learning rate for the algorithm. In full-batch learning this problem can be solved by only using the sign of the gradient. After that, all weight updates become the same size. This helps in keeping the saddle points and plateaus big enough even with tiny gradients. But Rprop doesn't really work on very large datasets. In those datasets it needs to execute mini-batch weights updates since it does not cope up wih the idea of stochastic gradient descent In SGD, when learning rate is small; it averages the gradients over next mini-batches. For example, it occurs when the weight with the gradient 0.1 on nine mini-batches and the gradient of -0.9 on tenths mini-batch. With Rprop, the weight grows much larger. The reason behind using RMSprop is to keep the moving average of the squared gradients for each weight. And then the gradient is divided by square root of the mean square. Learning rate is adapted by dividing by the root of squared gradient, but since we only have the estimate of the gradient on the current mini-batch, the moving average of it needs to be used. Default value for the moving average parameter is 0.9. It works very well for most applications.

3.5 Applying the prepared model for testing

After out model is fully prepared, we have tested and checked the dataset to nd out which one is giving more accurate result for our data. Details about it have been described in Chapter 4.

Chapter 4

Implementation and Result

In this chapter, the whole implementation procedure of our model have been thoroughly discussed. We have used CNN with KFold along with 7 other CNN architectures(ResNet50, VGG16, VGG19, InceptionV3, Xception, MobileNetV2, MobileNet) to train and test our dataset. In each of the section, at first the definition of the model have been shown, then the version we have used to train our model of that architecture has been shown as well. Finally, the way we used that specific architectures and the parameters that we have used, have been discussed with images from our work. In each of the section, test accuracy after each 5 epochs until 50 epochs with their loss, accuracy, val-loss, val-accuracy have been shown. Then, an extensive graphical overview (Accuracy vs Epochs curve and Loss vs Epochs curve) of the each model of the test result have been shown along with some sample test images from our dataset and the predicted result of our used architectures have been compared to show an extensive comparative analysis. At the end of this chapter, a comparative analysis of different CNN architectures with their accuracy and loss have been shown with a graphical visualization to compare the architectures more effectively.

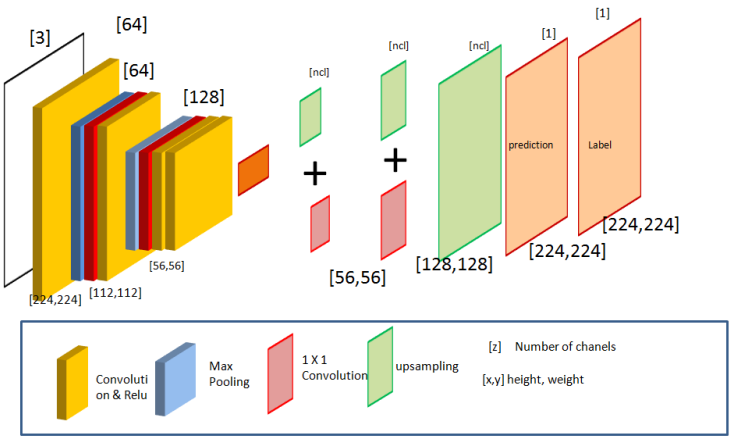

Figure 4.1: CNN Model

4.1 Basic CNN with Cross Validation

4.1.1 Cross Validation Concept

Cross Validation is a very useful method to improve the performance of machine learning models. It helps in knowing how a machine learning model would work on an independent data set. To know about the accuracy of the predictions of a model in training and testing, cross validation comes in handy [10]. Cross validation divides the data into two parts. One is for training and the other is for testing or validating. Cross validation trains the model on training set and tests the predictions against the validation set. Besides, cross validation tests the model in the training phase to find out if there is any over-fitting and also to get the idea how the model would act on independent data. Furthermore cross validation reduces variance when several validations are performed. The results from all the rounds are averaged to find out the accuracy. In short, cross validation is mainly applied to compare the performance of different models on the same dataset. Also to find out the suitable parameters of a model, cross validation is applied. Figure 4.1 shows our basic cnn model and Figure 4.2 shows an example of dividing dataset for cross validation.

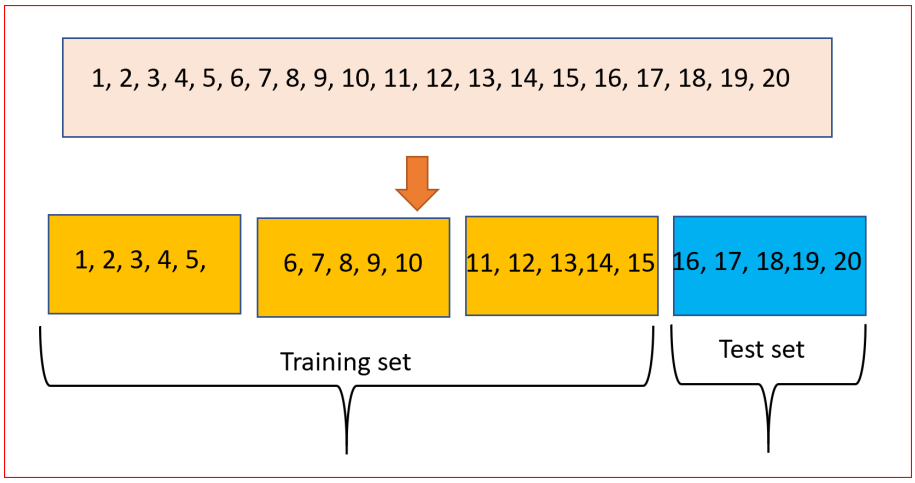

Figure 4.2: An example of Dividing Dataset for Cross Validation

4.1.2 K–Fold

K-fold is the most common cross validation used in machine learning. K – fold divides the input dataset into k equal subsets. Each of them are called a fold. Then it keeps a fold as validation and the rest k-1 folds are used for training.
After the training is complete then it tests the accuracy of the model by validating the predicted results against that fold kept for validation. Then the whole process is followed again with a different validation set, in total k times. That means each entry is used for validation just once. After all the predictions have been calculated and collected, the model estimates the accuracy by averaging the predictions. Thus the over-fitting and under-fitting of the model can be handled since most of the data is used for fitting as well as for validation set.
Figure 4.1.2 shows the k-fold mechanism for the value of k=10.

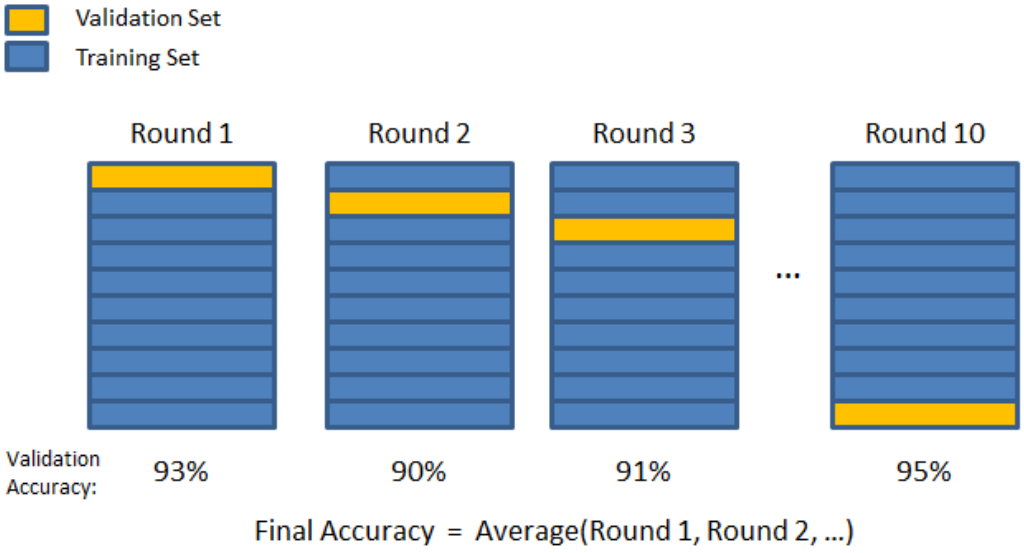

Figure 4.3: K-Fold Mechanism

4.1.3 Applying CNN with Cross-Validation

At the beginning of the training process, we applied CNN with having only two convolutional layer along with two max-pooling and two dropout layer with one of the each. Its just a basic CNN to start the procedure. After that, we used 1 flatten layer. Then we trained our model with this. After we trained our model with this simple CNN, we plotted how our model is reacting to our dataset. After that, we used cross-validation idea to modify our model. To use the idea, we split our dataset into 3 splits. So, now what will happen is, the previous CNN model will run again, but this time with a simple but very effective modification. As we split our dataset into 3 folds, the whole CNN will run 3 times considering one of the folds as a validation set and the remaining two folds as training set. At the first step the first split will be the validation set and the remaining 2 splits will be the training set. In the next step, the 2nd split will be the validation step and the 1 st and 3 rd split will be the training set. At the end step, the last split will be the validation set and the first two will be the training set. As we are splitting our same dataset multiple times with having different validation set, the accuracy will arise. Later we will run various CNN architecture on our dataset to compare the accuracy as well as to find out the best model to perform tasks on such dataset.
Figure 4.4 shows the model we have built to perform CNN on our dataset.

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_3 (Conv2D)            (None, 224, 224, 64)      1792
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 112, 112, 64)      0
_________________________________________________________________
dropout_3 (Dropout)          (None, 112, 112, 64)      0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 112, 112, 64)      36928
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 56, 56, 64)        0
_________________________________________________________________
dropout_4 (Dropout)          (None, 56, 56, 64)        0
_________________________________________________________________
flatten_2 (Flatten)          (None, 200704)            0
_________________________________________________________________
dense_3 (Dense)              (None, 128)               25690240
_________________________________________________________________
dense_4 (Dense)              (None, 2)                 258
=================================================================
Total params: 25,729,218
Trainable params: 25,729,218
Non-trainable params: 0
```

Figure 4.4: Build Architecture of CNN model

The table above shows the loss,accuracy, validation loss and validation accuracy per 5 epochs of our model.It is clearly shown that the accuracy was rising up over time and the loss was degrading.

Number of Epochs	Loss	Accuracy	Val Loss	Val Accuracy
5	0.8950	0.5946	0.5840	0.7481
10	0.6985	0.6458	0.5388	0.7443
15	0.6251	0.6757	0.5464	0.6780
20	0.5924	0.6970	0.5513	0.6780
25	0.5920	0.6927	0.5504	0.6856
30	0.5947	0.6861	0.5540	0.6686
35	0.5857	0.6927	0.5545	0.6648
40	0.5889	0.6965	0.5533	0.6667
45	0.5798	0.7032	0.5530	0.6705
50	0.5848	0.6918	0.5535	0.6686

Table 4.1: Outputs per 5 epochs in CNN Architecture

Before generating the accuracy of our model, we have plotted the accuracy and loss of every epoch to visualize the learning flow of our model. We generated a json file. Also we saved the weight of the model in a HDF5 file in order to utilize it for future studies. We performed these tasks on every architecture we applied on the dataset. Figure 4.5 shows the accuracy and loss our model throughout the training proccess.

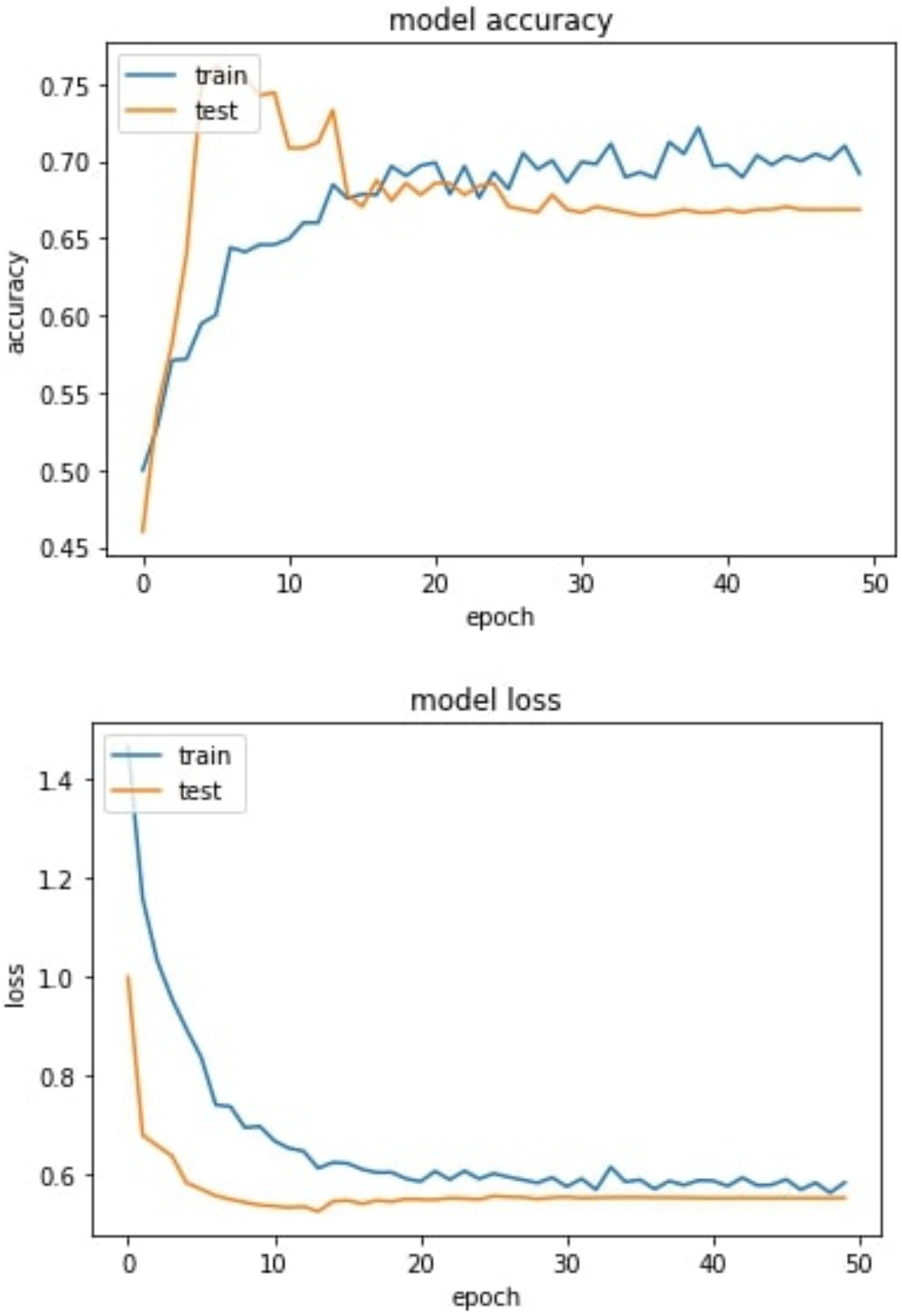

Figure 4.5: CNN Model Accuracy(accuracy vs epochs) and Model Loss(loss vs epochs)

Figure 4.6 shows 10 sample images from the dataset along with their labels.

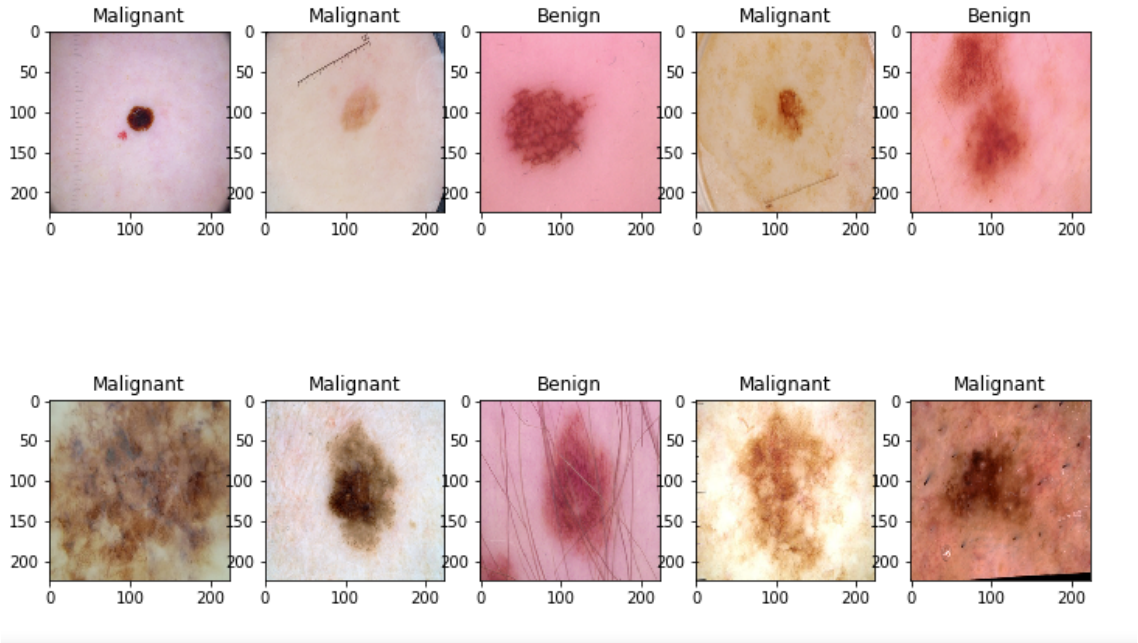

Figure 4.6: Output of original test data

Figure 4.7 shows the prediction of our model on those 10 images.

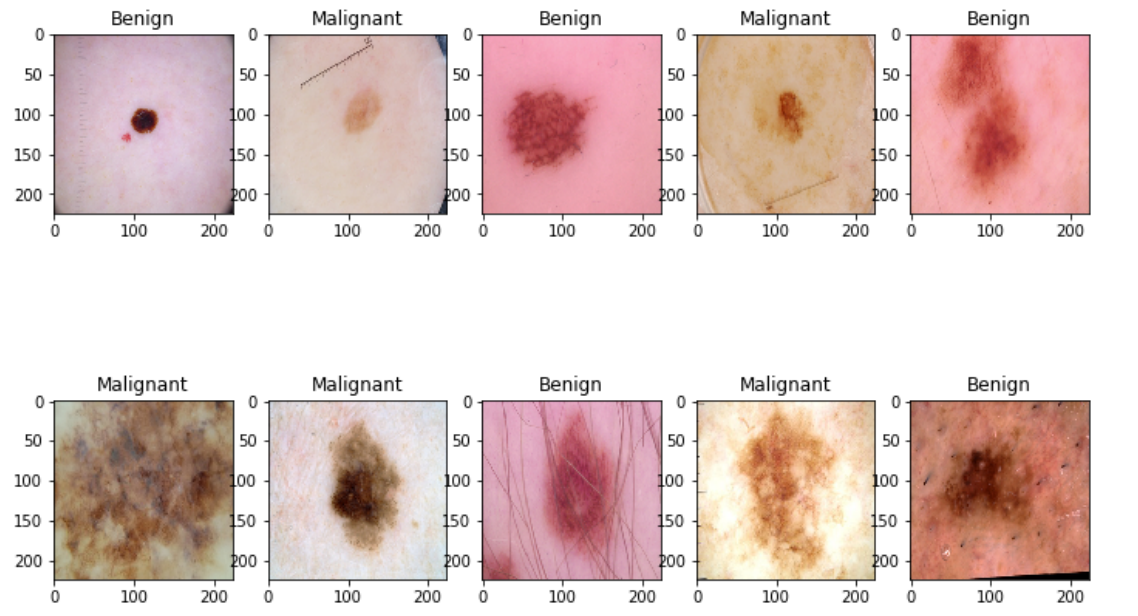

Figure 4.7: CNN model prediction on the same test data

4.2 ResNet

4.2.1 ResNet Concept

ResNet which is a short form of Residual Networks is one the classic neural networks used in Image recognition. In 2012, Krizhevsky proved that this architecture was more accurate than regular hand-crafted feature learning on the ImageNet [21]. ResNet is considered as a deep neural network since it is consists of many hidden layers and thus the layer learns more complex features. It is almost related to other neural networks that have convolution, pooling, activation and fully connected layers stacked one over another. There are some identity connections between the layers. The network learns many features at the end of its layers. But there are some drawbacks when a network goes deeper. Some difficulties may be faced and the accuracy starts saturating and eventually degrades [8]. CIFAR-10 and ImageNet shows more errors as the number of layers are increased. Deep networks are not easier to train because of vanishing gradient problem. As the gradient is back-propagated to earlier layers, the gradient becomes extremely small because of repeated multiplication [18].With the concept of ResNet, the drawbacks of training very deep networks has been resolved. If the gradient is vanished with non-linearity, the gradient can directly flow to the input layer and the network will learn better.
Figure 4.8 shows the residual mapping of ResNet.

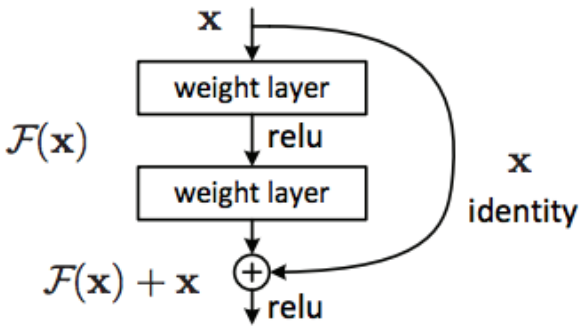

Figure 4.8: ResNet Concept

4.2.2 ResNet50

ResNet-50 is a convolutional neural network that is trained on more than a million images from the ImageNet database [24]. The network is 50 layers deep and can classify images into 1000 categories of object. For this reason, the network can learn rich feature representations for a wide range of images. The network has an image input size of 224x224. New images can be classified using the ResNet-50 model. ResNet-50 works similar to GoogleNet.

4.2.3 Applying ResNet50 in Our Dataset

After we trained our model with basic CNN with KFold, then we used ResNet50 architecture to measure the accuracy of our dataset. We kept all the parameter's

value same as we used before in CNN model. Inside the ResNet50 architecture we changed couple of parameters according to our needs. For example, we set the value of include-top = true. Most of the previous works on this type of sector have used couple of CNN architectures like ResNet50 with setting their weight's value 'imagenet' as they used a pre-trained model for their work. However, we used the value to 'None' as we wanted to train our model from a scratch and check the actual accuracy after training the model with ResNet architecture. Another change that we made was to change the pooling operation to 'pooling' from "MaxPooling" and used a value of 'avg' (average). Means, it will take the average value of a given marix of pixels rather than taking the maximum value as we did in MaxPooling operation. After declaring the model, we compiled it by setting the optimizer as 'Adam' and loss function as 'binary-crossentropy'. Finally, setting the 20% of the train dataset as validation dataset, we fit the model and then at the end we plotted the the graph of accuracy vs epoch and loss vs epoch to see the visual representation of the accuracy and the loss of our model. We got about 82.424% accuracy by using ResNet50 architecture in our dataset. Finally we created a json file and a h5 file to save the model and then deleted the model for training our dataset with a new architecture.
The table below shows the loss,accuracy, validation loss and validation accuracy per 5 epochs of ResNet50.It is clearly shown that the accuracy was rising up over time and the loss was degrading.

Number of Epochs	Loss	Accuracy	Val Loss	Val Accuracy
5	0.4000	0.7918	0.6062	0.6004
10	0.3300	0.8412	0.4712	0.7765
15	0.2700	0.8838	0.3907	0.7936
20	0.2273	0.9075	0.4228	0.8220
25	0.1828	0.9227	0.4171	0.8239
30	0.1292	0.9540	0.4441	0.8314
35	0.1065	0.9565	0.4802	0.8239
40	0.0878	0.9687	0.4681	0.8371
45	0.0731	0.9725	0.4780	0.8352
50	0.0665	0.9782	0.4991	0.8258

Table 4.2: Outputs per 5 epochs in ResNet50 Architecture

Figure 4.9 shows the accuracy and loss of ResNet50 throughout the training proccess.

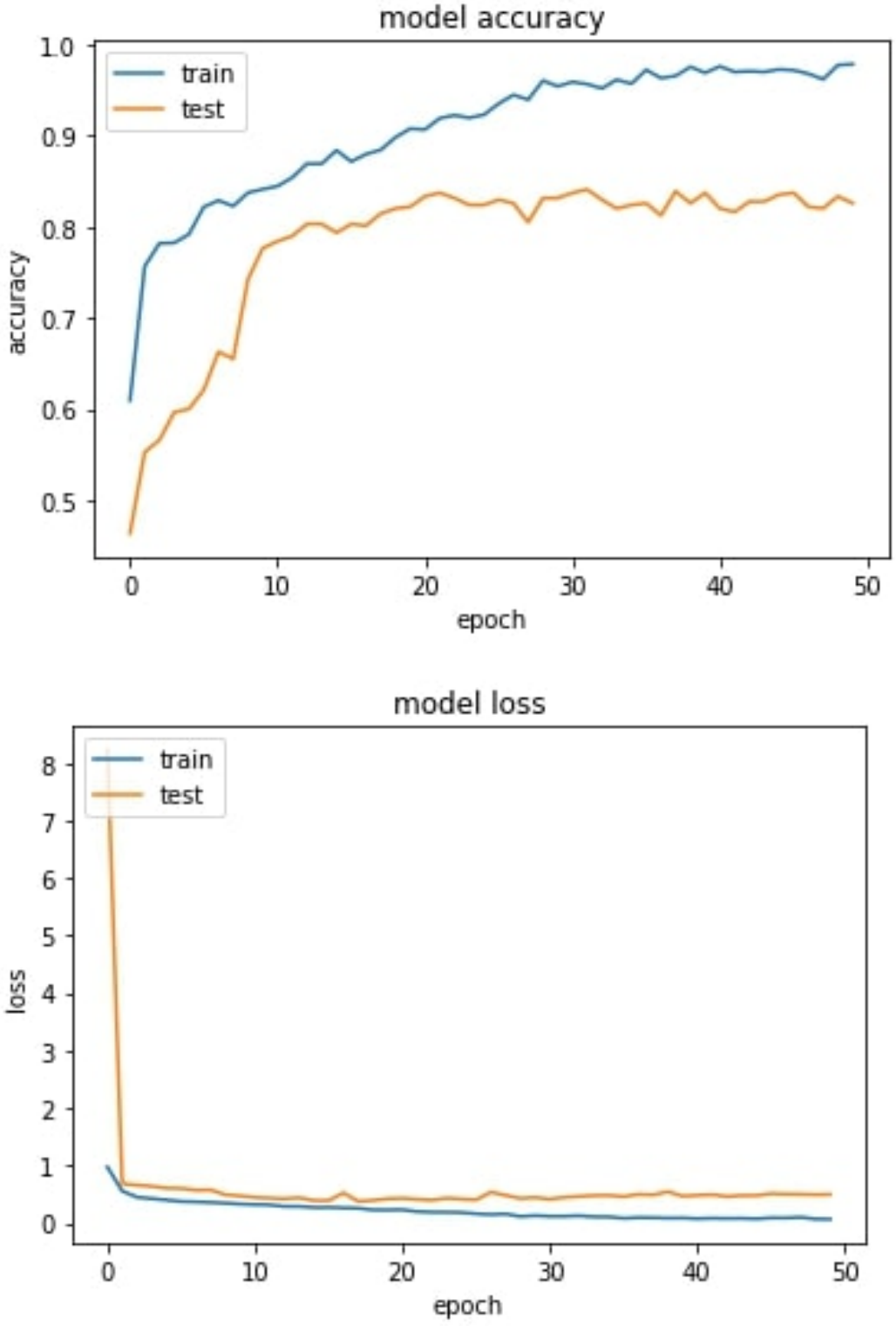

Figure 4.9: ResNet50 Model Accuracy(accuracy vs epochs) and Model Loss(loss vs epochs)

Figure 4.10 shows 10 sample images from the dataset along with their labels.

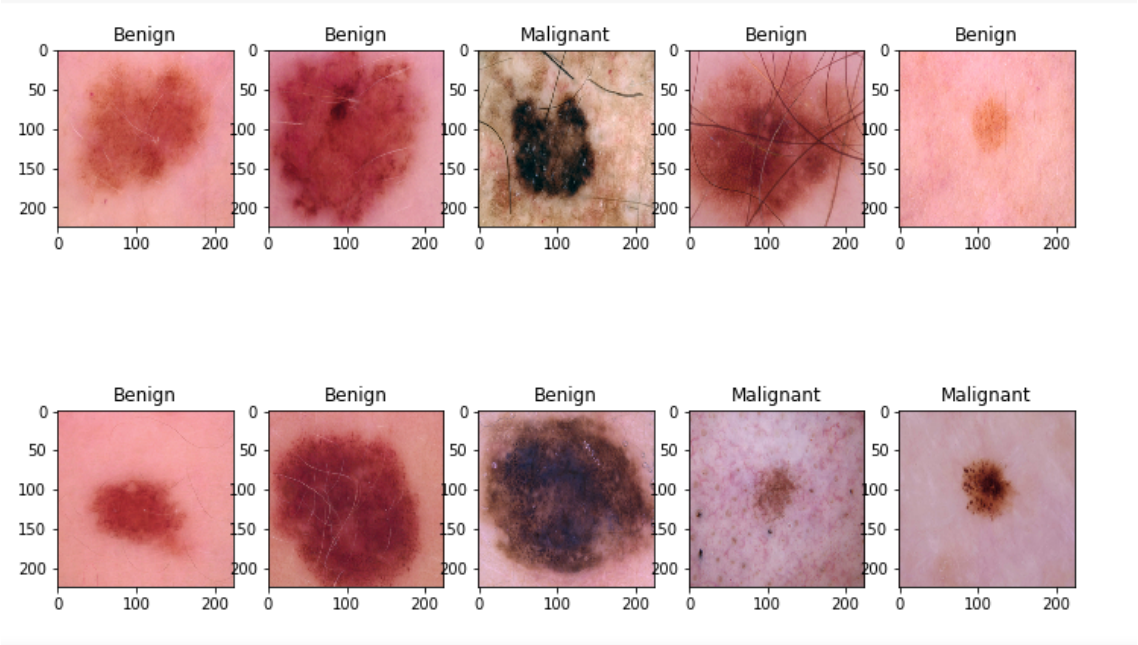

Figure 4.10: Output of original test data

Figure 4.11 shows the prediction of ResNet50 on those 10 images.

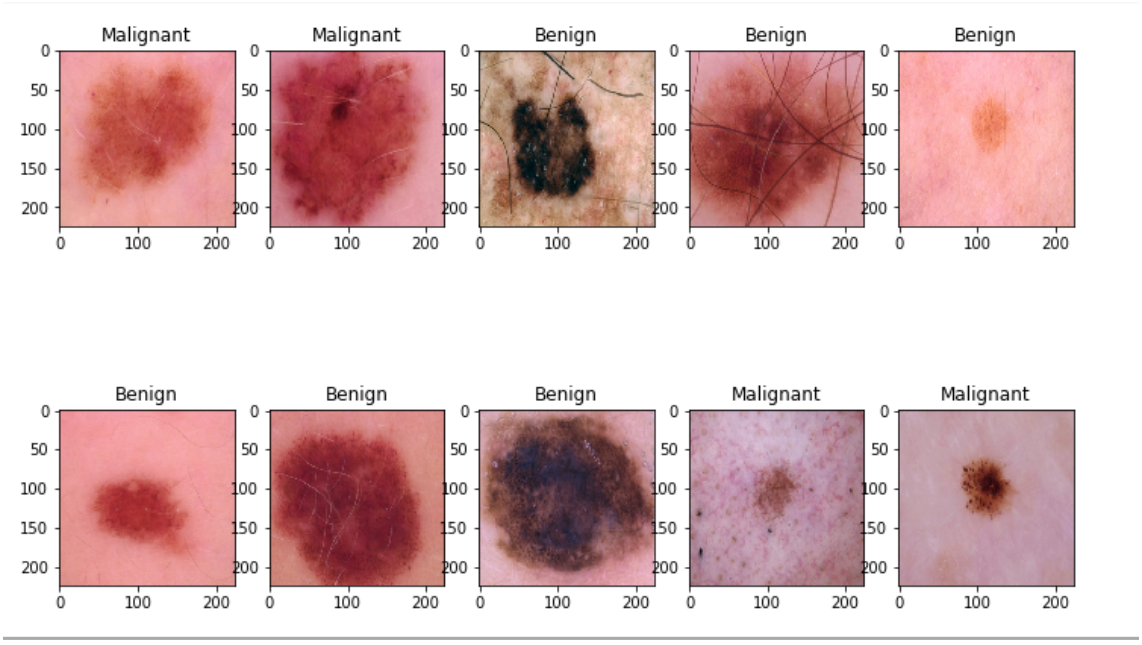

Figure 4.11: ResNet50 prdeiction on the same test data

4.3 VGG

4.3.1 VGG Concept

The word VGG is the abbreviation of Oxford Visual Geometry Group. Researchers from that group came up with a model in ILSVRC and later cnn done by this model won the image classification task. VGG is capable of classifying objects in photographs. Besides the model weights are freely available and can be loaded and used in any other models and applications. Researchers made two type of VGG which are VGG16 and VGG19 where number stands for layers in that architecture [2] [17]

4.3.2 VGG16

The VGG16 architecture has twelve convolutional layers. Some of those layers are followed by a maximum pooling layer. Then there are four fully connected layers and a 1000-way softmax classifier at end. Those layers are broadly described below.

- **First and Second Layers:** Feature map 64. Filter size 3x3. Stride 2. Input image dimensions 224x224x64. Resulting image dimensions 112x112x64.
- **Third and Fourth Layer:** Feature map 128 and rest are same. Resulting image dimensions 56x56x128.
- **Fifth and Sixth Layers:** Feature map 256 and rest are same. Resulting image dimensions 56x56x256.
- **Seventh to Twelfth Layer:** 2 set of 512 features map with a max pooling in between and rest are same. Resulting image dimensions 7x7x512
- **Thirteenth Layer:** A fully connected layer with 25088 feature maps each of size 11.
- **Fourteenth and Fifteenth Layers:** 2 fully connected layers having 4096 units.
- **Output Layer:** A layer with softmax and 1000 possible values.

Figure 4.12 shows the architecture of VGG16.

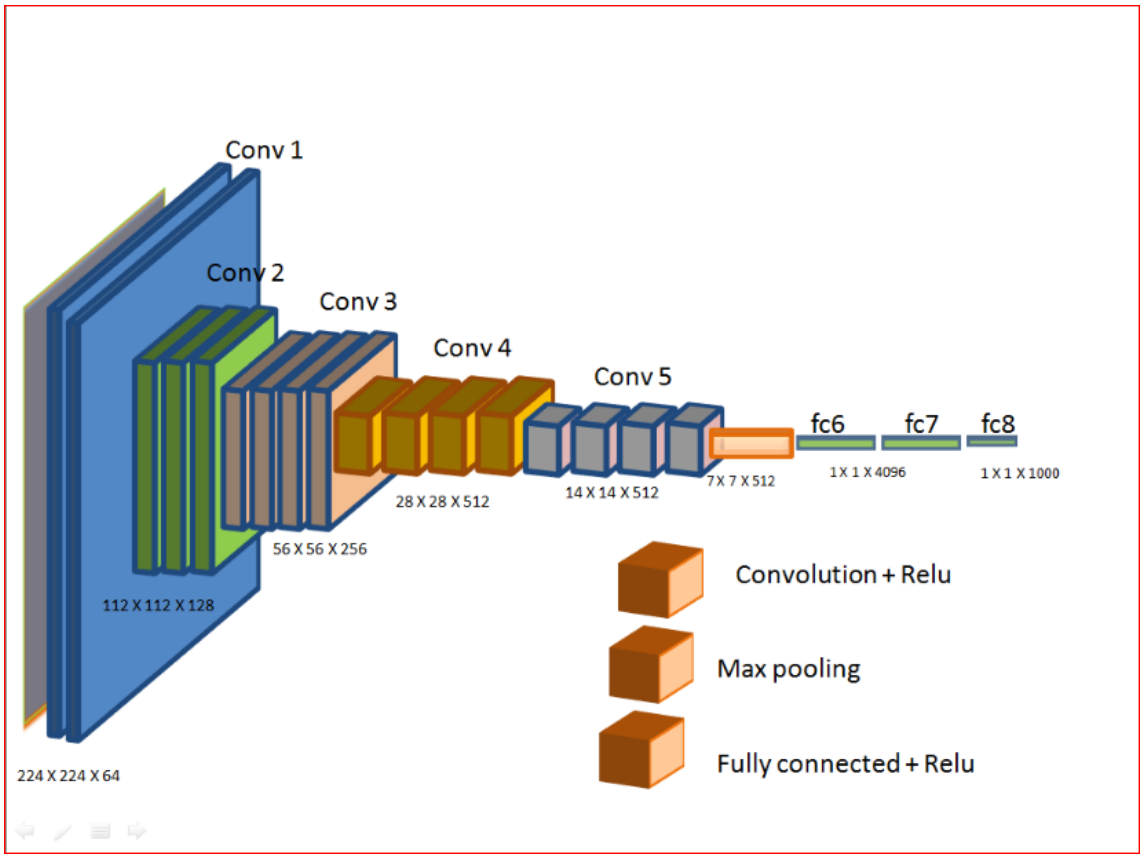

Figure 4.12: VGG16

4.3.3 Applying VGG16 in Our Dataset

After applying ResNet50, we trained our model with VGG16 architecture to measure the accuracy of our dataset. We pass the same parameters in the model and started training it. After training our model with VGG16, we got about 84.242% accuracy from our dataset.
The table above shows the loss,accuracy, validation loss and validation accuracy per 5 epochs of VGG16.It is clearly shown that the accuracy was rising up over time and the loss was degrading.
Figure 4.13 shows the accuracy and loss of VGG16 throughout the training proccess.

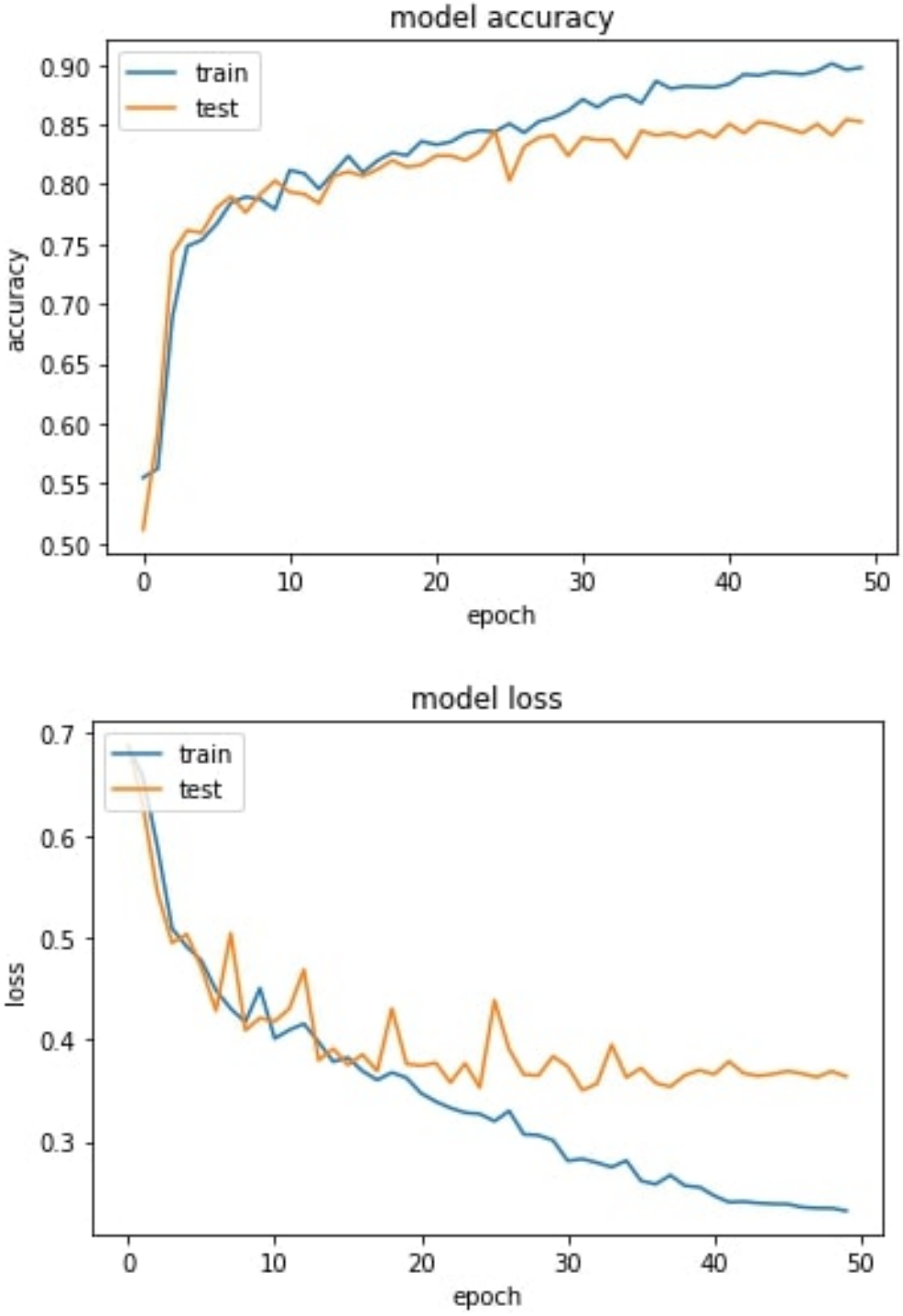

Figure 4.13: VGG16 Model Accuracy(accuracy vs epochs) and Model Loss(loss vs epochs)

Number of Epochs	Loss	Accuracy	Val Loss	Val Accuracy
5	0.4910	0.7534	0.5031	0.7595
10	0.4502	0.7790	0.4211	0.8030
15	0.3783	0.8236	0.3910	0.8106
20	0.3624	0.8359	0.3760	0.8163
25	0.3270	0.8440	0.3528	0.8447
30	0.3011	0.8615	0.3835	0.8239
35	0.2812	0.8677	0.3628	0.8447
40	0.2552	0.8810	0.3699	0.8390
45	0.2388	0.8928	0.3660	0.8466
50	0.2321	0.8976	0.3637	0.8523

Table 4.3: Outputs per 5 epochs in VGG16 Architecture

Figure 4.14 shows 10 sample images from the dataset along with their labels.

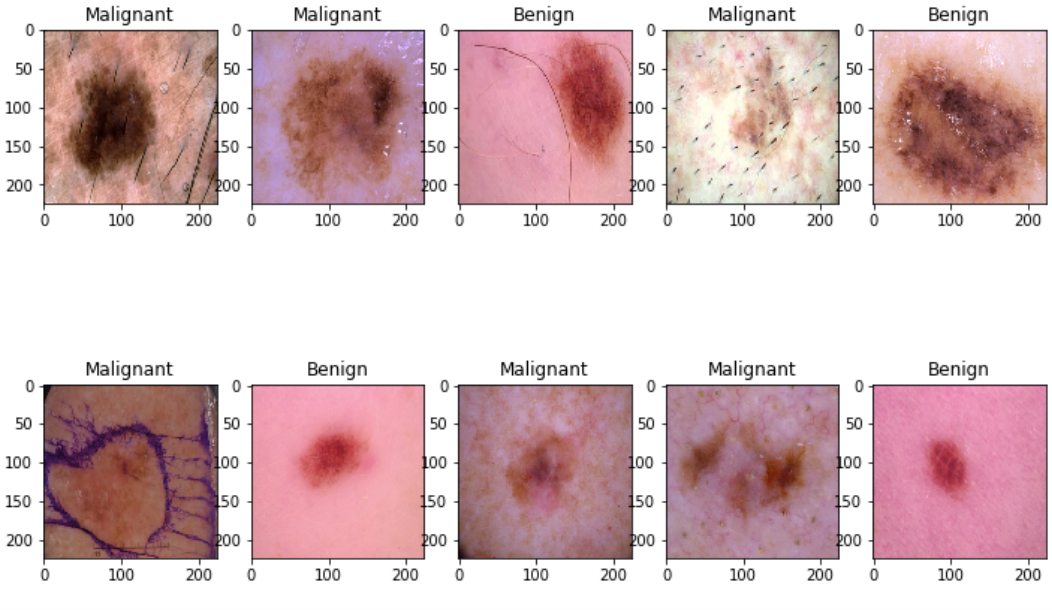

Figure 4.14: Output of original test data

Figure 4.15 shows the prediction of VGG16 on those 10 images.

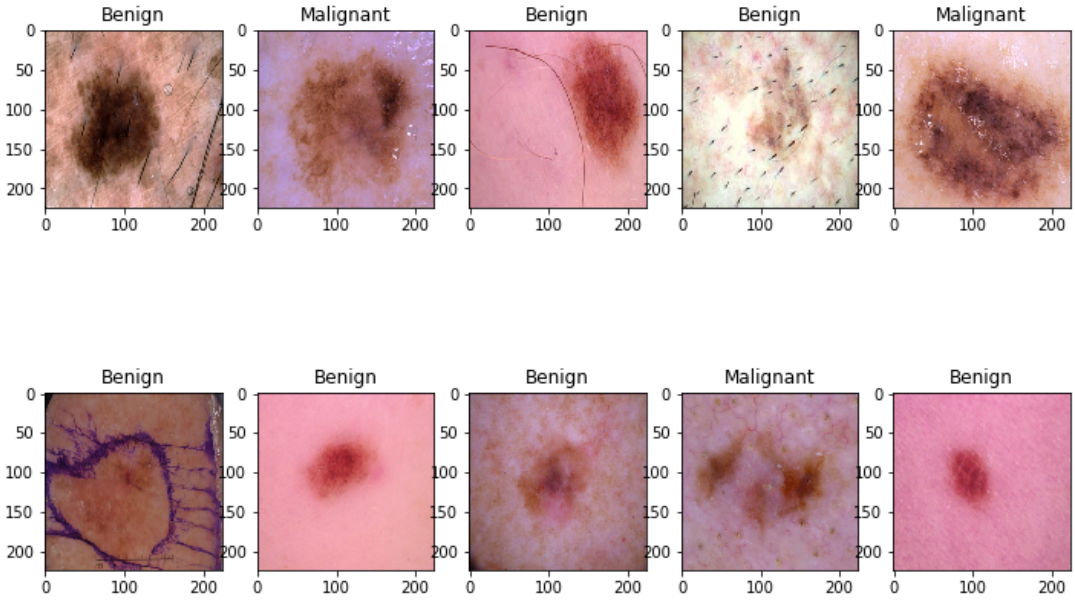

Figure 4.15: VGG16 prediction on the same test data

4.3.4 VGG19

VGG19 has the same architecture as VGG16, the only difference is that VGG19 has three additional layers with very small (33) convolution filters.
Figure 4.16 shows the architecture of VGG19.

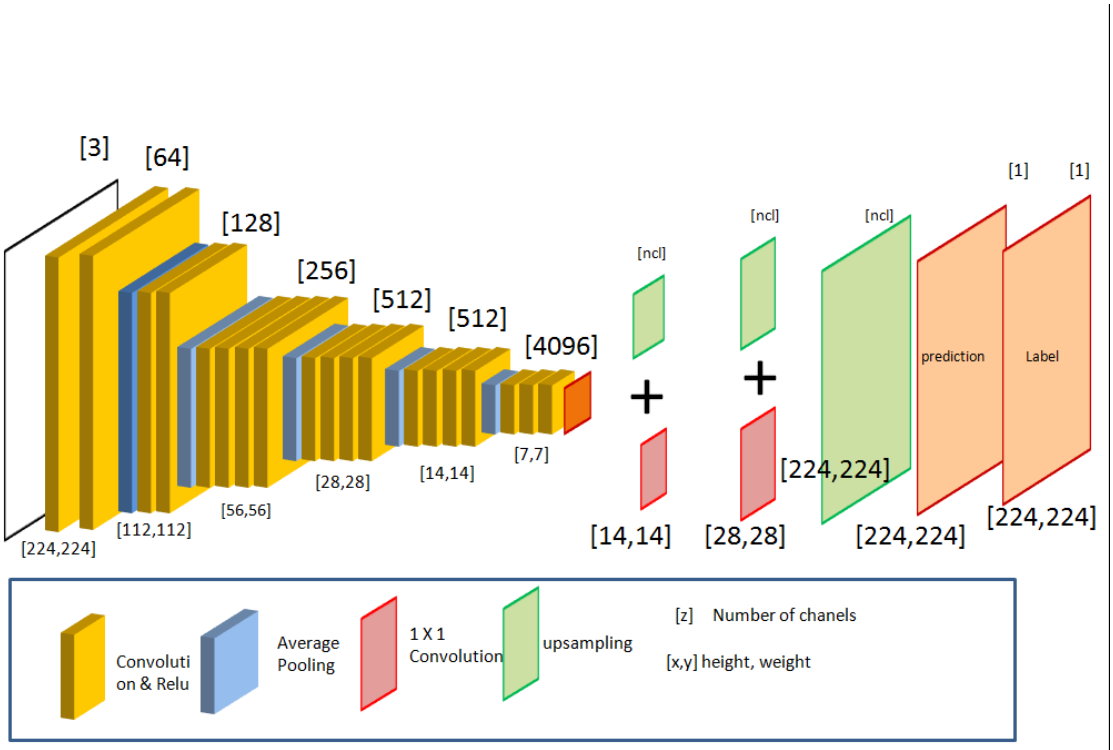

Figure 4.16: VGG19

4.3.5 Applying VGG19 in Our Dataset

We trained our model with VGG19 architecture to measure the accuracy of our dataset by passing the same parameters to the model. After training our model with VGG19, we got about 84.545% accuracy from our dataset. The table below shows loss, accuracy and the validation loss and accuracy per 5 epochs of VGG19.

Number of Epochs	Loss	Accuracy	Val Loss	Val Accuracy
5	0.5042	0.7549	0.6090	0.6932
10	0.4403	0.7876	0.4094	0.8030
15	0.3890	0.8004	0.3868	0.8068
20	0.3594	0.8255	0.3983	0.8087
25	0.3311	0.8421	0.4045	0.8239
30	0.3182	0.8464	0.3724	0.8352
35	0.2696	0.8725	0.3789	0.8409
40	0.2628	0.8796	0.3941	0.8277
45	0.2227	0.8971	0.4001	0.8277
50	0.2009	0.9161	0.3781	0.8447

Table 4.4: Outputs per 5 epochs in VGG19 Architecture

Figure 4.17 shows accuracy and loss of VGG19 throughout the training proccess.
Figure 4.18 shows 10 sample images from the dataset along with their labels.
Figure 4.19 shows the prediction of VGG19 on those 10 images.

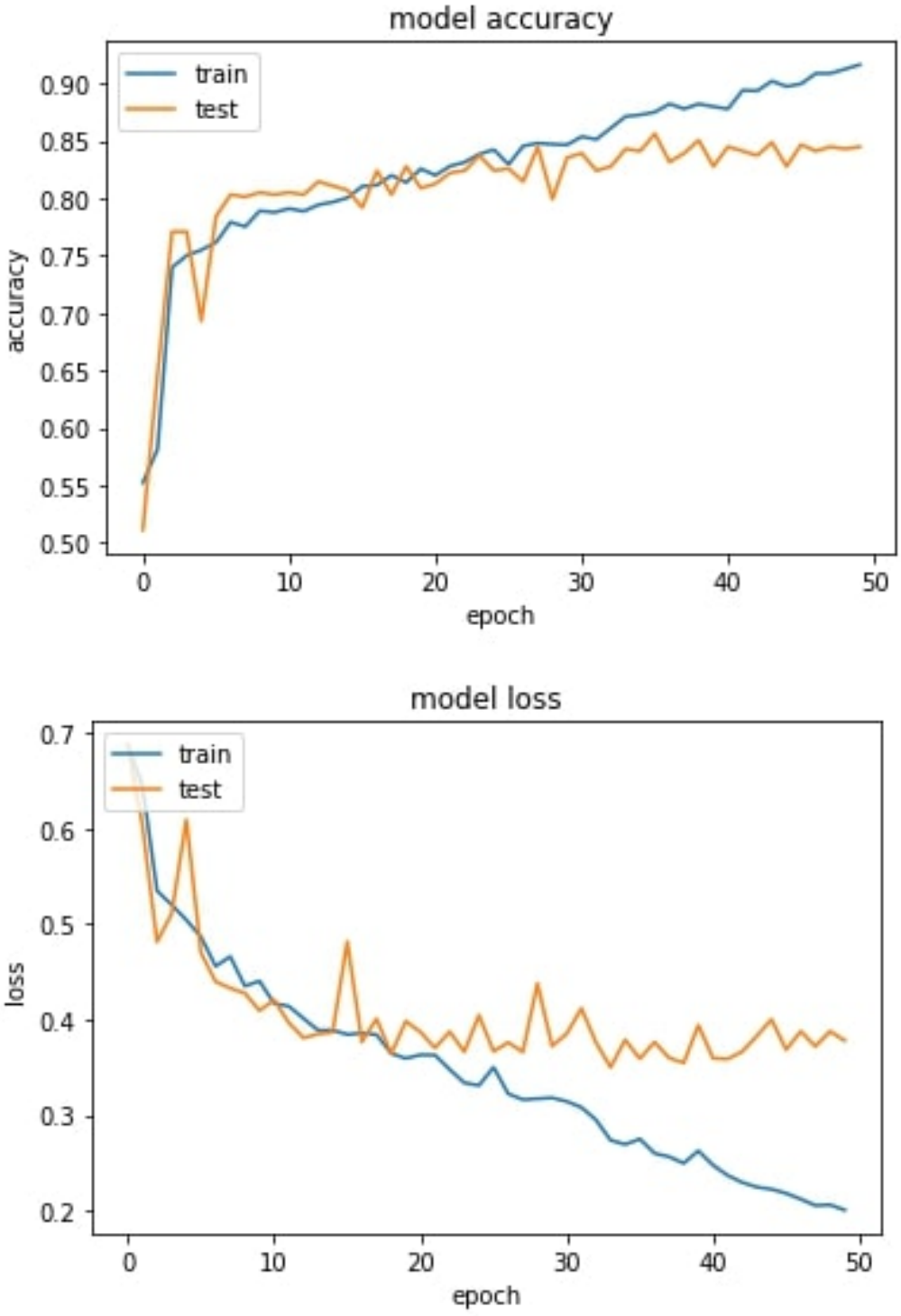

Figure 4.17: VGG19 Model Accuracy(accuracy vs epochs) Model Loss(loss vs epochs)

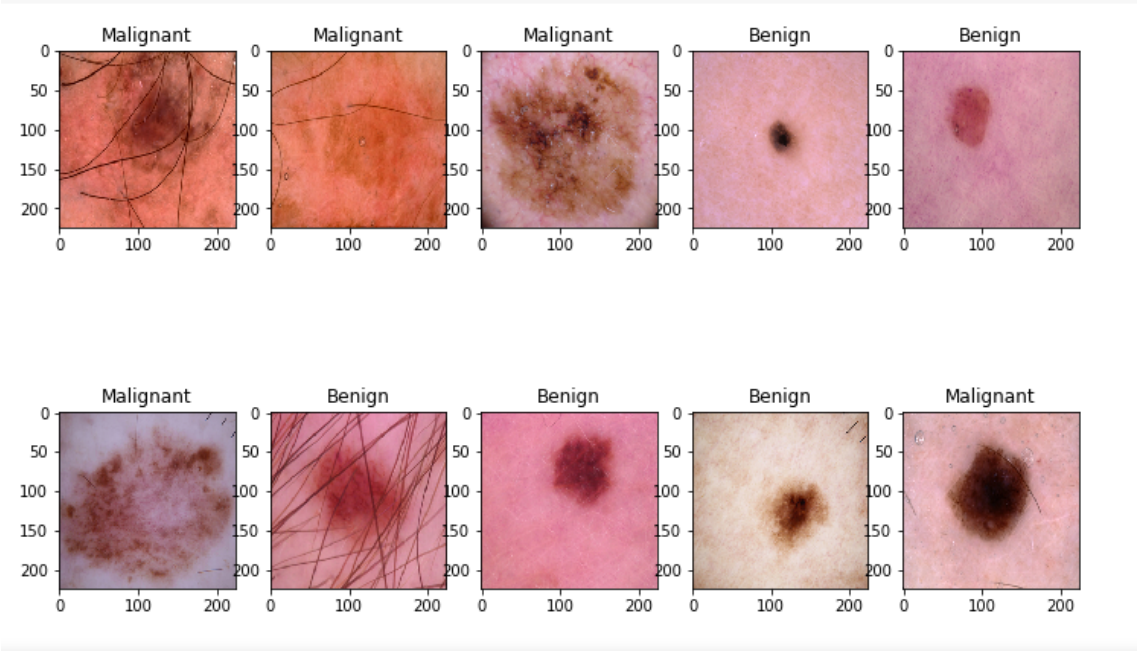

Figure 4.18: Output of original test data

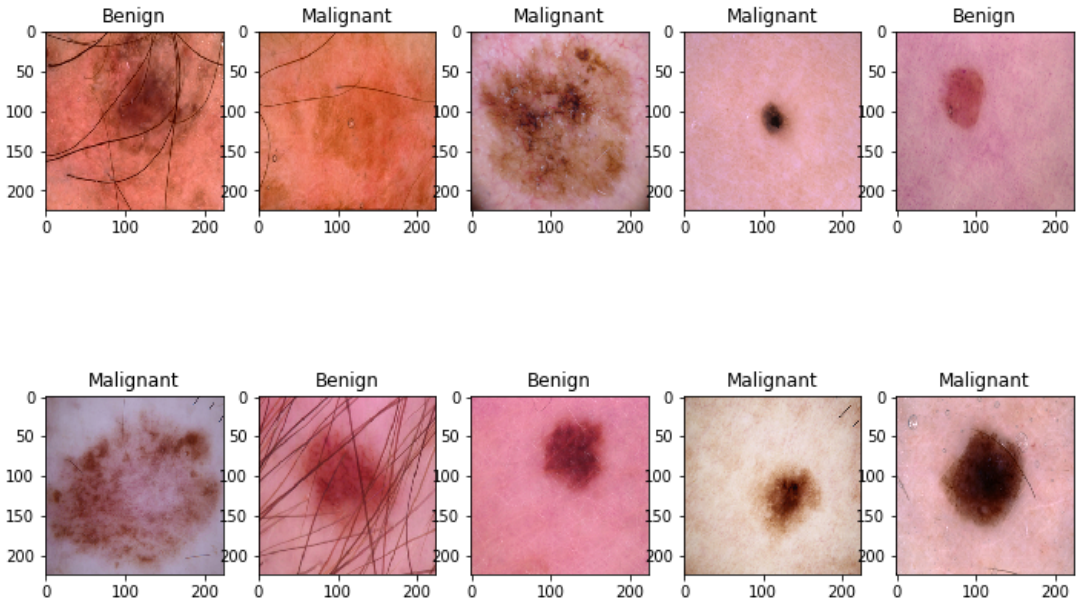

Figure 4.19: VGG19 prediction on the same test data

4.4 Inception

4.4.1 Inception Concept

In Convolutional Neural Networks it is important to choose the right layer to apply, among the most common options are applying filters or max-pooling. All we need is to find the optimal local construction and to repeat it spatially[16]. For this reason, it is important to know features from previous layers. So when subsequent layers are being created in any deep learning model, gathering information from previous layers should be a concern. This is where the inception layer comes to the use. It allows the internal layers to pick and choose relevant filter sizes to learn the required information. So even if the size of the face in the image does not match, the layer works accordingly to recognize the face. It would possibly take a higher filter size for the first image, while the second image would take a smaller one[15]. Thus inception pushes performance in terms of speed and accuracy. The rapid development of inception leads to implementing several versions of it. The popular versions are Inception is Inception v1, Inception v2, Inception v3, Inception v4 and Inception-ResNet[13].
Figure 4.20 shows the naive version of inception module and the module after dimension reduction.

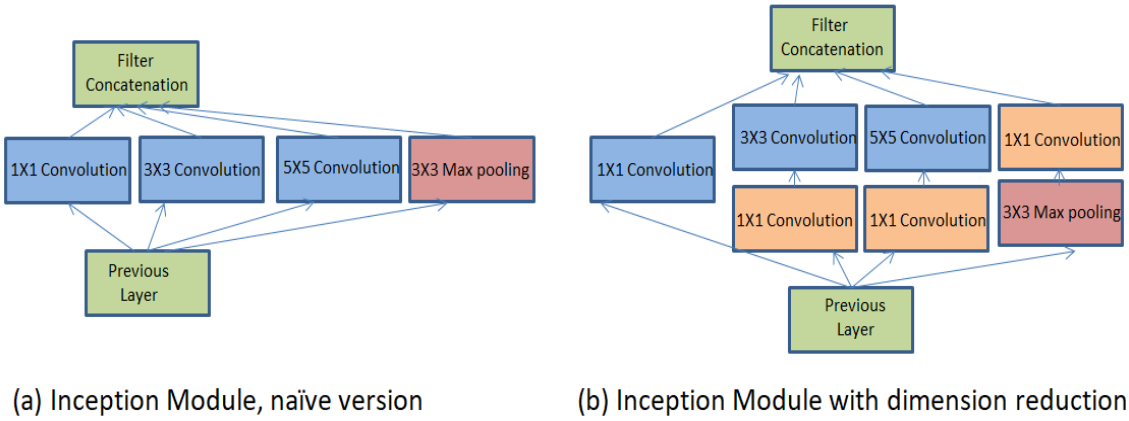

Figure 4.20: Inception

4.4.2 InceptionV3

Inception v3 is the upgraded version of Inception v2. Since the authors found out applying auxiliary classifiers does not contribute up to expectation until the end of training process, when the accuracy reaches a saturation state. So they came up with an idea of developing Inception v3 without changing the modules of Inception v2. They introduced new features to the v3 along with the features from v2. Some

of those features are RMSProp Optimizer, 7x filters, BatchNorm in the Auzxiliary Classifiers and Label Smoothing etc.

4.4.3 Applying InceptionV3 in Our Dataset

After applying VGG19, we trained our model with InceptionV3 architecture to measure the accuracy of our dataset. We did all the process exactly as we did in ResNet50 architecture by keeping all other parameter's value exactly the same. After training our model with InceptionV3, we got about 83.787% accuracy from our dataset.
The table below shows the loss,accuracy, validation loss and validation accuracy per 5 epochs of InceptionV3.It is clearly shown that the accuracy was rising up over time and the loss was degrading.

Number of Epochs	Loss	Accuracy	Val Loss	Val Accuracy
5	0.3553	0.8374	0.7788	0.5114
10	0.3172	0.8525	0.8059	0.5758
15	0.2770	0.8696	0.6249	0.7045
20	0.2477	0.8914	0.4638	0.7652
25	0.2113	0.9161	0.3570	0.8409
30	0.1890	0.9232	0.3774	0.8409
35	0.1566	0.9445	0.3981	0.8220
40	0.1559	0.9355	0.3889	0.8333
45	0.1316	0.9502	0.3910	0.8201
50	0.1253	0.9564	0.3890	0.8277

Table 4.5: Outputs per 5 epochs in InceptionV3 Architecture

Figure 4.21 shows the accuracy and loss of InceptionV3 throughout the training process.

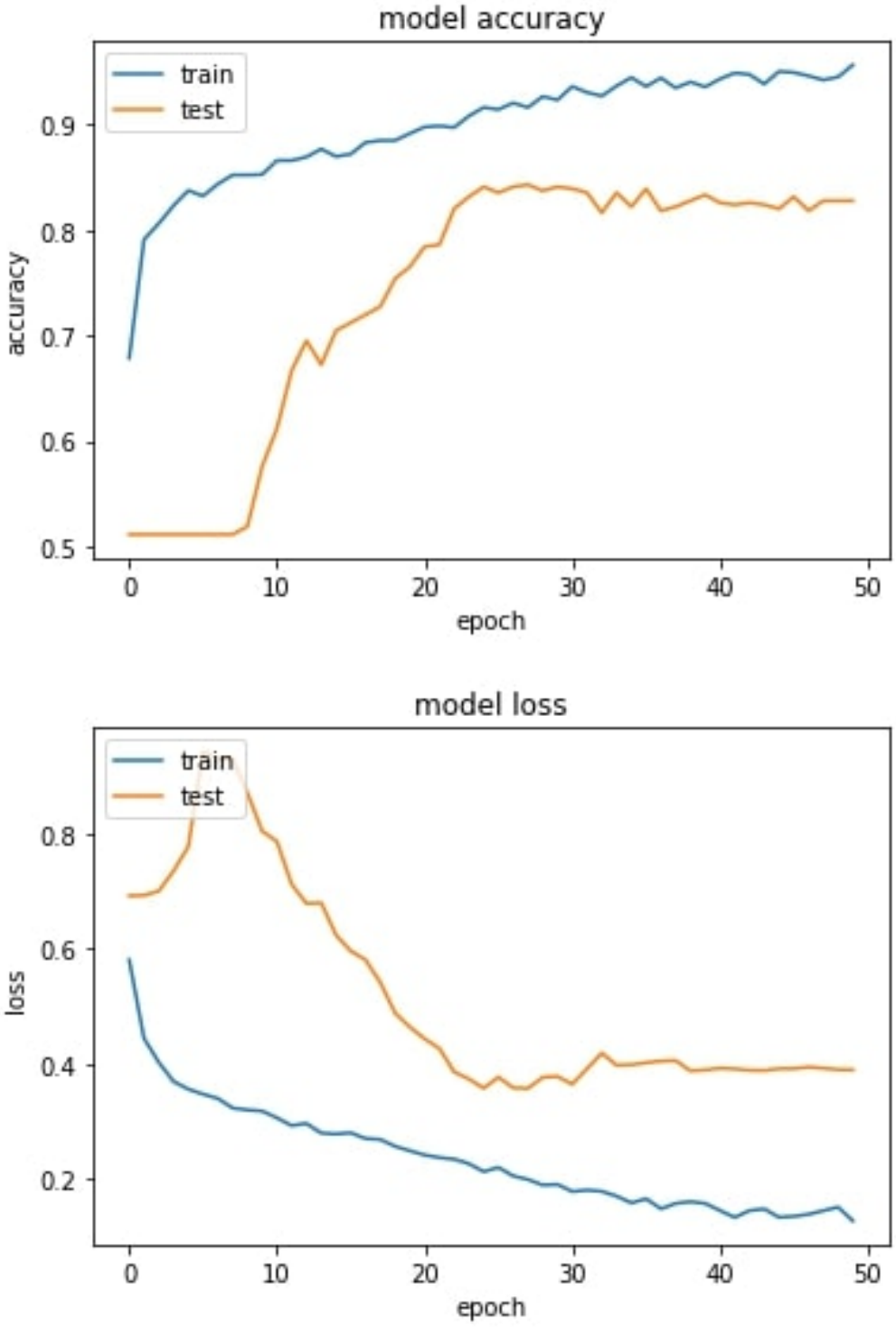

Figure 4.21: InceptionV3 Model Accuracy(accuracy vs epochs) Model Loss(loss vs epochs)

Figure 4.22 shows 10 sample images from the dataset along with their labels.

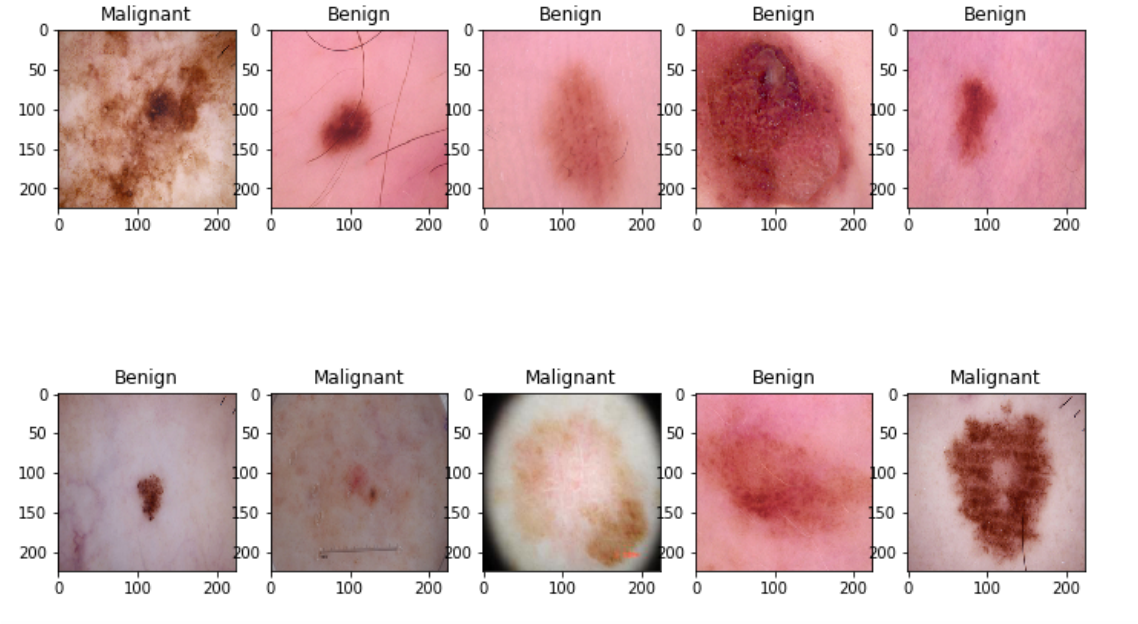

Figure 4.22: Output of original test data

Figure 4.23 shows the prediction of InceptionV3 on those images.

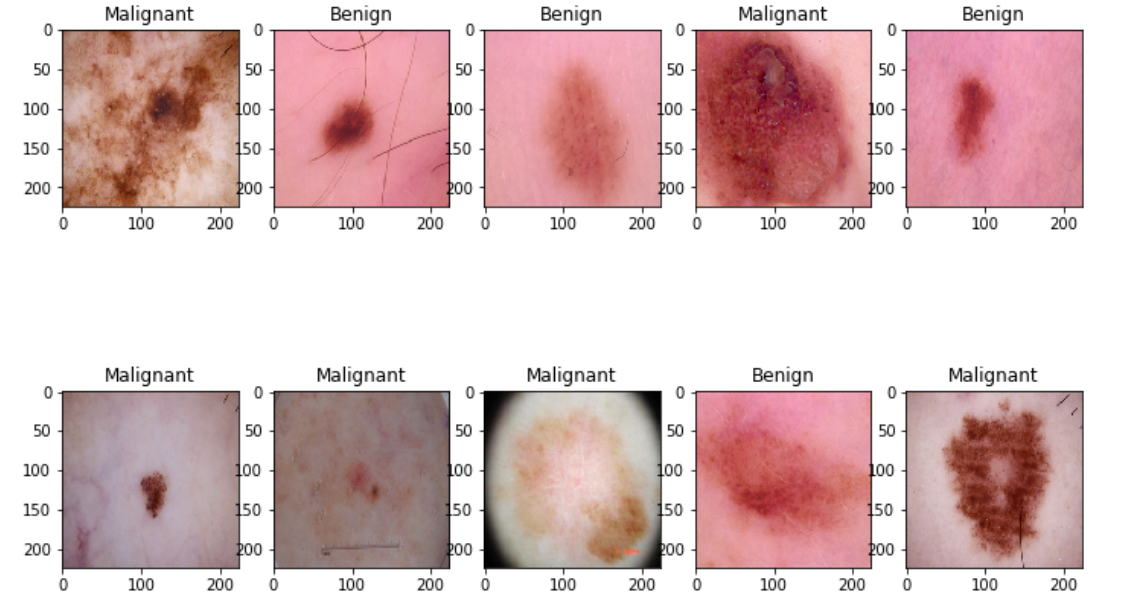

Figure 4.23: InceptionV3 prediction on the same test data

4.5 Xception

4.5.1 Xception Concept

The name Xception stands for Extreme version of inception with depthwise separable convolution. Depthwise convolution means the channel-wise nxn spatial convolution. If a layer has five channels then there will be 5 nxn spatial convolution. Xception is considered better than Inception v3 on various datasets. There are two types of depthwise separable convolution, one is original depthwise separable convolution and modified depthwise separable convolution. Original depthwise separable convolution is the depthwise convolution followed by a pointwise convolution. Pointwise convolution is the 1x1 convolution to change the dimension. Depthwise separable convolution does not perform convolution across all its channels like conventional approach. For this reason, this is a convenient approach since the runtime gets optimized and the model shows better accuracy also the model gets lighter. On the other hand, modified depthwise separable convolution is the pointwise convolution followed by depthwise convolution. This modification is inspired by 1x1 convolution done by Inception v3 before performing nxn spatial convolution. Besides, to show the mean accuracy prediction when multiple objects appears in a single image, xception is the most fruitful approach[23].
Figure 4.24 shows the orginal depthwise separable convolution in Xception.

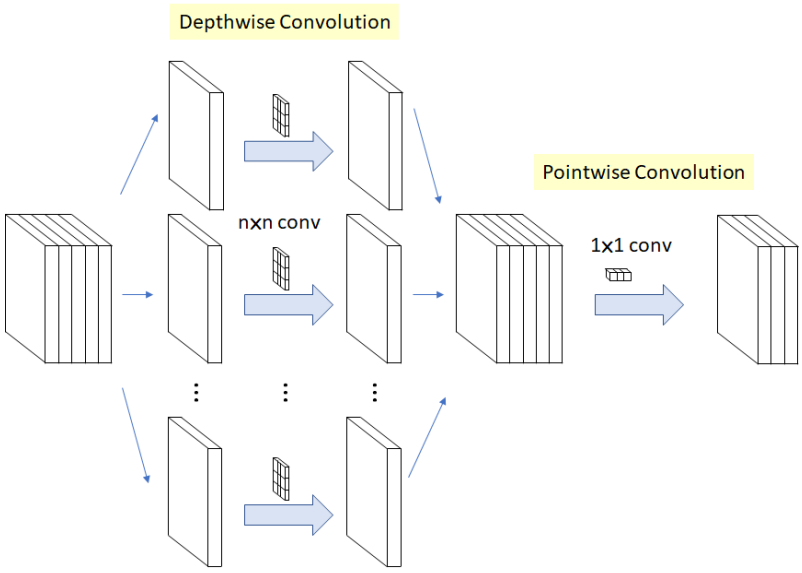

Figure 4.24: Original Xception

Figure 4.25 shows the modified depthwise separable convolution in xception.

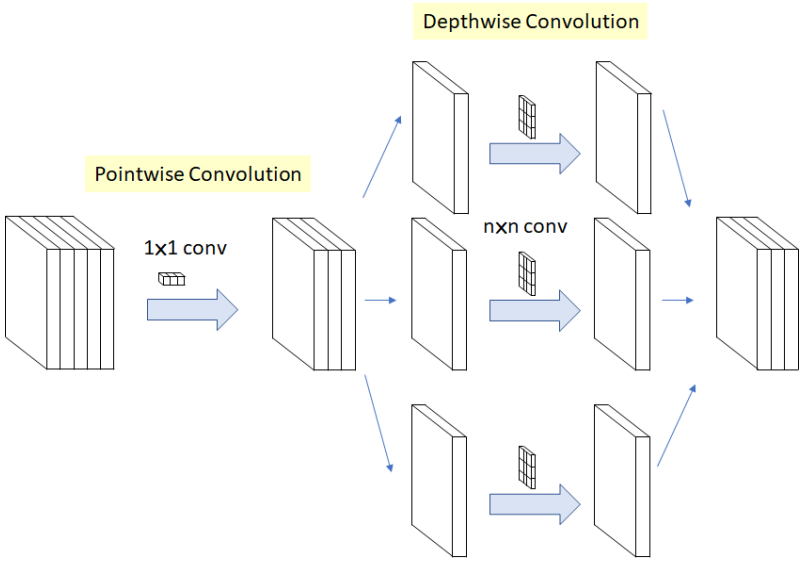

Figure 4.25: Modified Xeption

4.5.2 Applying Xception in Our Dataset

After applying VGG19, we trained our model with Xception architecture to measure the accuracy of our dataset. We did all the process exactly as we did with other architectures by keeping all other parameter's value exactly the same. After training our model with Xception, we got about 85.303% accuracy from our dataset.
The table below shows the loss,accuracy, validation loss and validation accuracy per 5 epochs of Xception.It is clearly shown that the accuracy was rising up over time and the loss was degrading.

Number of Epochs	Loss	Accuracy	Val Loss	Val Accuracy
5	0.3301	0.8492	0.6911	0.5360
10	0.2418	0.9047	0.6942	0.5360
15	0.1891	0.9350	0.7985	0.5360
20	0.1702	0.9426	0.9698	0.5379
25	0.1384	0.9606	0.4709	0.7803
30	0.1246	0.9625	0.3569	0.8220
35	0.1020	0.9768	0.3543	0.8295
40	0.0809	0.9829	0.3598	0.8371
45	0.0747	0.9853	0.3662	0.8295
50	0.0678	0.9867	0.3709	0.8314

Table 4.6: Outputs per 5 epochs in Xception Architecture

Figure 4.26 shows the accuracy and loss of Xception throughout the training process.

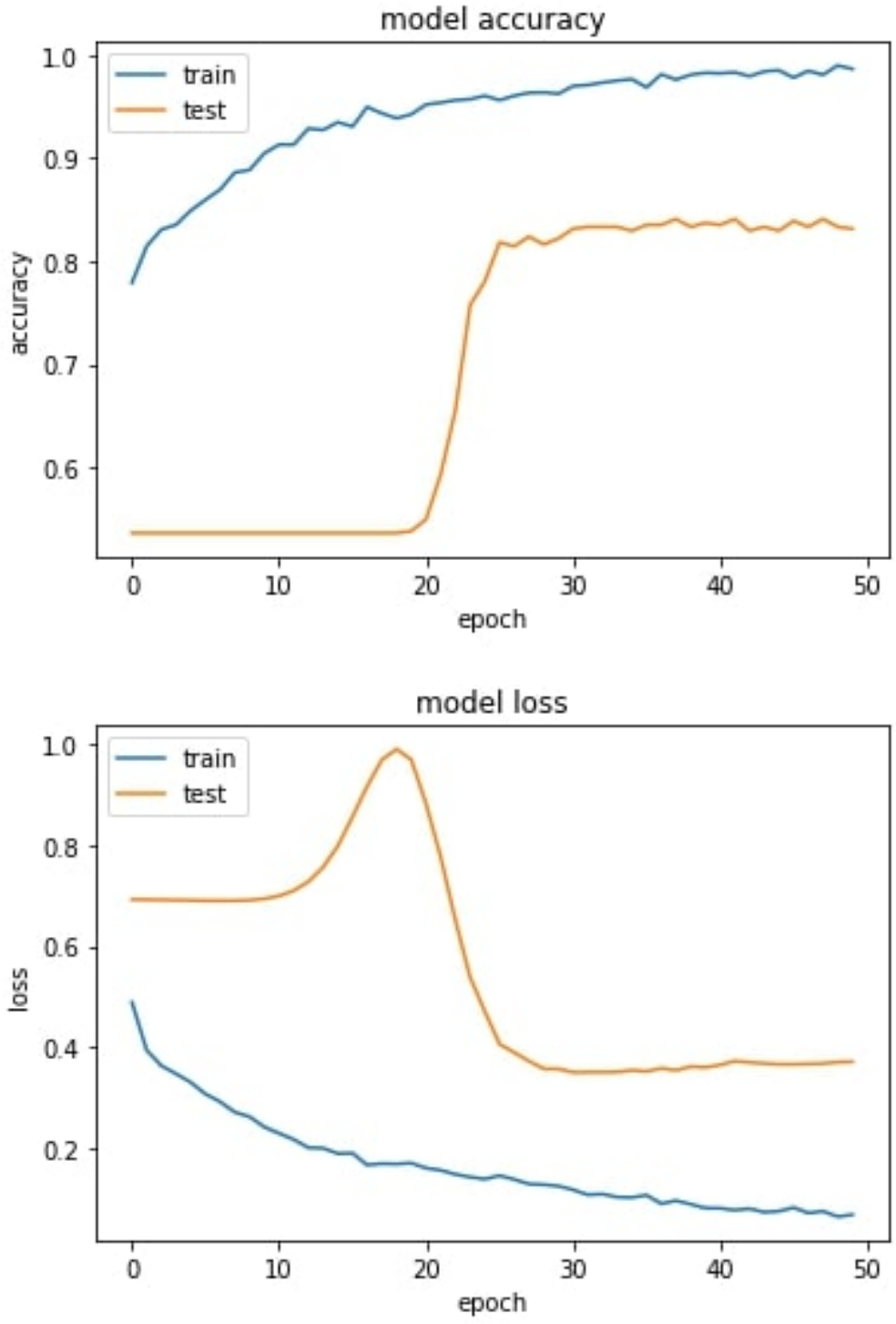

Figure 4.26: Xception Model Accuracy(accuracy vs epochs) Model Loss(loss vs epochs)

Figure 4.27 shows 10 sample images from the dataset along with their labels.

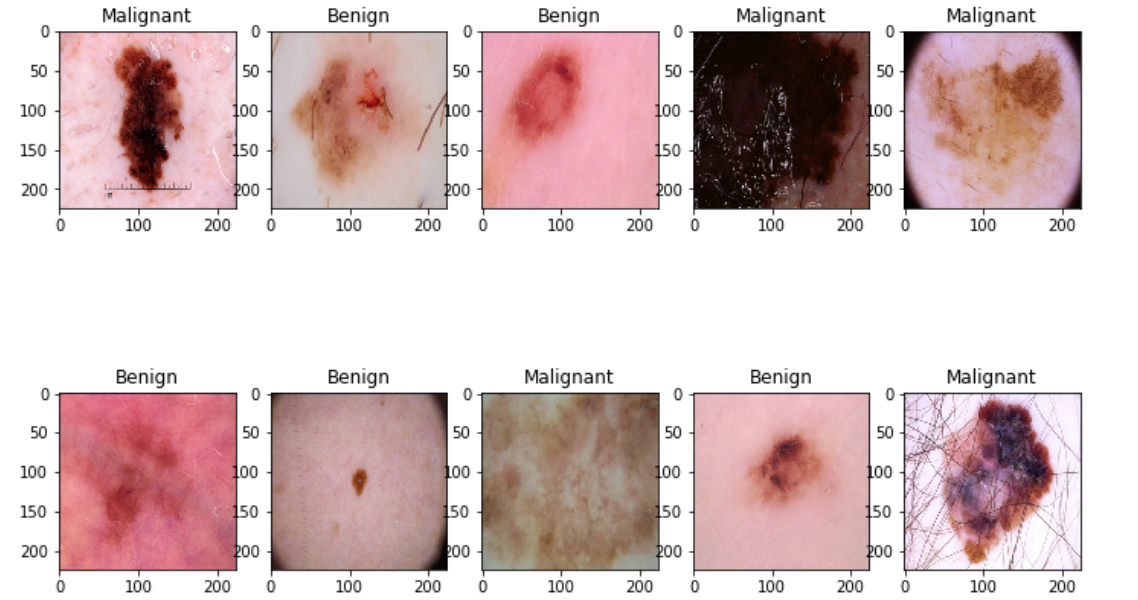

Figure 4.27: Output of original test data

Figure 4.28 shows the prediction of Xception on those images.

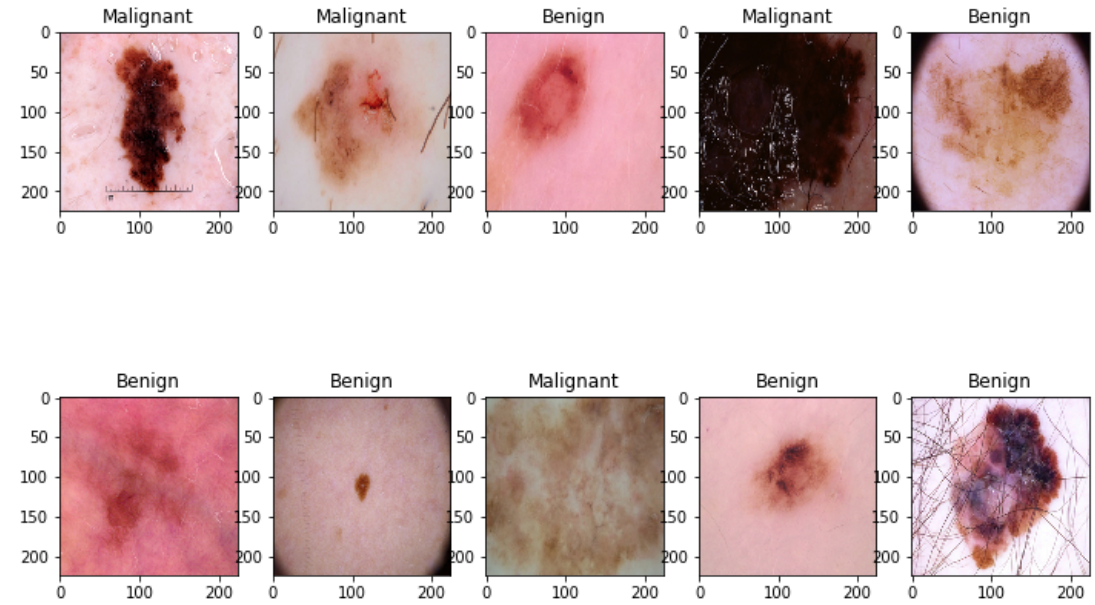

Figure 4.28: Xception prediction on the same test data

4.6 MobileNet

4.6.1 MobileNet Concept

MobileNet is another depthwise separable convolutional approach with smaller model size and complexity. There are two parameters in MobileNet, width multiplier and resolution multiplier where width multiplier is used for thinner models and resolution multiplier is used in order to reduce representation. Width multiplier controls the input width of a layer and resolution multiplier controls the input image resolution. MobileNet only got 1% loss in accuracy, where the parameters are reduced enormously [14].
Figure 4.29 shows the architecture of MobileNet.

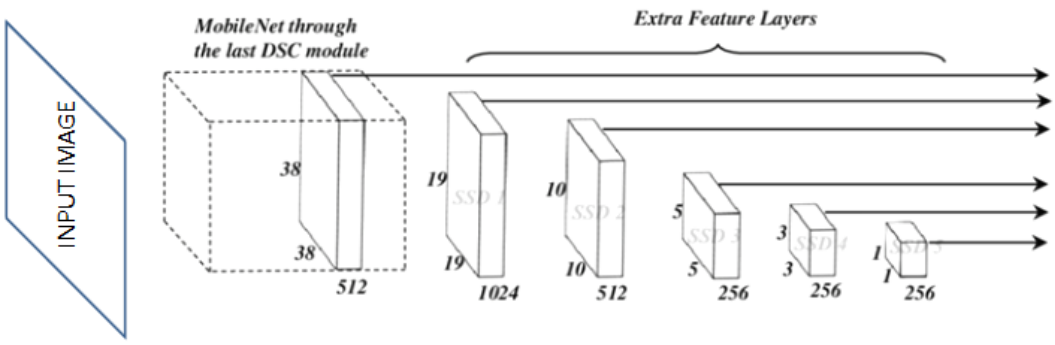

Figure 4.29: MobileNet

4.6.2 MobileNetV2

MobileNetV2 also uses depthwise separable convolutions with three convolutional layers now in a single block. There is a new 1x1 convolution layer which expands the number of channels before it goes into depthwise convolution. The last two layers are same as MobileNet V1. But the pointwise layer works just opposite of v1, it reduces the number of channels. For example, if 24 channels are passed into a block, expansion layer converts it into 24x6 = 144 channels. The depthwise convolution is performed to those channels. After that the projection layer reduces those channels back to 24.[9]. Another new thing about mobilenet v2 is the residual connection between building blocks. These connections work just like same as ResNet's. In MobileNetV2 the gadient flows smoothly throughout the network. Rest are same as

MobileNet v1.
Figure 4.30 shows the architecture of MobileNetV2

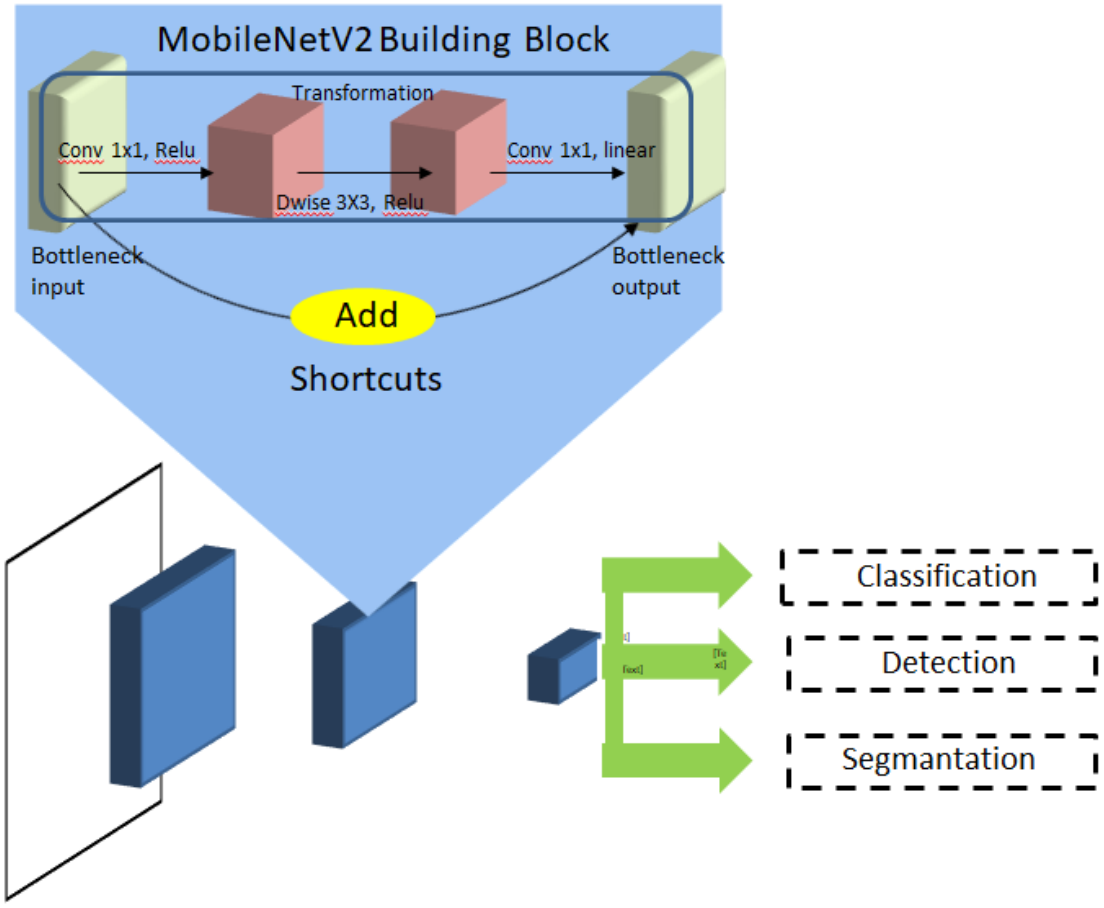

Figure 4.30: MobileNetV2

4.6.3 Applying MobileNetV2 in Our Dataset

After applying Xception, we trained our model with MobileNetV2 architecture to measure the accuracy of our dataset. We introduced a new variable named 'alpha' which was needed to compile the MobileNet architecture. Alpha controls the width of the network.This is also known as width multiplier. Value of 'alpha' can be less than 1(proportionally decreases the number of filters in each layer), greater than 1(proportionally increases the number of filters in each layer) or equal to 1(default number of filters from the paper are used at each layer. We used the default value of 'alpha' which is 1. We did all the process exactly similar as we did in other architectures. After training our model with MobileNetV2, we got about 54.545% accuracy from our dataset.
Figure 4.31 shows the accuracy and loss of MobileNetV2 throughout the training process.

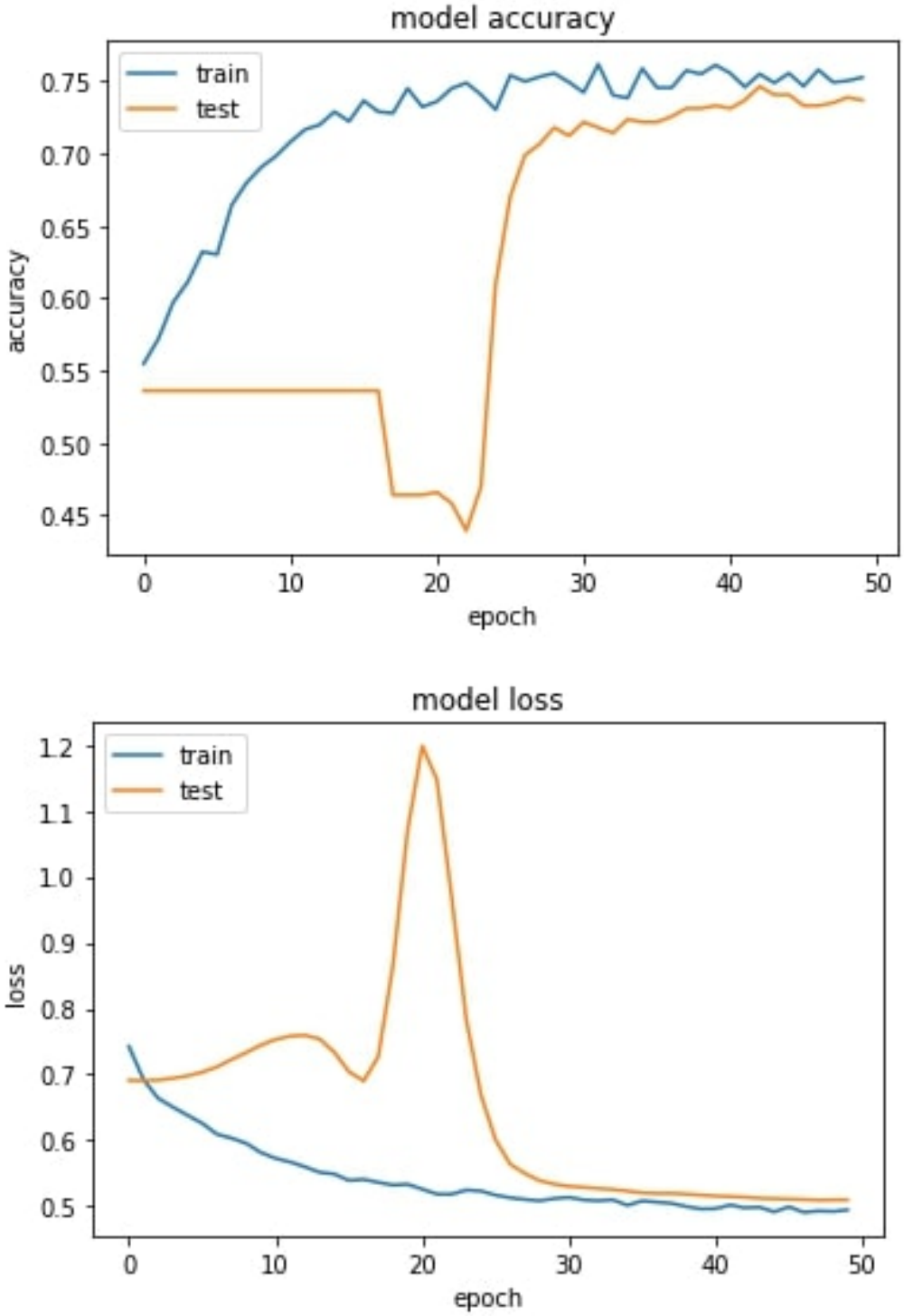

Figure 4.31: MobileNetV2 Model Accuracy(accuracy vs epochs) Model Loss(loss vs epochs

Figure 4.32 shows 10 sample images from the dataset along with their labels.

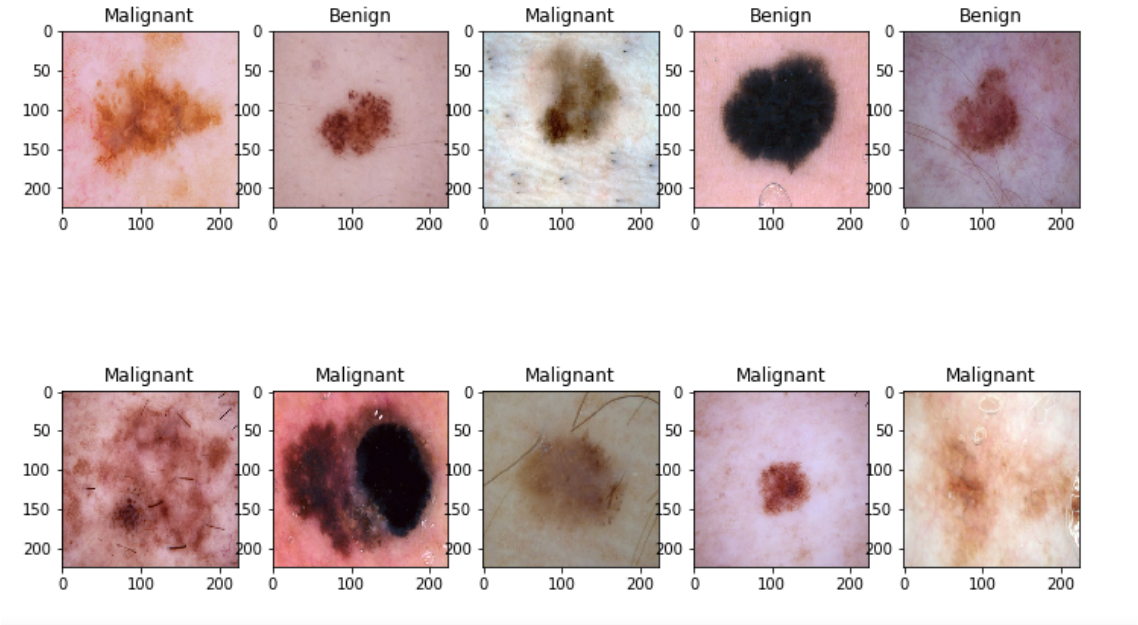

Figure 4.32: Output of original test data

Figure 4.33 shows the prediction of MobileNetV2 on those images.

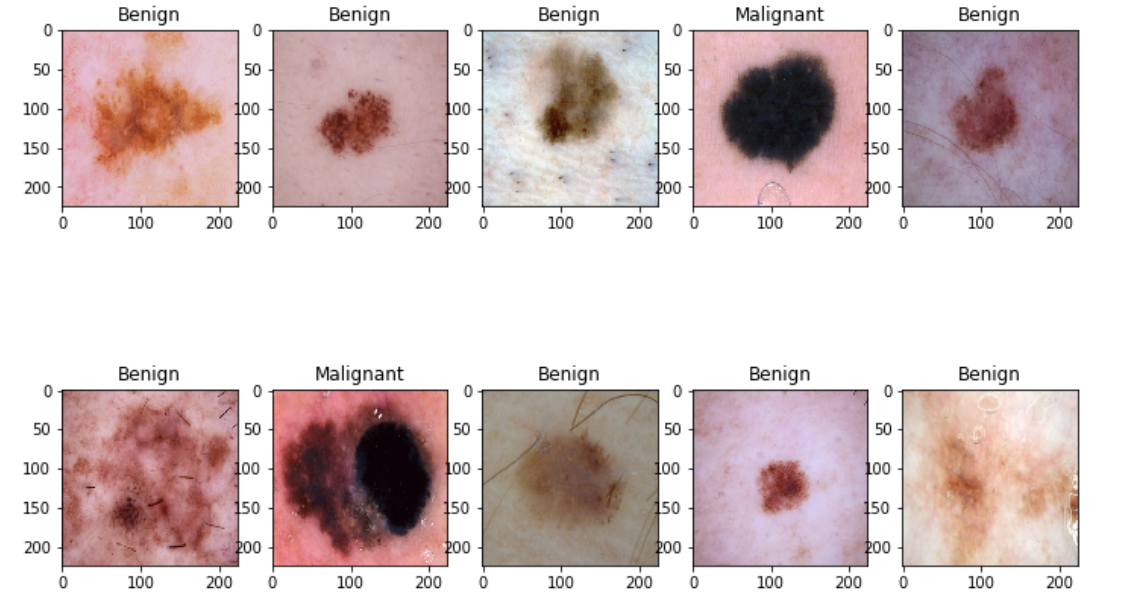

Figure 4.33: MobileNetV2 prediction on the same test data

The table below shows the loss,accuracy, validation loss and validation accuracy per 5 epochs of MobileNetV2.It is clearly shown that the accuracy was rising up over time and the loss was degrading.

Number of Epochs	Loss	Accuracy	Val Loss	Val Accuracy
5	0.5157	0.7416	0.6911	0.5360
10	0.4680	0.7629	0.6910	0.5360
15	0.4582	0.7814	0.6947	0.5360
20	0.4485	0.7724	0.7017	0.5360
25	0.4419	0.7857	0.7110	0.5360
30	0.4414	0.7918	0.7226	0.5360
35	0.4494	0.7809	0.7365	0.5360
40	0.4472	0.7895	0.7516	0.5360
45	0.4506	0.7771	0.7678	0.5360
50	0.4396	0.7871	0.7844	0.5360

Table 4.7: Outputs per 5 epochs in MobileNetV2 Architecture

4.6.4 MobileNet

MobileNet V1 consists of a regular 33 convolution layer followed by 13 blocks of 3x3 depthwise convolution, batch normalization, ReLu with 1x1 pointwise convolution, batch normalization and ReLu[12]. There are no pooling layers in between those depthwise separable convolution blocks. Stride of 2 is used to reduce the spatial dimension of those inputs. The number of output channels are also doubled in the pointwise layers. For a input shape 224x224x3 the output would be 7x7x1024 feature map. All these layers have batch normalization. MobileNet uses ReLU6 as its activation function. ReLU6 prevents activation from becoming too big. The architecture is completed with a global average pooling layer at the very end. That layer is implemented with a classification layer or a 11 convolution, and a softmax.

4.6.5 Applying MobileNet in Our Dataset

After applying MobileNetV2, we trained our model with MobileNet architecture to measure the accuracy of our dataset. We passed the same parameter as we used in MobileNetV2 architecture and applied them in MobileNet architectures along with other necessary parameterts. The only exception was to introduce a new variable named 'depth-multiplier' which was needed to compile MobileNet architecture. 'depth-multiplier' was introduced for depthwise convolution (also called as the resolution multiplier). After training our model with MobileNet, we got about 75% accuracy from our dataset.
Figure 4.8 shows the accuracy and loss of MobileNet throughout the training process.

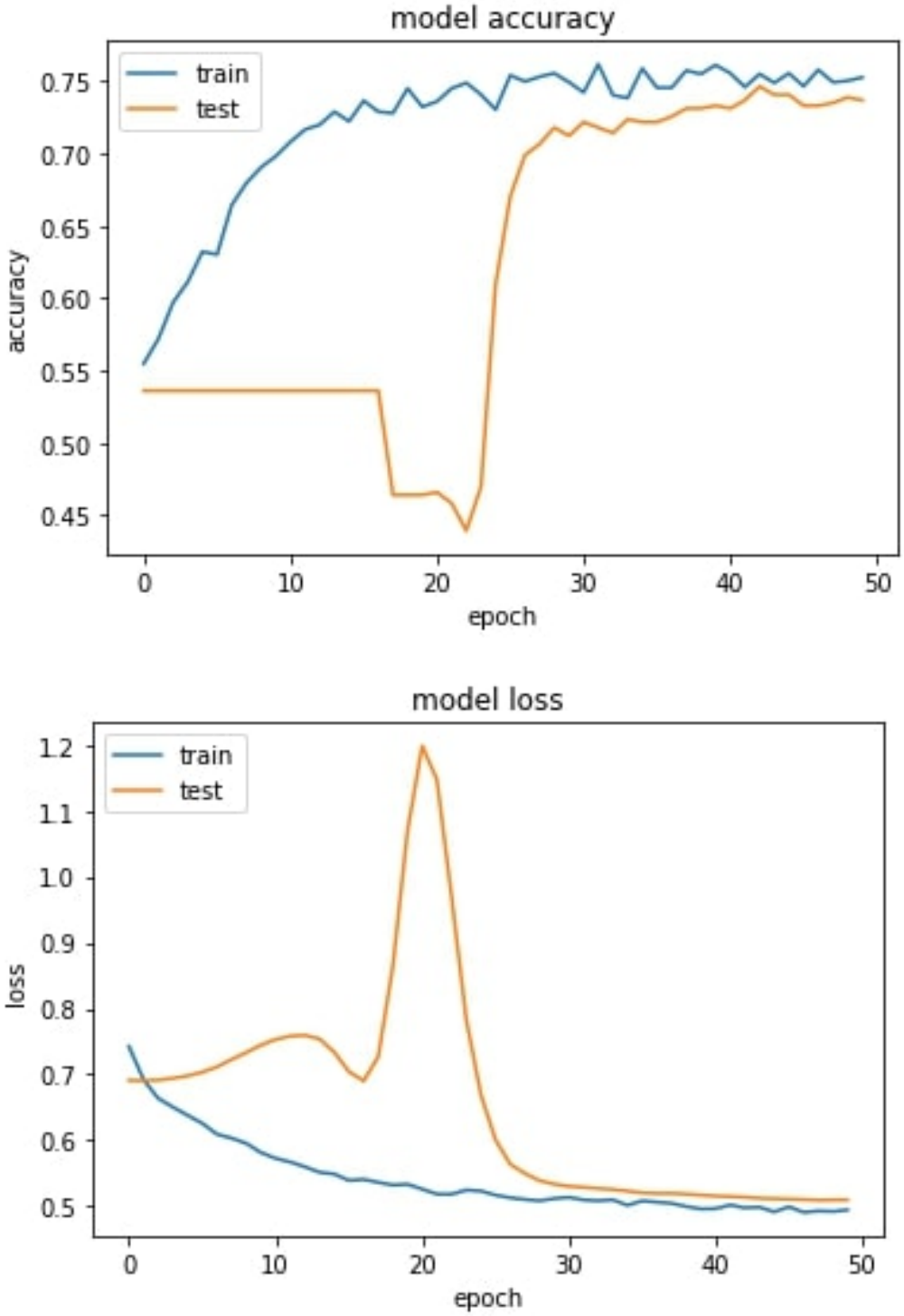

Figure 4.34: MobileNetV1 Model Accuracy(accuracy vs epochs) and Model Loss(loss vs epochs

Figure 4.34 shows 10 sample images from the dataset along with their labels.

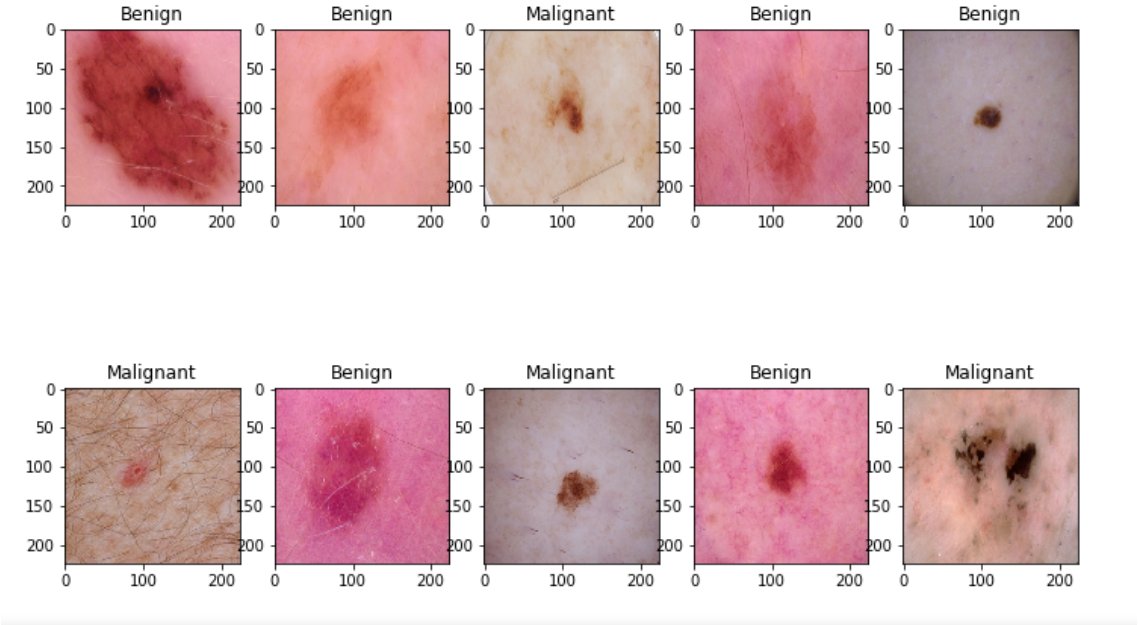

Figure 4.35: Output of original test data

Figure 4.35 shows the prediction of MobileNet on those images.

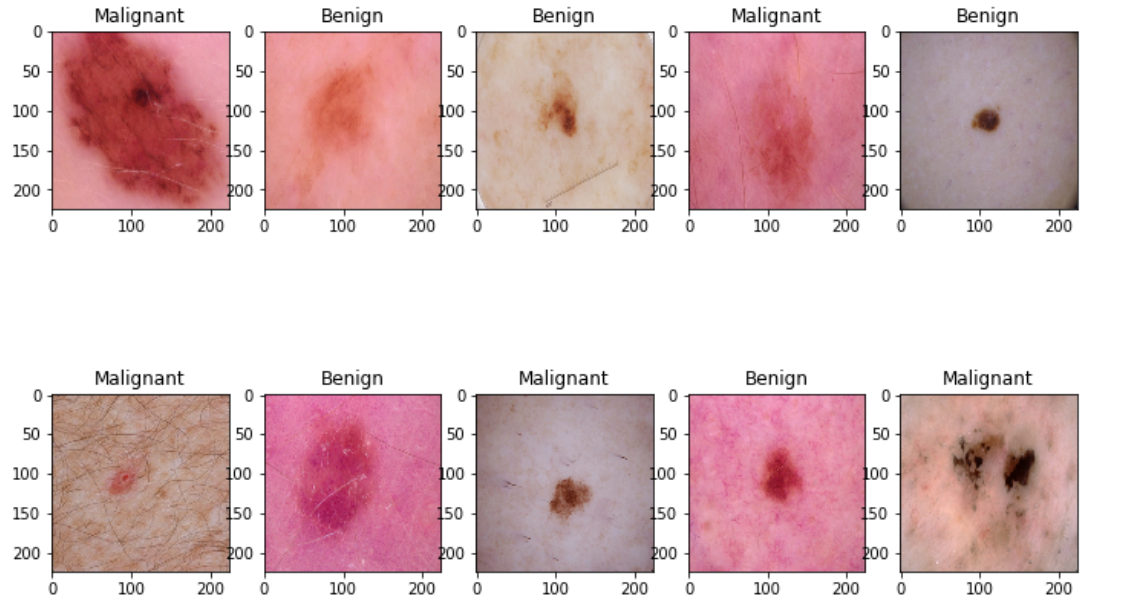

Figure 4.36: MobileNet prediction on the same test data

The table below shows the loss,accuracy, validation loss and validation accuracy per 5 epochs of MobileNet.It is clearly shown that the accuracy was rising up over time and the loss was degrading.

Number of Epochs	Loss	Accuracy	Val Loss	Val Accuracy
5	0.6381	0.6321	0.6979	0.5360
10	0.5815	0.6980	0.7446	0.5360
15	0.5490	0.7221	0.7338	0.5360
20	0.5334	0.7321	1.0703	0.4640
25	0.5232	0.7302	0.6694	0.6098
30	0.5118	0.7492	0.5336	0.7121
35	0.5015	0.7587	0.5232	0.7216
40	0.4960	0.7610	0.5170	0.7330
45	0.4915	0.7553	0.5115	0.7405
50	0.4945	0.7525	0.5097	0.7367

Table 4.8: Outputs per 5 epochs in MobileNet Architecture

4.7 Comparison of CNN Architectures

The table below shows the comperative analysis of Accuracy and Loss of different CNN architectures that we have applied.

Model Name	Accuracy	Loss
CNN	0.7530	0.2470
KFold	0.7053	0.2947
ResNet50	0.8242	0.1758
VGG16	0.8424	0.1576
VGG19	0.8454	0.1546
InceptionV3	0.8378	0.1622
Xception	0.8530	0.1470
MobileNetV2	0.5454	0.4546
MobileNet	0.7500	0.2500

Table 4.9: Comparative Analysis of Accuracy and Loss on Different CNN Architectures

As we have performed 7 CNN architectures along with our model and k-fold on the dataset, a huge number of variance in prediction is witnessed on those architectures. Out of those nine architectures, the worst accuracy we got was from mobilenetv2. It was only 54.54%. Also the highest accuracy we got was from Xception which was 85.30%.

The reasons behind this variance of accuracy are discussed below: Firstly MobilenetV2 which is a lightweight model was faster in training than most other architectures since it has bottleneck residual block that takes a lower dimensional vector as input and passes the same for the output. In this way MobilenetV2 reduces computational hazard and has a improve runtime. But unfortunately it ended up resulting in an unexpected accuracy of 54.54%. The accuracy during the epochs was higher than other CNN architectures and it reached 80% in the last epoch. Since MobilenetM2 does not have any activation function in its output block, this could be a reason behind this accuracy drop.

In k-fold we have got a validation accuracy of 70.53% with 3 folds. Since the model

was a simple one with sequential 4 layers in it only, it could not extract enough features to learn about the dataset. The highest accuracy we have got was in fold 2 where the accuracy was 80.43%.
MobileNet was the fastest of all the predefined architectures that we have used in our dataset. This architecture was trained in 1/3rd time of other architectures like ResNet50 or Xception. But the accuracy was only 75% which is lower than the simplest model we have built at the beginning. MobileNet does not have any pooling layer from the beginning through the very end. It has a global average pooling before the output layer. The input shapes were reduced with strided convolution. Depthwise separable convolutional block was the reason behind the improved runtime.
However, we have built a very simple sequential model with only 4 layers where there are two conv2d block with maxpooling and dropout and an output layer followed by a dense layer. Consequently, this model was the fastest in the training. The model was 12 times faster than VGG models. But we have got a marginal accuracy of 75.30%. Since the model was not deep enough, it could not extract enough features to distinguish between the melanoma of skin with a better accuracy.
ResNet50 is a deep network with 50 layers. This network builds residual block between the layers to solve the gradient vanishing problem and also for better learning in the training phase. This was a popular approach before the idea of depthwise separable convolution. This network uses maxpooling throughout the network with just one global average pooling in the output block. This model gave out 82.42% of accuracy in our dataset. InceptionV3 was the fastest model out of all the deep models we have used with our dataset. It was almost as fast as MobileNet and MobileNetV2 model. Also the accuracy was more than ResNet50. Since we know Inception is a multilevel feature extractor model which means 1x1, 3x3 and 5x5 conv blocks are computed within the same module. For this reason, this architecture was pretty deep with 48 layers and also computation time is faster. Inception uses maxpooling in those conv blocks and concatenates the output blocks with average pooling which means the best features are first selected from each block and then those features were generalized. This is why InceptionV3 gives out an accuracy of 83.78% with a comparatively lower run time.
VGG16 and VGG19 both are sequential architectures. Furthermore, they are quite identical in their structure. The only difference is that VGG19 has three more convolution blocks. Although these architectures are not deep enough, they can predict the output more accurately than many deeper models since data is learned and pass through the model in a sequence extracting the core features by using maxpooling. The accuracy of these two architectures is also on a margin of 30%. VGG16 gave out the accuracy of 84.24% and VGG19 was 84.54%. The only drawback VGG architectures have is that they are pretty much slower in terms of training and evaluating. On our dataset, these two architectures have the 2nd highest runtime.
Xception is one of the latest additions in CNN architectures. This architecture delivered the best accuracy out of all the architectures we have used on our dataset. There are certain reasons behind this. This architecture was built with the concept of previously defined architectures for example ResNet and Inception to be specific. This model uses an modified depthwise convolutional block which starts with a point-wise convolution. Furthermore, there is no non linearity between the blocks. Modified conv blocks are implemented throughout the model like Inception module.

There are residual blocks too between the layers. This architecture uses maxpooling to change the input shape in intermediate levels. Moreover, this architecture works better without any activation function. The accuracy of Xception on our dataset was 85.30%.
To describe the general comparison on the above mentioned architectures, we can see that the architectures using depthwise separable convolution has the faster runtime. Besides the architectures using maxpooling to change the input shape gives out better accuracy than the architectures using stridal convolution. Another thing can be witnessed that sequential architectures have better accuracy than non-linear models. To detect the melanoma types from the images of skin, these two factors could be the core reactors, sharp image generation and sequential learning.

Figure 4.37 shows the variance score of all the architectures.

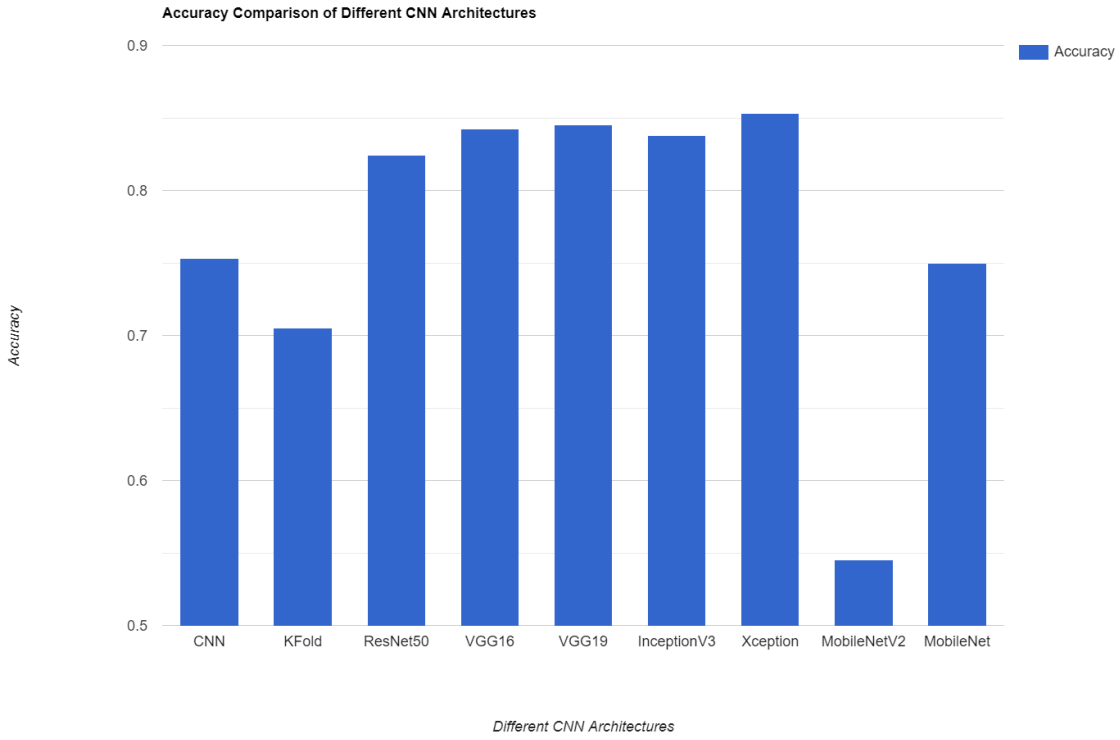

Figure 4.37: Explained Variance Score of all the Architectures

Figure 4.38 shows the loss of all the architecture

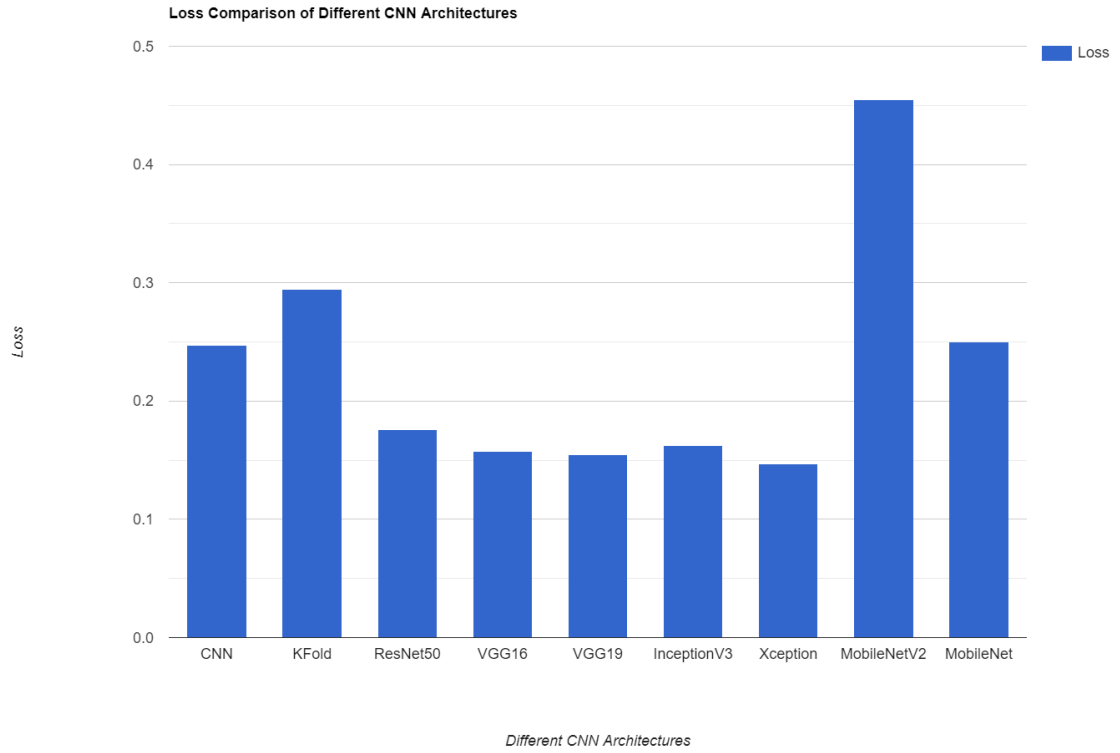

Figure 4.38: Value loss of all the Architectures

Chapter 5

Conclusion

The main goal of our thesis work was to find and create a suitable CNN model which can be trained and later on can be used to detect skin cancer more efficiently. During the research period we found couple of thesis work on this sector. However, most of the work was on only CNN. Comparing CNN architectures was our idea and we got quite a good result out of it. To train our model with CNN we found help from other research works, but we had to train our model with different CNN architectures by ourselves. Our model could be made even better if we changed the parameters, i.e. increasing the value of epochs, lowering batch size, changing the value of dropout etc. which would take longer period of times obviously. In our model, MobilenetV2 gave us the lowest accuracy with having almost 54.545% accuracy. The best architecture that we found for our dataset was the Xception. This architecture gave us about almost 85.303% accuracy for our dataset. Due to low power computer and other factors like, dataset image quality, image pre-processing, getting less accuracy can arise. Another concern that may occur that CNN takes a lot of time to process and train image data. Moreover, we had a dataset with having similar type of images, however when we will test it for real life implementation, we may get degraded accuracy than what we got now.

5.1 Future Possibilities

Detecting skin cancer using CNN and its architectures have a lots of opportunity to improve. The world is changing very fast, and the technology along with the whole computer world is changing very fast as well. Everything is getting faster and more efficient. Thus, may be with a better image classifier algorithm we can improve our detection accuracy and help doctors to find skin cancer faster for any patients. In Keras, it also provides the way to create and build anyone's own CNN model according to their needs. Thus, by learning all the possibilities of all the parameters that have been used in our model, we can define our own CNN architecture which will overcome all the obstacles to detect skin cancer more efficiently. Moreover, we are still trying to figure out a best process to combine all the architectures together and then show the combined result, which is also known as the ensembling process. We are working to complete two types of ensembling, one is, to average the predicted output of the each models and then predict the output of an test image with mostly predicted value from all the architectures. The second one is, training one model after another using the predicted output from the previous model as the input of the

next model and then finish all the algorithms in this process. Another important work that we are trying to do from our work is to convert our dataset into CSV file, means into numerical values and apply machine learning algorithms furthermore to determine the accuracy of our proposed system with a higher rate of accuracy.

Bibliography

[1] S. Jain, N. Pise, *et al.*, "Computer aided melanoma skin cancer detection using image processing", *Procedia Computer Science*, vol. 48, pp. 735–740, 2015.

[2] Simonyan, Karen, Zisserman, and Andrew, *Very deep convolutional networks for large-scale image recognition*, Apr. 2015. [Online]. Available: https://arxiv.org/abs/1409.1556/.

[3] M. H. Jafari, N. Karimi, E. Nasr-Esfahani, S. Samavi, S. M. R. Soroushmehr, K. Ward, and K. Najarian, "Skin lesion segmentation in clinical images using deep learning", in *2016 23rd International conference on pattern recognition (ICPR)*, IEEE, 2016, pp. 337–342.

[4] J. Kawahara, A. BenTaieb, and G. Hamarneh, "Deep features to classify skin lesions", in *2016 IEEE 13th International Symposium on Biomedical Imaging (ISBI)*, IEEE, 2016, pp. 1397–1400.

[5] E. Nasr-Esfahani, S. Samavi, N. Karimi, S. M. R. Soroushmehr, M. H. Jafari, K. Ward, and K. Najarian, "Melanoma detection by analysis of clinical images using convolutional neural network", in *2016 38th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, IEEE, 2016, pp. 1373–1376.

[6] V. Pomponiu, H. Nejati, and N.-M. Cheung, "Deepmole: Deep neural networks for skin mole lesion classification", in *2016 IEEE International Conference on Image Processing (ICIP)*, IEEE, 2016, pp. 2623–2627.

[7] A. Esteva, B. Kuprel, R. A. Novoa, J. Ko, S. M. Swetter, H. M. Blau, and S. Thrun, *Dermatologist-level classification of skin cancer with deep neural networks*, Jan. 2017. [Online]. Available: https://www.nature.com/articles/nature21056?draft=collection&proof=true.

[8] S. Hershey, S. Chaudhuri, D. P. Ellis, J. F. Gemmeke, A. Jansen, R. C. Moore, M. Plakal, D. Platt, R. A. Saurous, B. Seybold, *et al.*, "Cnn architectures for large-scale audio classification", in *2017 ieee international conference on acoustics, speech and signal processing (icassp)*, IEEE, 2017, pp. 131–135.

[9] M. Hollemans, *Compressing deep neural nets*, Sep. 2017. [Online]. Available: https://machinethink.net/blog/compressing-deep-neural-nets/.

[10] H. Srivastava, *What is k-fold cross validation?*, Dec. 2017. [Online]. Available: https://magoosh.com/data-science/k-fold-cross-validation/.

[11] *Convolutional neural network (cnn)*, Oct. 2018. [Online]. Available: https://developer.nvidia.com/discover/convolutional-neural-network.

[12] M. Hollemans, *Mobilenet version 2*, Apr. 2018. [Online]. Available: https://machinethink.net/blog/mobilenet-v2/.

[13] B. Raj, *A simple guide to the versions of the inception network*, May 2018. [Online]. Available: https://towardsdatascience.com/a-simple-guide-to-the-versions-of-the-inception-network-7fc52b863202.

[14] M. Rizwan, *Vgg16 - implementation using keras*, Oct. 2018. [Online]. Available: https://engmrk.com/vgg16-implementation-using-keras/.

[15] F. Shaikh, *Deep learning in the trenches: Understanding inception network from scratch*, Oct. 2018. [Online]. Available: https://www.analyticsvidhya.com/blog/2018/10/understanding-inception-network-from-scratch/.

[16] *A guide to inception model in keras*, Mar. 2019. [Online]. Available: https://maelfabien.github.io/deeplearning/inception/.

[17] J. Brownlee, *How to use the pre-trained vgg model to classify objects in photographs*, Aug. 2019. [Online]. Available: https://machinelearningmastery.com/use-pre-trained-vgg-model-classify-objects-photographs/.

[18] P. Dwivedi, *Understanding and coding a resnet in keras*, Mar. 2019. [Online]. Available: https://towardsdatascience.com/understanding-and-coding-a-resnet-in-keras-446d7ff84d33/.

[19] V. M, "Melanoma skin cancer detection using image processing and machine learning", *International Journal of Trend in Scientific Research and Development*, vol. Volume-3, pp. 780–784, Jun. 2019. DOI: 10.31142/ijtsrd23936.

[20] T. Saba, M. A. Khan, A. Rehman, and S. L. Marie-Sainte, "Region extraction and classification of skin cancer: A heterogeneous framework of deep cnn features fusion and reduction", *Journal of medical systems*, vol. 43, no. 9, p. 289, 2019.

[21] C. Shorten, *Introduction to resnets*, May 2019. [Online]. Available: https://towardsdatascience.com/introduction-to-resnets-c0a830a288a4/.

[22] *Skin cancer statistics*, Jul. 2019. [Online]. Available: https://www.wcrf.org/dietandcancer/cancer-trends/skin-cancer-statistics.

[23] S.-H. Tsang, *Review: Xception - with depthwise separable convolution, better than inception-v3 (image...* Mar. 2019. [Online]. Available: https://towardsdatascience.com/review-xception-with-depthwise-separable-convolution-better-than-inception-v3-image-dc967dd42568.

[24] *Dagnetwork*. [Online]. Available: https://www.mathworks.com/help/deeplearning/ref/resnet50.html#mw_7e914435-71af-41f7-9ec9-8f2092892982.mw_6dc28e13-2f10-44a4-9632-9b8d43b376fe/.