TRIBHUVAN UNIVERSITY

INSTITUTE OF ENGINEERING

PULCHOWK CAMPUS

A progress report

On

Derma Doctor Assistant
(An Application Using Deep Neural Network)

By:

Srijana Bhusal (072/Bex/440)

Sunidhi Amatya (072/Bex/443)

Madhusudan Mainali (072/Bex/449)

Anil Kumar Sah (072/Bex/450)

DEPARTMENT OF ELECTRONICS AND COMPUTER ENGINEERING

LALITPUR, NEPAL

June, 2019

ACKNOWLEDGEMENT

We would like express our first and foremost gratitude towards the Department of Electronics and Computer Engineering, Pulchowk Campus for providing us the opportunity to undertake this project which has helped us learn and implement our skills and knowledge. We are grateful to the Head of Department, Dr. Surendra Shrestha and Deputy head of Department Mrs. Bibha Sthapit for providing us support and guidance throughout the project.

We would like to acknowledge Mr. Daya Sagar Baral for his continuous supervision, unwavering support, invaluable suggestions and consultation throughout the project which have propelled us to come a long way into accomplishing the project with the finest result.

Our thanks and appreciations also go to the authors of various papers that helped us gain the in-depth knowledge of various aspects. Finally, we would like to thank all the people who are directly or indirectly related for the successful completion of this project.

Sincerely,

Srijana Bhusal
Sunidhi Amatya
Madhusudan Mainali
Anil Kumar Sah

ABSTRACT

Skin diseases are becoming a common phenomenon these days as different types of allergies are increasing rapidly. Most skin diseases tend to pass on from one person to another and therefore it is important to control it at initial stages to prevent it from spreading. Dermatology is one of the most unpredictable and difficult terrains to diagnose due to its complexity. In most developing countries, it is expensive for a large number of people. The proposed system deals with the creation of an application that helps in diagnosis of Skin disease. However, our system may not be able to completely replace the manual system of disease detection but it is at least expected that it will pre-inform the victim about possible cases, mostly when they hesitate to go for a checkup unless it gets severe. Our project uses image processing and machine learning technology to detect diseases. The system consists of 2 parts- image processing and the machine learning. The image processing part deals with applying various filters to the images to remove noise and make them uniform. It is necessary to remove the unwanted elements from the image before processing else it will affect the output efficiency. The Machine learning part deals with the processing of data and generation of result. Also, to increase the efficiency and make the application smart and easy we have bridged the prediction of skin disease possible through mobile app called “Derma doctor assistant”.

Contents

Table of figure:

Table of table:

Table of equation:

1. INTRODUCTION

1.1 Background

In today’s world, people of different age groups are suffering from skin diseases such as eczema, scalp ringworm, skin fungal, skin cancer of different intensity, psoriasis etc. These diseases strike without warning and have been one among the major disease that has life risk for the past ten years. If skin diseases are not treated at earlier stage, then it may lead to complications in the body including spreading of the infection from one individual to the other. The skin diseases can be prevented by investigating the infected region at an early stage. It is important to control it at initial stages to prevent it from spreading. Also, damage done to the skin through skin diseases could damage the mental confidence and wellbeing of people. Therefore, this has become a huge problem among people and it has become a crucial thing to treat these skin diseases properly at the initial stages itself to prevent serious damage.

Many of the skin diseases are very dangerous, particularly if not treated at an early stage. Skin diseases are becoming common because of the increasing pollution. Skin diseases tend to pass from one person to another. Human habits tend to assume that some skin diseases are not serious problems. Sometimes, most of the people try to treat these infections of the skin using their own method. However, if these treatments are not suitable for that particular skin problem then it would make it worse. And also, sometimes they may not be aware of the dangerousness of their skin diseases, for instance skin cancers. With advance of medical imaging technologies, the acquired data information is getting so rich toward beyond the human’s capability of visual recognition and efficient use for clinical assessment.

An automatic medical images analysis system has usually three stages: (1) Proper Enhancement, (2) feature extraction and selection (3) Classification. The proper enhancement is the most important, since it affects the precision of the subsequent steps. Supervised enhancement is somewhat easy to implement by varying its parameters for variety of lesion shapes, sizes, and colors along with diverse skin types and textures. The proposed system thus helps in classifying the different skin types through image processing.

An android based mobile app will be detecting the disease remotely. The user will be able to capture the infected area through the simple mobile camera accessed by the app and the prediction model will display the possible list of diseases.

1.2 Objectives

- To implement neural network and image processing to detect various skin diseases.
- To provide clinical assistance to detect skin disease on visually unrecognized areas by medical personnel.
-To help medical field change their conventional way of diagnosing disease, increasing the speed and reliability.
-To collect local dataset and contribute in data warehouse from Nepal so that further improvements and analysis can done over the data.
-To develop an android app to simplify the peoples' queries by fighting the hesitation that people may have regarding various skin diseases including STDs.

1.3 Problem Statement

Engineering technologies have been the main backbone for the existence of medical field these days. We cannot see almost any fields of medicine aloof and untouched by engineering mind. Right from heavy MRI and CT-SCAN machines to small needles in the hospitals, engineering tools have covered almost every nooks and corners of hospitals. But still we can see some of the diagnosis trends have been following manual system and people have to wait sometimes for weeks and months to get their culture reports and diagnose disease which sometimes can prove to be too late and lethal to people. Had they been aware in time there would be less chance of the severity of disease.

Also, it's a general trend that people in our country are mostly shy to share their disease with others due to possible humiliation they might face from people. Not only this we ourselves under look many people who get skin infections and try to avoid them in order to prevent ourselves from getting attacked with the same infection. In most of the cases isolation is surely required but sometimes people get boycotted from society because of the same problem and had they been treated in time they could have prevented the possible humiliation as well as saved their health. So, our project targets to assist both the doctor and the infected patient to determine the possible disease. The project targets in solving the problem of general people who can easily detect their problem by simple clicks on their mobile app and on the other hand assists doctors in pre predicting the disease and taking necessary steps.

1.4 Organization of Report

The rest of this report is structured as follows:

- **Chapter 2: Literature Review**

This section discusses about the related work and research that have been carried out on skin diseases of different types. These works and research carried out by research personnel that directly and indirectly have contributed significantly to our project. It also discusses about the related technologies and different approaches for skin disease detection, dataset cleaning and android interfacing.

- **Chapter 3: Methodology**

This chapter covers overall methodology used in our project. This chapter is about the implementation of the rules and algorithms for the collected dataset of various skin diseases. The challenges faced during implementation and how they were overcome are discussed. The methodology includes data collection, data preprocessing, data training and model comparison.

- **Chapter 4: Output**

The snapshot of our current system are shown highlighting the features and User Interface of the system.

- **Chapter 5: Results and Analysis**

This section reports the evaluation and analysis of the designed rules. It discusses the performance metrics, evaluation process and the result analysis. Reasons and recommendations for the improvements of the rules are also discussed.

- **Chapter 6: Conclusion**

This chapter is the summary of the achieved objectives and goal of the project. It also discusses about the limitations of the current system recommendation for the future enhancement.

2. LITERATURE REVIEW

Looking into the current situations of computerized skin disease diagnosis systems, there are few solutions available which are still under research developments. Certain limitations and drawbacks are identified in those hence this solution tries to overcome the existing problems with different approaches. Many medical related approaches have been made to classify the skin diseases. Many research articles have been published based on the skin diseases identification and classification. More reliable representation schemes and features extraction algorithms being developed, tremendous progress has been made in the last decade towards classification problems under variation in viewpoint, illumination and under partial occlusion.

2.1 Related Works

Many feature extraction-based models have been developed and are applied in the MATLAB to solve the classification problems. The problem of plants skin disease detection has been proposed in the paper "Skin Diseases Detection Models using Image Processing: A Survey" by Nisha Yadav, Virender Kumar Narang, Utpal Shrivatava, March, 2016, where the approach of image processing and Artificial Neural Network (AAN) was made. In the paper "Cancer Detection Using Image Processing Techniques" by Mokhld S. AL-Tarawneh 20, January-June 2012 the problem of Lung cancer detection was studied where the disease was rectified using Image segmentation and feature extraction. Detecting the human skin was studied in "Detection and Identification of Human Skin Diseases Using CIE lab Values" paper by C.B. Tatepamulwar, V.P. Pawar, K.S. Deshpande, H. S. Fadewar, June 6,2016. And many more such papers have given their different model to solve the classification problems. Some suffers issue like segmentation problems and some with the image noises.

3. METHODOLOGY

3.1. System Architecture

The overall system architecture of our system is shown below. User can simply take snaps of the infected body part and get the predicted possible disease through mobile app interface. A database holding the trained module is working at the background of the app and information from the database is retrieved whenever necessary.

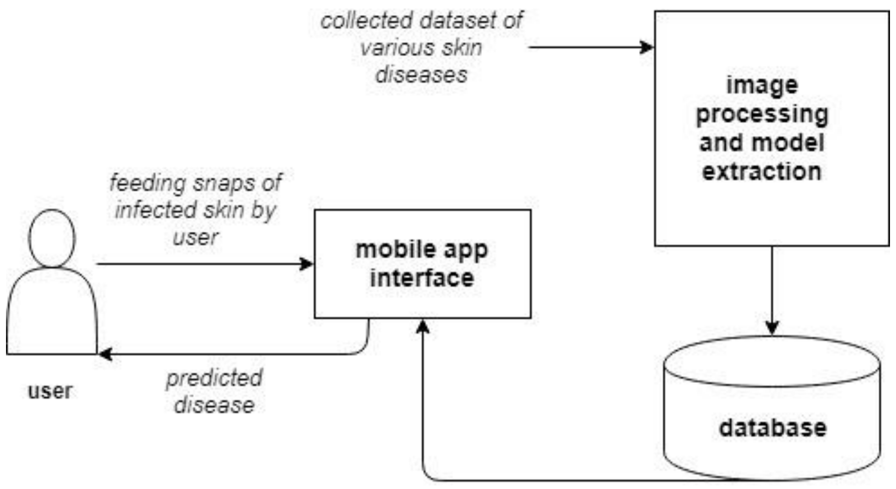

collected dataset of various skin diseases
image processing and model extraction
feeding snaps of infected skin by user
mobile app interface
user
predicted disease
database

Figure 1: Overall System Architecture

The proposed system involves the image analysis of skin disease of in different layer of deep learning. The system involves capturing the image of skin disease and compared with the previous saved trained model to make it recognizable. The trained model is obtained through the training of raw data with particular learning rate, filters and passed through many visible and hidden layers of deep learning. The obtained model also contains the test model having less data as compared to trained model. These models are used in analysis of image which are raw data obtained from medical fields. The raw data required for our proposed system are obtained from the hospital. Thus, in brief our system will able to make analysis of skin disease's image which are used to be analyzed. The flowchart of data preprocessing and model extraction method involves the following steps.

- Data collection
- Data preprocessing and model extraction
- Information extraction
- Classification

3.2. Data Collection

Collecting dataset of several skin diseases became one of the most difficult part of our project. We put all our efforts to work on local data set of our country itself but due to several tedious procedures related to ethical clearance and the task took longer than expected time. However, we have been able to collect few datasets from different hospitals and clinics but they weren't sufficient to produce satisfactory training results.

However, we took an alternative step to browse all the possible sites in internet and found few helpful links through which we crawled the data and have been using them for training our model currently. We have been using dataset from DermNet currently and selected ten different diseases for training. Following diseases are considered for the project:

a) Acne and Rosacea
b) Cellulitis Impetigo and other Bacterial Infections
c) Eczema
d) Herpes HPV and other STDs
e) Light Diseases and Disorders of Pigmentation
f) Lupus and other Connective Tissue diseases
g) Melanoma Skin Cancer Nevi and Moles
h) Psoriasis pictures Lichen Planus and related diseases
i) Scabies Lyme Disease and other Infestations and Bites
j) Tinea Ringworm Candidiasis and other Fungal Infections

3.3. Data Pre-processing model extraction:

All of our task has been performed in a high-level neural network API called Keras running on the top of TensorFlow. We have trained our dataset on our laptop itself which supports NVDIA GEFORCE GTX graphics card with 8 GB Ram. Hence, all the features have been set accordingly to train our data. Earlier we gave a thought of running separate hair removing filters but the tasks was worth another project itself and we didn't have much time to focus on that part. However, we have trained our data successfully as well in this existing dataset.

3.3.1 Image Augmentation

To achieve a good performance, we performed image augmentation using ImageDataGenerator API in Keras. Image augmentation creates training images artificially through different ways of processing or combination of multiple processing, such as random rotation, shifts, shear and flips, etc. we created our dataset and generated them as well with desired properties. Following properties were considered in our case:

rescale=1./255

rotation range=20

width shift range=0.2

height_shift_range=0.2

horizontal flip=True

fill mode='nearest')

Figure 2: Image flips via the horizontal_flip and vertical_flip arguments

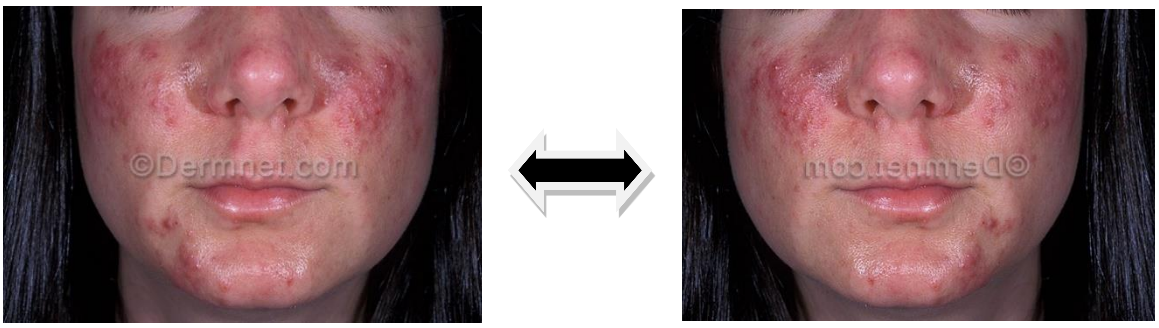


Fig: Original image

Fig: vertically Flipped image

Figure 3: Image rotations via the rotation_range argument

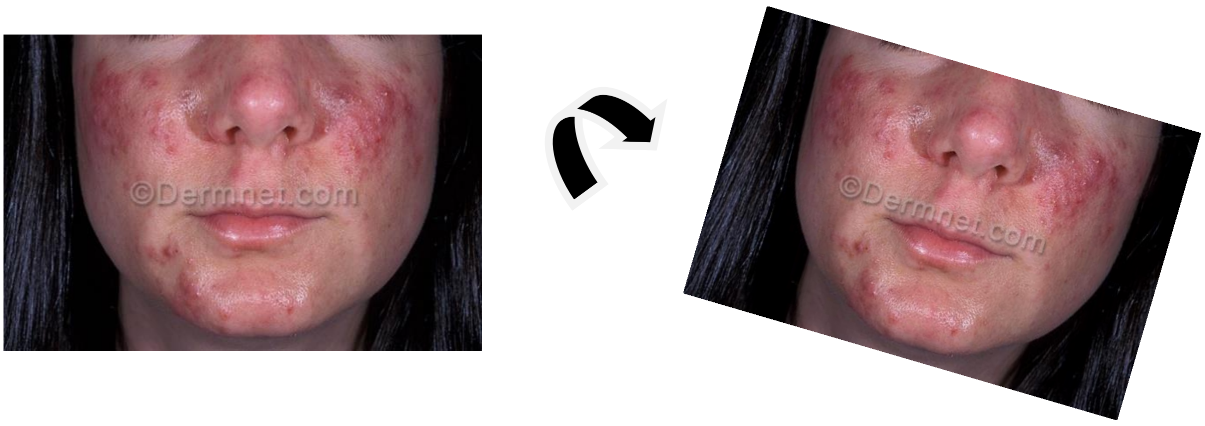


Fig: Original image

Fig: Rotated image

Figure 4: Image shear via shear argument

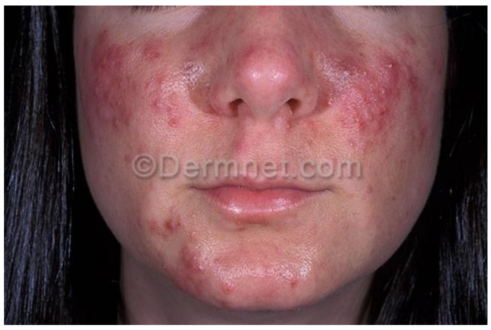


Fig: Original image

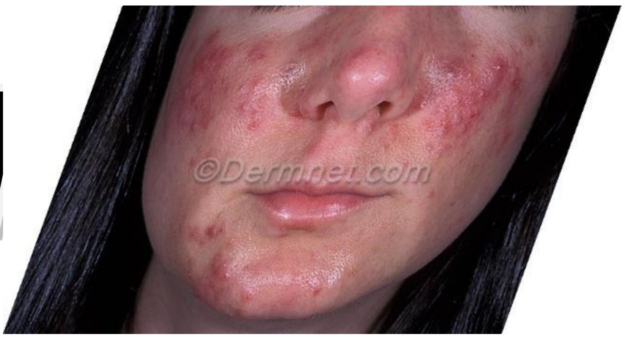


Fig: Sheared image

Figure 5: Image zoom via the zoom range argument.

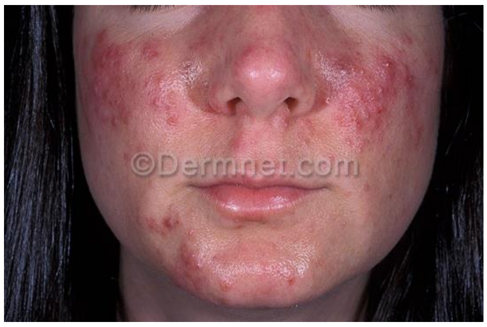


Fig: Original image

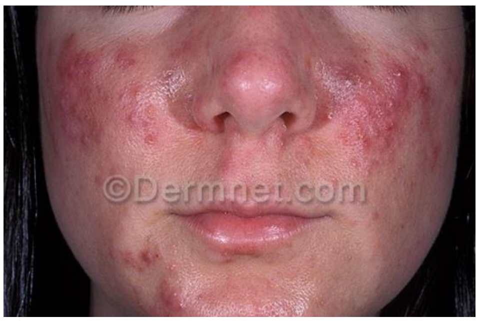


Fig: Zoomed image

Figure 6: Image width_shift_range

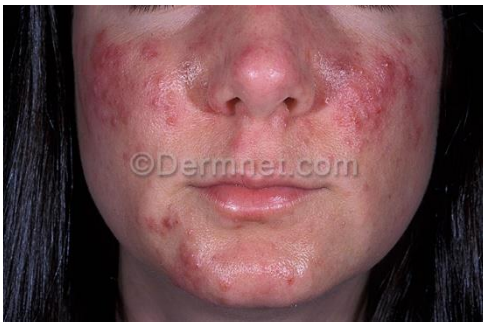


Fig: Original image

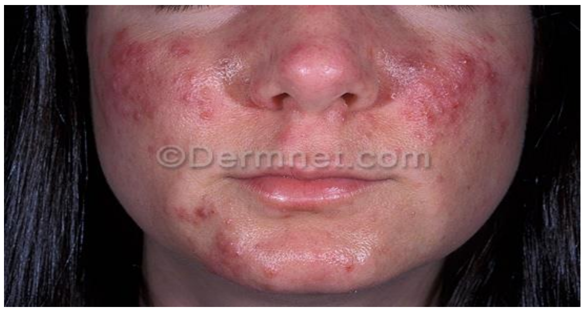


Fig: shifted image

Figure 7: Image height_shift_range

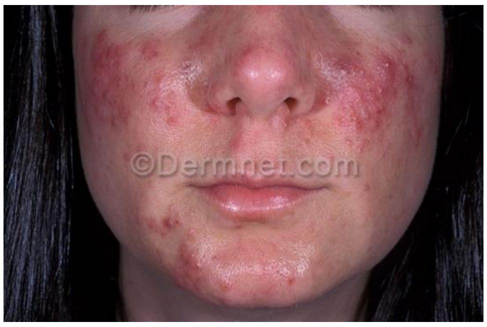


Fig: Original Image

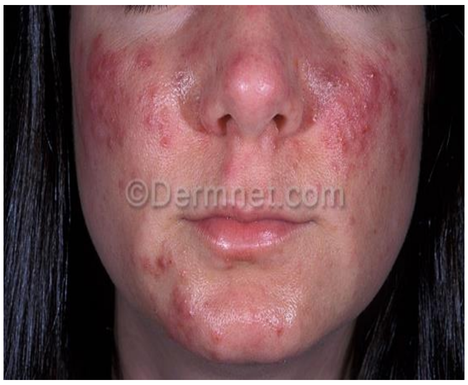


fig: height_shifted_image

3.3.2 Model Formation

For the model extraction,a ll of our task has been performed in a high-level neural network API called Keras running on the top of TensorFlow. We have trained our dataset on our laptop itself which supports NVDIA GEFORCE GTX graphics card with 8 GB RAM. Hence, all the features have been set accordingly to train our data. Earlier we gave a thought of running separate hair removing filters but the tasks was worth another project itself and we didn’t have much time to focus on that part. However, we have trained our data successfully as well in this existing dataset.

3.3.2.1 Keras Workflow

Keras provides a very simple workflow for training and evaluating the models. It is described with the following diagram:

The block diagram of Keras Workflow is figured below:

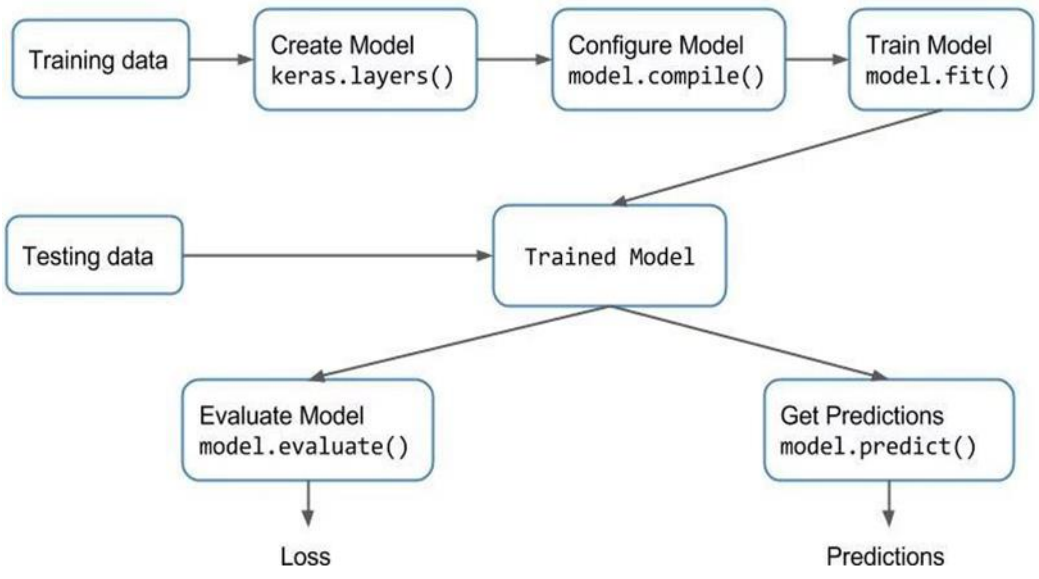


Figure 8: Block diagram of keras workflow

For creating the model, we have used the Convolution Neural Network, so that the model is trained by feeding the input picture via adjusting the weights and biases of the neural network. . A sequence of convolution, maxpooling and normalization is done in several layers of CNN and is finally regularized. At the end of the network, probability is calculated for each class and prediction is made on the basis of probabilities.

Here is the overview of different layers of CNN and its module we used.

3.3.2.2 Convolution Layer

A convolution is an orderly procedure where two sources of information are intertwined. A kernel, also called a filter, is a smaller-sized matrix in comparison to the input dimensions of the image that consists of real valued entries. Kernels are then convolved with the input volume to obtain so-called 'activation maps' also called feature maps.

$$s(t) = \int x(a) . w(t - a) da$$

The above operation is called convolution. The convolution operation is typically denoted with an asterisk (*):

$$s(t) = (x * w)(t)$$

$$\text{i.e. } s(t) = (x * w)(t) = \int x(a). w(t - a) da$$

In convolutional network terminology, the first argument (in this example, the Function x) to the convolution is often referred to as the input, and the second argument (in this example, the function w) as the kernel. The output is sometimes referred to as the feature map. The patch selection is then slided (towards the right and then downwards just like window when the boundary of the matrix is reached based on the stride value) by a certain amount called the 'stride' value, and the process is repeated till the entire input image has been processed.

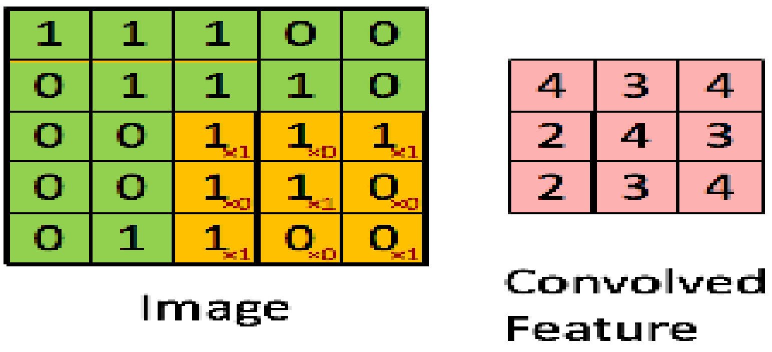


Figure 9: Convolution of input image

3.3.2.3 Zero-Padding:

Zero-padding adds zeros around the border of an image.

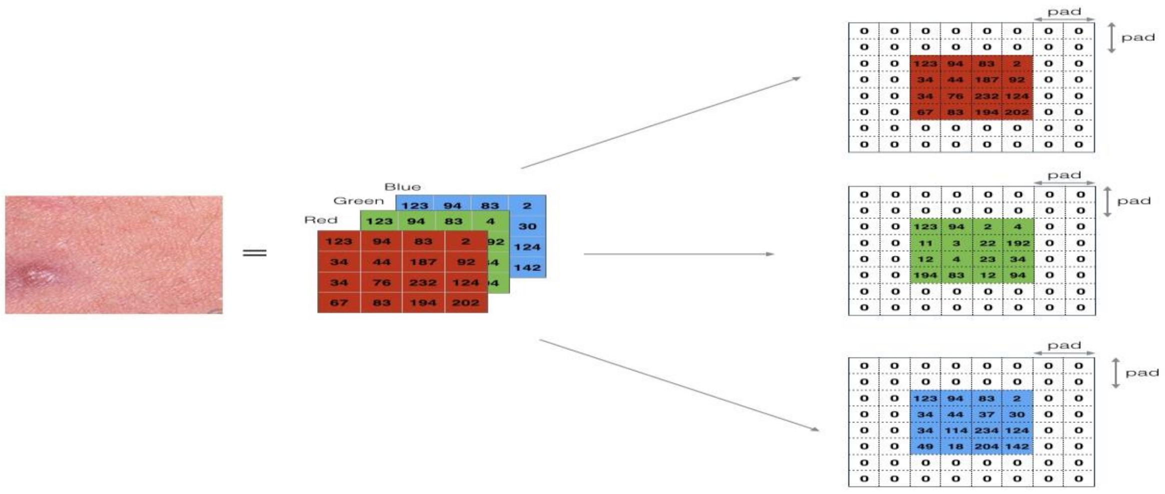


Figure 10: Figure of Zero-padding

3.3.2.4 Activation Layer

Each nodes value along their weighted value are multiplied and then are added with some bias value, and then they are passed through an activation function so the it will able to decide which neurons is to be turned ON or OFF. Only non-linear activation function is used between the subsequence convolution layers for learning. If non-linear activation function is used then there will be not any learning. So, we have used ReLU activation function in our case, all the negative values to 0 so that a matrix has no negative values. ReLU is the abbreviation of Rectified Linear Units. A stack of images becomes a stack of images with no negative values. This layer applies the non-saturating activation function. Other functions are also used to increase nonlinearity, for example:

The saturating hyperbolic tangent f(x)=tanh(x), f(x)=|tanh(x)|, and the sigmoid function f(x)={1+e^(-x)}^(-1). ReLU is often preferred to other functions, because it trains the neural network several times faster without a significant penalty to generalization accuracy.

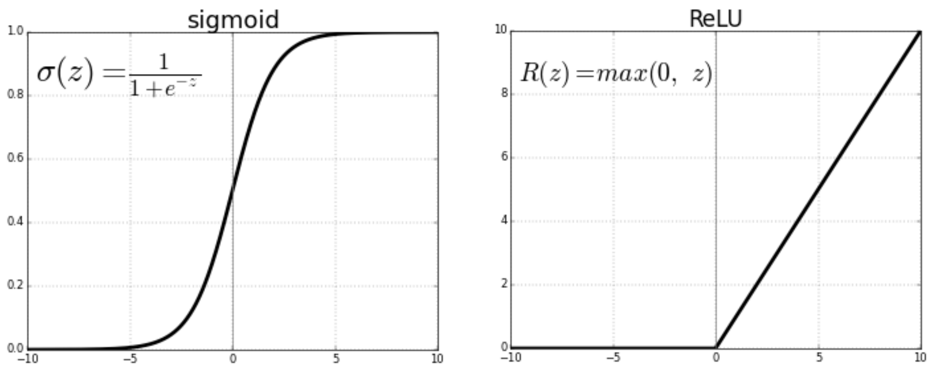


Figure 11: Logistic Sigmoid v/s ReLU

Many papers follow the convolution, pooling and activation order, but this isn't strictly the case. There also can be the order convolution, activation and pooling. Here, we have used the convolution, activation and pooling order.

3.3.2.5 Pooling Layer

Another important concept of CNNs is pooling, which is a form of non-linear down-sampling. There are several non-linear functions to implement pooling among which max pooling is the most common. It partitions the input image into a set of non-overlapping rectangles and, for each such sub-region, outputs the maximum. Pooling reduces the spatial dimensions (Width x Height) of the Input Volume for the next Convolution Layer. It does not affect the depth dimension of the Volume. The transformation is either performed by taking the maximum value from the values observable in the window (called ‘max pooling’), or by taking the average of the values. Max pooling has been favored over others due to its better performance characteristics.

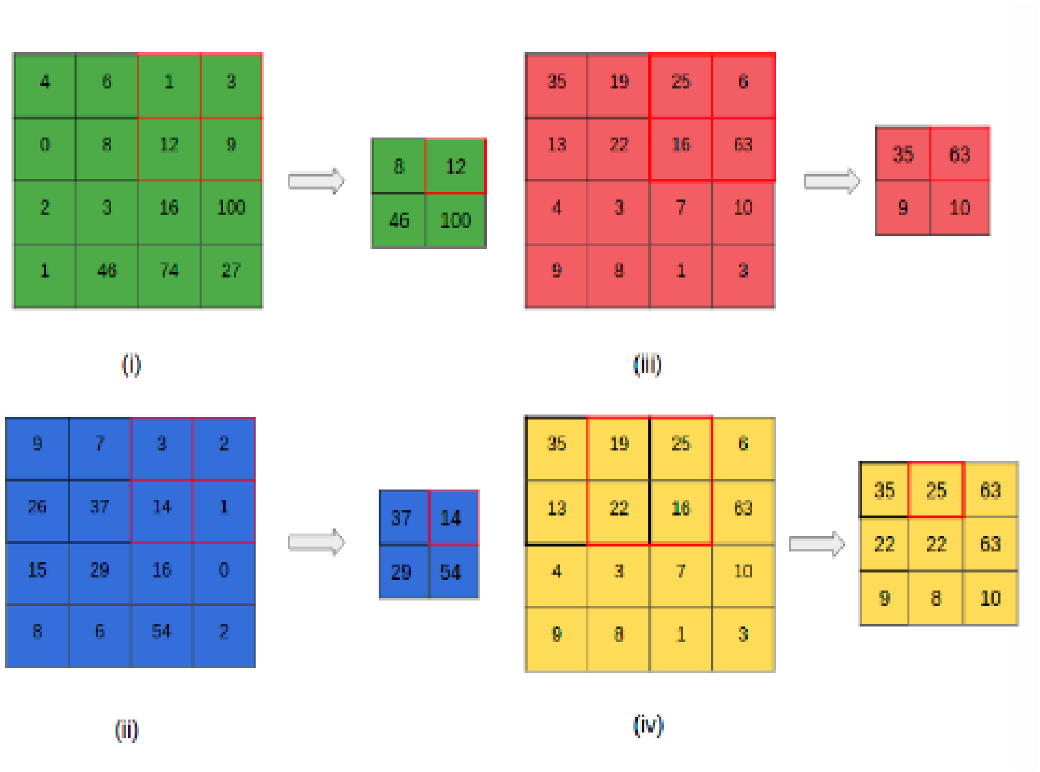


Figure 12: Figure of Pooling Layer

8.3.2.6 Fully Connected Layer

Finally, after several convolutional and max pooling layers, the high-level reasoning in the neural network is done via fully connected layers. Neurons in a fully connected layer have connections to all activations in the previous layer, as seen in regular neural networks. Their activations can hence be computed with a matrix multiplication followed by a bias offset. So why fully connected layer is used? They are used because the output from the convolution layer represent the high-level features of data. After passing through pooling, the output is obtained and they are flattened and connected to output layer adding a fully connected layer. It is usually cheap ways of learning non-linear combination of the features.

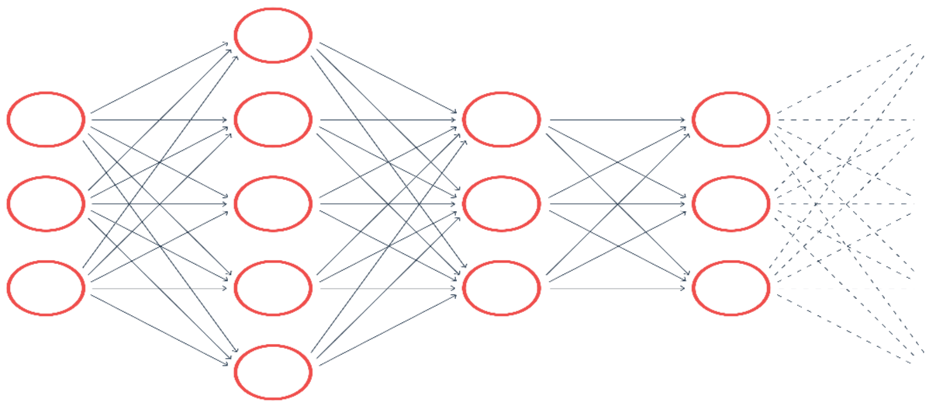


Figure 13: Densely connected layer

3.8.2.7 Loss Layer

The loss layer specifies how training penalizes the deviation between the predicted and true labels and is normally the final layer. Various loss functions appropriate for different tasks may be used there. The SoftMax function is used in various multiclass classification method. Cross-entropy loss is used for predicting K independent probability values in [0,1].Regularization is a process of introducing additional information to solve an ill-posed problem or to prevent overfitting. CNNs use various types of regularization. Regularization is a vital feature in almost every state-of-the-art neural network implementation. To perform dropout on a layer, you randomly set some of the layer's values to 0 during forward propagation. Dropout forces an artificial neural network to learn multiple independent representations of the same data by alternately randomly disabling neurons in the learning phase.

3.8.2.8 Optimizer

3.8.2.8.1 Gradient Descent with Momentum

Almost always, gradient descent with momentum converges faster than the standard gradient descent algorithm. In the standard gradient descent algorithm, you would be taking larger steps in one direction and smaller steps in another direction which slows down the algorithm. In the image shown below, you can see that standard gradient descent takes larger steps in the y-direction and smaller steps in the x-direction. If our algorithm is able to reduce the steps taken in the y-direction and concentrate the direction of the step in the x-direction, our algorithm would converge faster. This is what momentum does, it restricts the oscillation in one direction so that our algorithm can converge faster. Also, since the number of steps taken in the y-direction is restricted, we can set a higher learning rate.

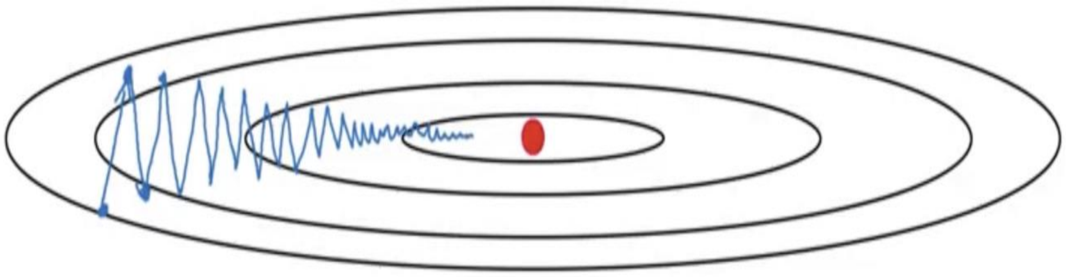


Figure 14: Gradient Descent with Momentum

$$v_{dw} = \beta \cdot v_{dw} + (1 - \beta) \cdot dw$$

$$v_{db} = \beta \cdot v_{dw} + (1 - \beta) \cdot db$$

$$W = W - \alpha \cdot v_{dw}$$

$$b = b - \alpha \cdot v_{db}$$

Equation 1: Gradient Descent with momentum

3.8.2.8.2 RMSprop optimizer

The RMSprop optimizer is similar to the gradient descent algorithm with momentum. The RMSprop optimizer restricts the oscillations in the vertical direction. Therefore, we can increase our learning rate and our algorithm could take larger steps in the horizontal direction converging faster. The difference between RMSprop and gradient descent is on how the gradients are calculated. The following equations show how the gradients are calculated for the RMSprop and gradient descent with momentum. The value of momentum is denoted by beta and is usually set to 0.9. If you are not interested in the math behind the optimizer, you can just skip the following equations.

$$v_{dw} = \beta \cdot v_{dw} + (1 - \beta) \cdot dw^2$$

$$v_{db} = \beta \cdot v_{dw} + (1 - \beta) \cdot db^2$$

$$W = W - \alpha \cdot \frac{dw}{\sqrt{v_{dw}} + \epsilon}$$

$$b = b - \alpha \cdot \frac{db}{\sqrt{v_{db}} + \epsilon}$$

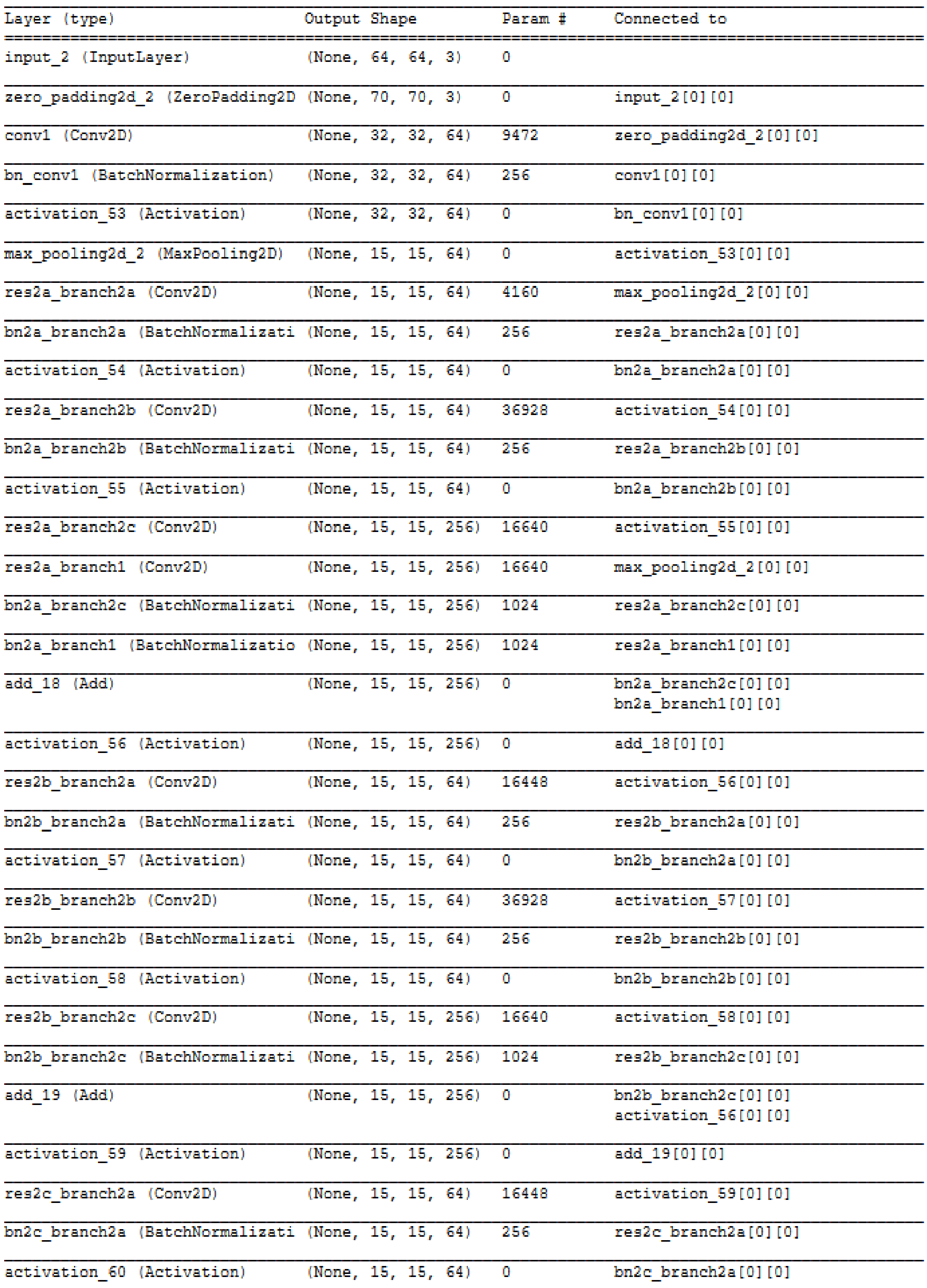
Equation 2: RMSprop optimizer

optimizer = optimizers.RMSprop(lr=0.00003, rho=0.9, epsilon=1e-08, decay=0.00001)

Here, (momentum)β=0.9, learning rate =0.00003, ε=1e-08 and decay rate =0.00001

The architecture of different model that we have implemented in the course of forming model are given below:

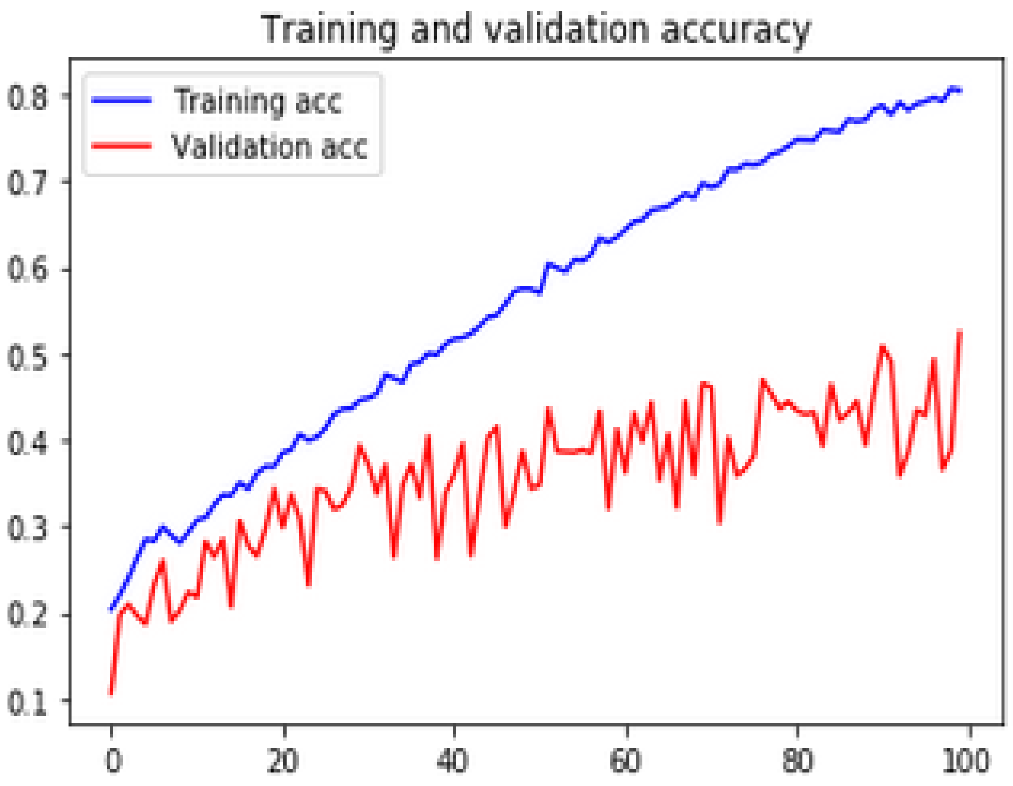
1. ResNet



```
________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
================================================================================================
input_2 (InputLayer)            (None, 64, 64, 3)    0
________________________________________________________________________________________________
zero_padding2d_2 (ZeroPadding2D (None, 70, 70, 3)    0           input_2[0][0]
________________________________________________________________________________________________
conv1 (Conv2D)                  (None, 32, 32, 64)   9472        zero_padding2d_2[0][0]
________________________________________________________________________________________________
bn_conv1 (BatchNormalization)   (None, 32, 32, 64)   256         conv1[0][0]
________________________________________________________________________________________________
activation_53 (Activation)      (None, 32, 32, 64)   0           bn_conv1[0][0]
________________________________________________________________________________________________
max_pooling2d_2 (MaxPooling2D)  (None, 15, 15, 64)   0           activation_53[0][0]
________________________________________________________________________________________________
res2a_branch2a (Conv2D)         (None, 15, 15, 64)   4160        max_pooling2d_2[0][0]
________________________________________________________________________________________________
bn2a_branch2a (BatchNormalizati (None, 15, 15, 64)   256         res2a_branch2a[0][0]
________________________________________________________________________________________________
activation_54 (Activation)      (None, 15, 15, 64)   0           bn2a_branch2a[0][0]
________________________________________________________________________________________________
res2a_branch2b (Conv2D)         (None, 15, 15, 64)   36928       activation_54[0][0]
________________________________________________________________________________________________
bn2a_branch2b (BatchNormalizati (None, 15, 15, 64)   256         res2a_branch2b[0][0]
________________________________________________________________________________________________
activation_55 (Activation)      (None, 15, 15, 64)   0           bn2a_branch2b[0][0]
________________________________________________________________________________________________
res2a_branch2c (Conv2D)         (None, 15, 15, 256)  16640       activation_55[0][0]
________________________________________________________________________________________________
res2a_branch1 (Conv2D)          (None, 15, 15, 256)  16640       max_pooling2d_2[0][0]
________________________________________________________________________________________________
bn2a_branch2c (BatchNormalizati (None, 15, 15, 256)  1024        res2a_branch2c[0][0]
________________________________________________________________________________________________
bn2a_branch1 (BatchNormalizatio (None, 15, 15, 256)  1024        res2a_branch1[0][0]
________________________________________________________________________________________________
add_18 (Add)                    (None, 15, 15, 256)  0           bn2a_branch2c[0][0]
                                                                 bn2a_branch1[0][0]
________________________________________________________________________________________________
activation_56 (Activation)      (None, 15, 15, 256)  0           add_18[0][0]
________________________________________________________________________________________________
res2b_branch2a (Conv2D)         (None, 15, 15, 64)   16448       activation_56[0][0]
________________________________________________________________________________________________
bn2b_branch2a (BatchNormalizati (None, 15, 15, 64)   256         res2b_branch2a[0][0]
________________________________________________________________________________________________
activation_57 (Activation)      (None, 15, 15, 64)   0           bn2b_branch2a[0][0]
________________________________________________________________________________________________
res2b_branch2b (Conv2D)         (None, 15, 15, 64)   36928       activation_57[0][0]
________________________________________________________________________________________________
bn2b_branch2b (BatchNormalizati (None, 15, 15, 64)   256         res2b_branch2b[0][0]
________________________________________________________________________________________________
activation_58 (Activation)      (None, 15, 15, 64)   0           bn2b_branch2b[0][0]
________________________________________________________________________________________________
res2b_branch2c (Conv2D)         (None, 15, 15, 256)  16640       activation_58[0][0]
________________________________________________________________________________________________
bn2b_branch2c (BatchNormalizati (None, 15, 15, 256)  1024        res2b_branch2c[0][0]
________________________________________________________________________________________________
add_19 (Add)                    (None, 15, 15, 256)  0           bn2b_branch2c[0][0]
                                                                 activation_56[0][0]
________________________________________________________________________________________________
activation_59 (Activation)      (None, 15, 15, 256)  0           add_19[0][0]
________________________________________________________________________________________________
res2c_branch2a (Conv2D)         (None, 15, 15, 64)   16448       activation_59[0][0]
________________________________________________________________________________________________
bn2c_branch2a (BatchNormalizati (None, 15, 15, 64)   256         res2c_branch2a[0][0]
________________________________________________________________________________________________
activation_60 (Activation)      (None, 15, 15, 64)   0           bn2c_branch2a[0][0]
```

Table 1:Architecture of ResNet

The training versus validation accuracy and loss so obtained by this architecture is:



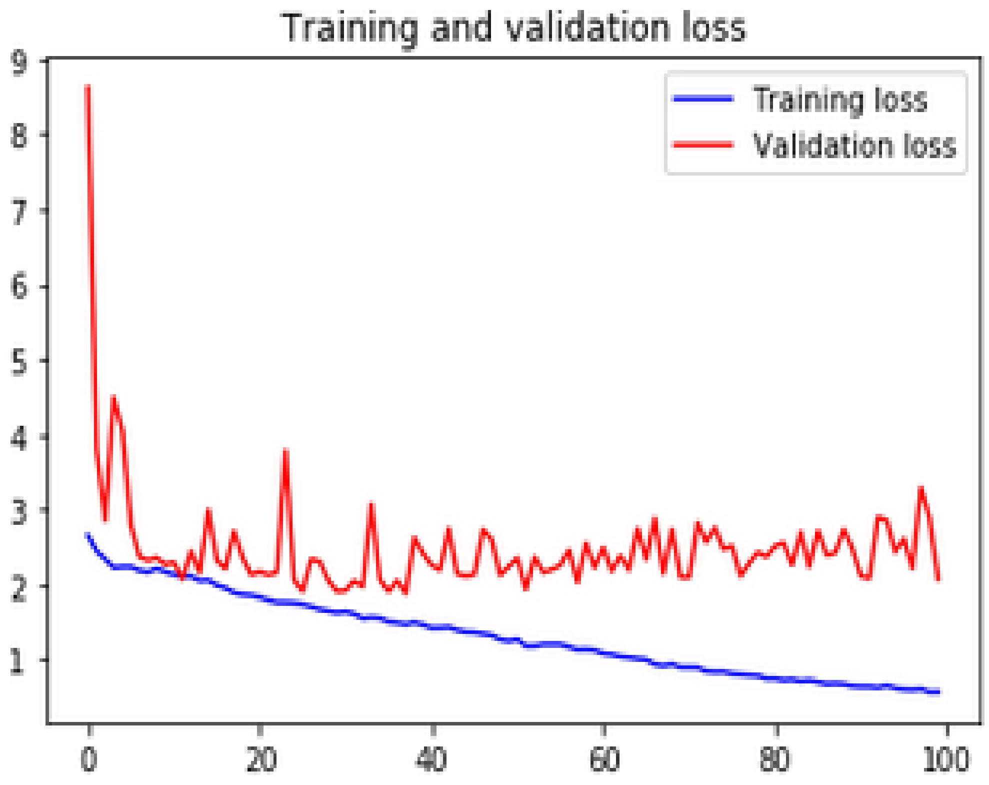
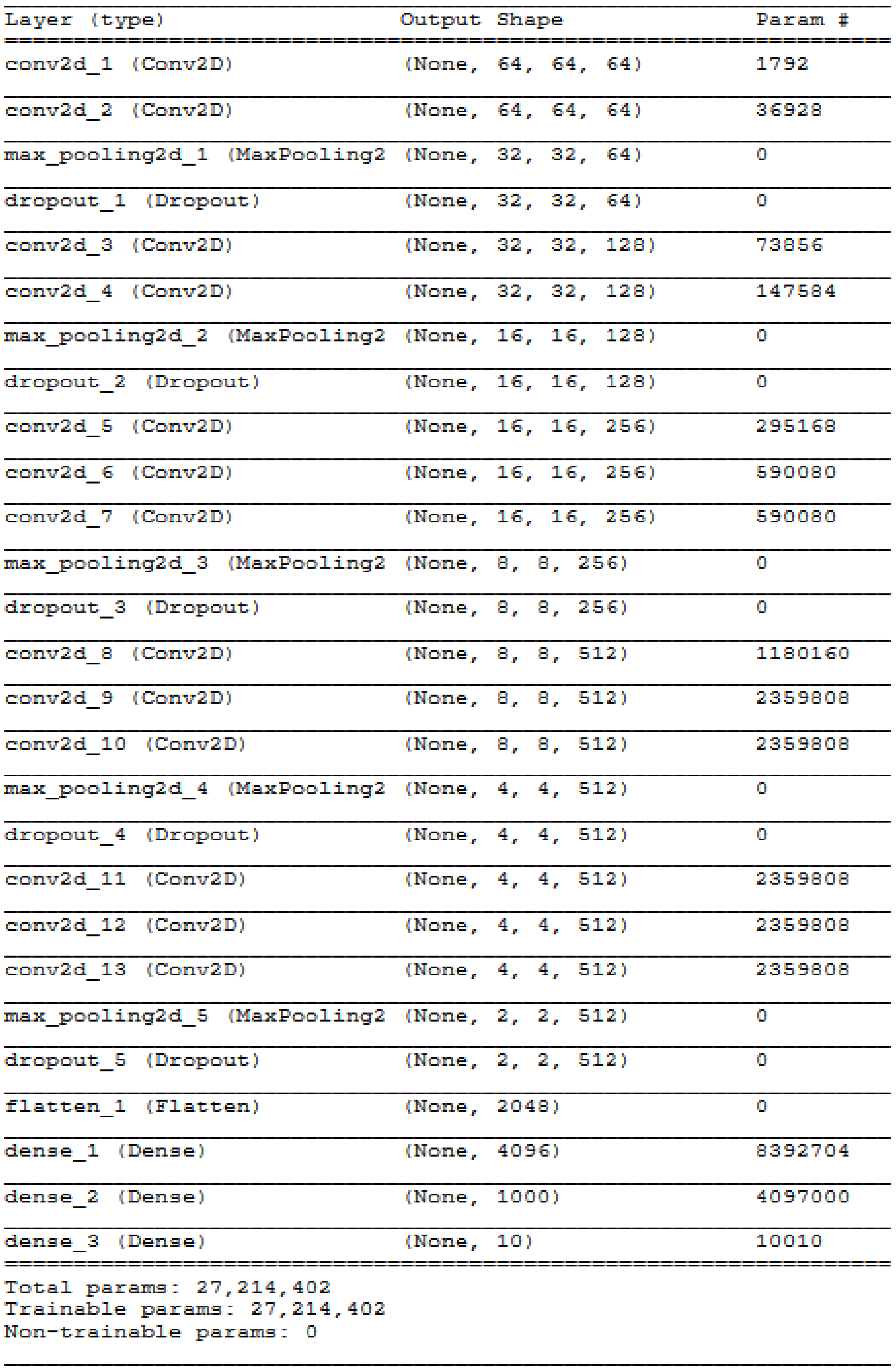


Figure 15: Figure of training versus validation accuracy and loss

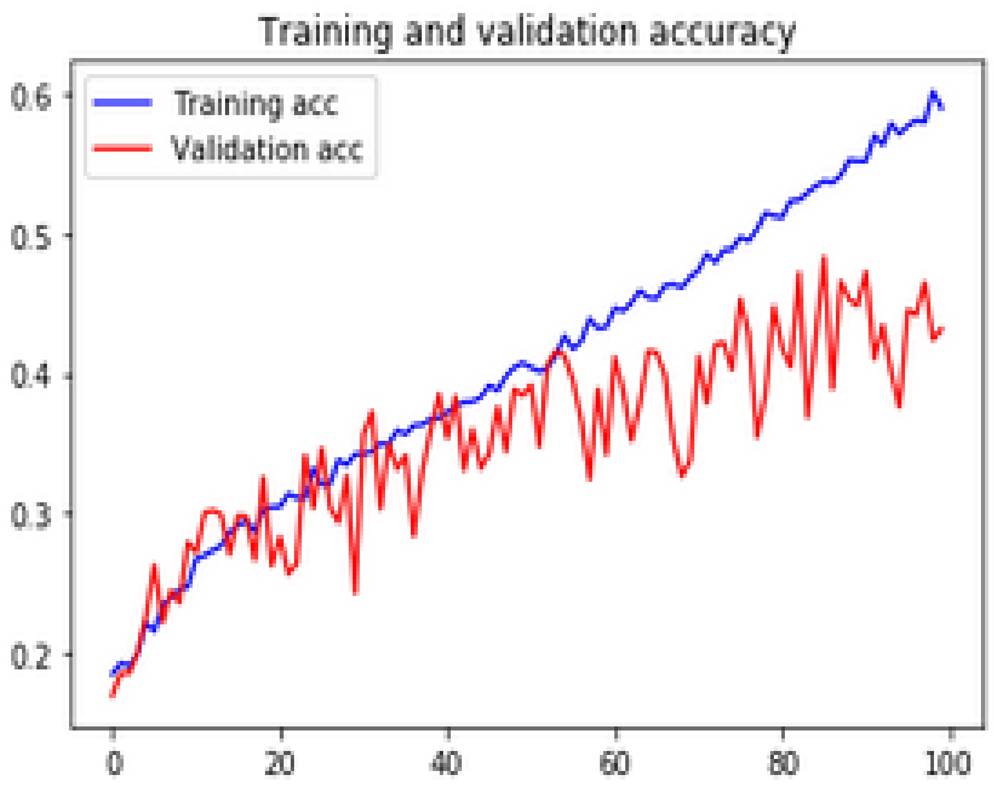
1. VGG16 with Adam optimizer



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 64, 64, 64)        1792
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 64, 64, 64)        36928
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 32, 32, 64)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 32, 32, 64)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 32, 32, 128)       73856
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 32, 32, 128)       147584
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 16, 16, 128)       0
_________________________________________________________________
dropout_2 (Dropout)          (None, 16, 16, 128)       0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 16, 16, 256)       295168
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 16, 16, 256)       590080
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 16, 16, 256)       590080
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 8, 8, 256)         0
_________________________________________________________________
dropout_3 (Dropout)          (None, 8, 8, 256)         0
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 8, 8, 512)         1180160
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 8, 8, 512)         2359808
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 8, 8, 512)         2359808
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 4, 4, 512)         0
_________________________________________________________________
dropout_4 (Dropout)          (None, 4, 4, 512)         0
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 4, 4, 512)         2359808
_________________________________________________________________
conv2d_12 (Conv2D)           (None, 4, 4, 512)         2359808
_________________________________________________________________
conv2d_13 (Conv2D)           (None, 4, 4, 512)         2359808
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 2, 2, 512)         0
_________________________________________________________________
dropout_5 (Dropout)          (None, 2, 2, 512)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 2048)              0
_________________________________________________________________
dense_1 (Dense)              (None, 4096)              8392704
_________________________________________________________________
dense_2 (Dense)              (None, 1000)              4097000
_________________________________________________________________
dense_3 (Dense)              (None, 10)                10010
=================================================================
Total params: 27,214,402
Trainable params: 27,214,402
Non-trainable params: 0
_________________________________________________________________
```

Table 2: Architecture of VGG16

The training versus validation accuracy and loss so obtained by this architecture is:



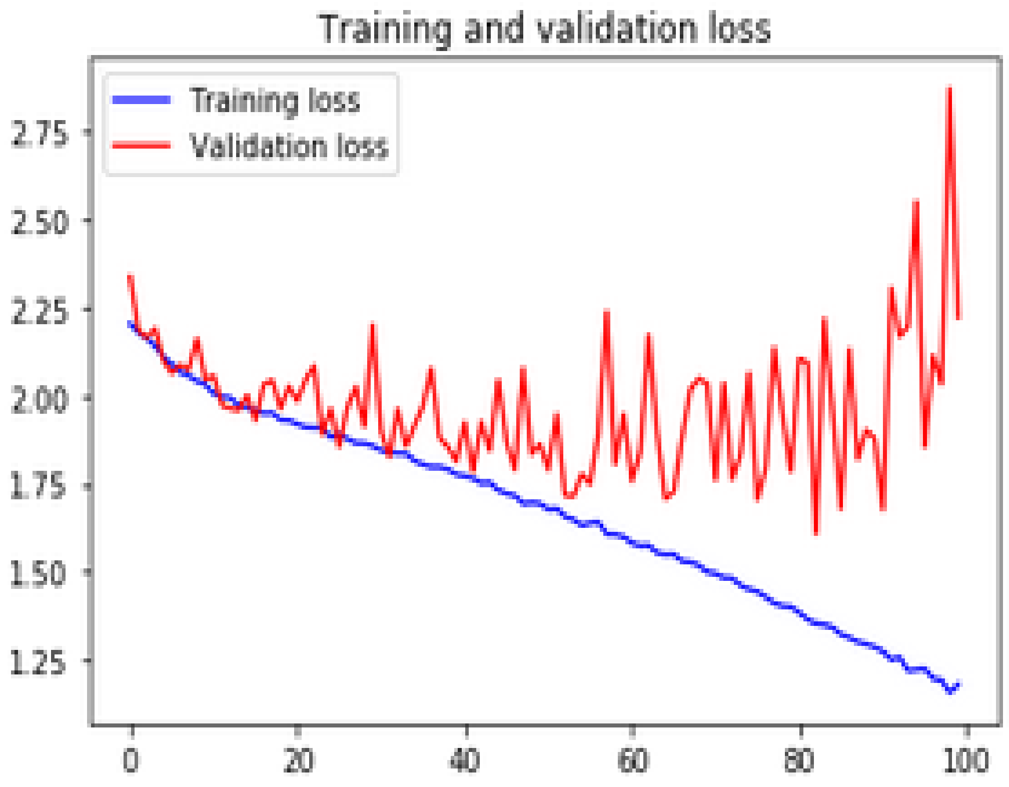
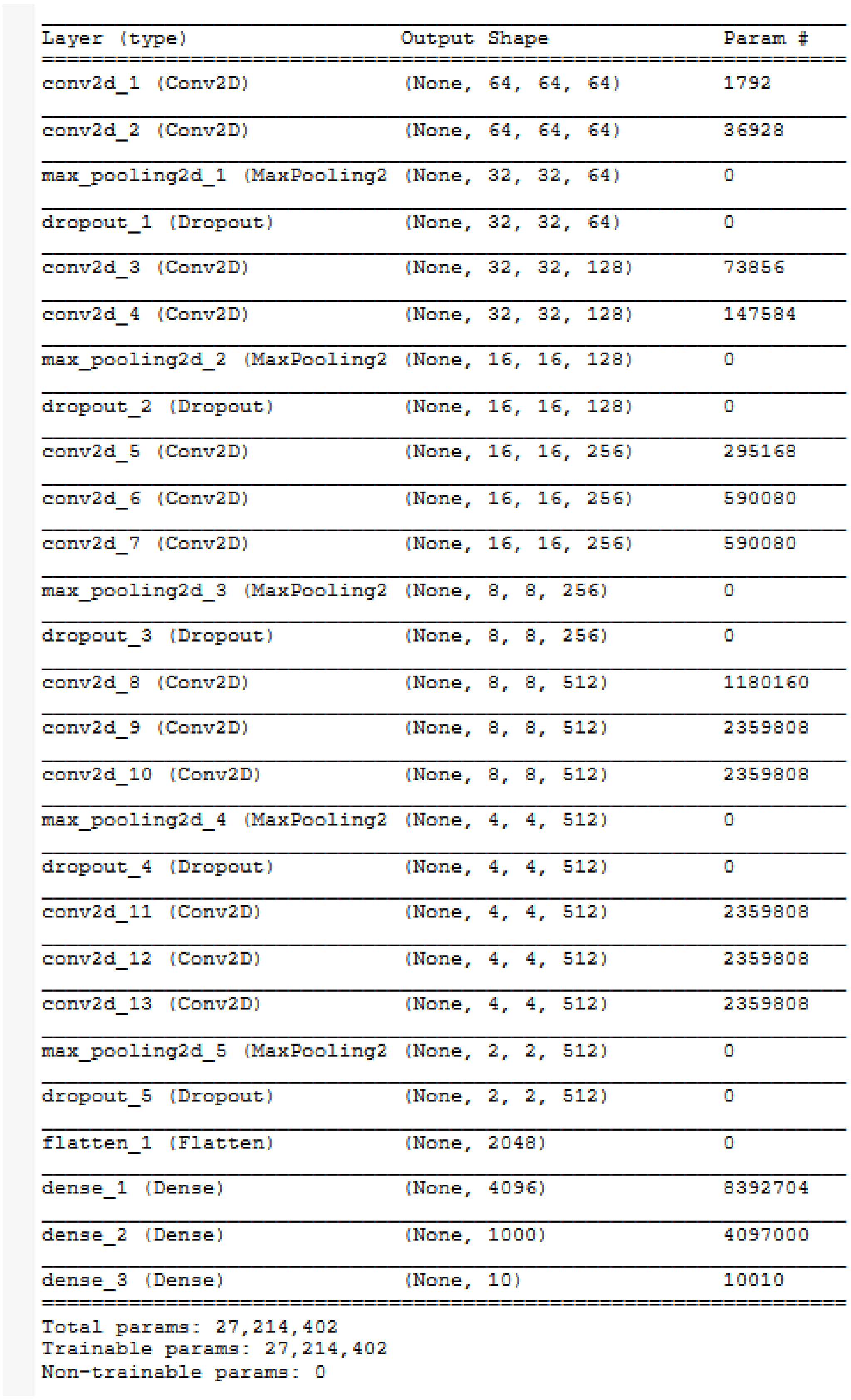


Figure 16: Figure of training versus validation accuracy and loss

2. VGG16 with RmsProp



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 64, 64, 64)        1792
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 64, 64, 64)        36928
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 32, 32, 64)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 32, 32, 64)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 32, 32, 128)       73856
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 32, 32, 128)       147584
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 16, 16, 128)       0
_________________________________________________________________
dropout_2 (Dropout)          (None, 16, 16, 128)       0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 16, 16, 256)       295168
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 16, 16, 256)       590080
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 16, 16, 256)       590080
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 8, 8, 256)         0
_________________________________________________________________
dropout_3 (Dropout)          (None, 8, 8, 256)         0
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 8, 8, 512)         1180160
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 8, 8, 512)         2359808
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 8, 8, 512)         2359808
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 4, 4, 512)         0
_________________________________________________________________
dropout_4 (Dropout)          (None, 4, 4, 512)         0
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 4, 4, 512)         2359808
_________________________________________________________________
conv2d_12 (Conv2D)           (None, 4, 4, 512)         2359808
_________________________________________________________________
conv2d_13 (Conv2D)           (None, 4, 4, 512)         2359808
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 2, 2, 512)         0
_________________________________________________________________
dropout_5 (Dropout)          (None, 2, 2, 512)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 2048)              0
_________________________________________________________________
dense_1 (Dense)              (None, 4096)              8392704
_________________________________________________________________
dense_2 (Dense)              (None, 1000)              4097000
_________________________________________________________________
dense_3 (Dense)              (None, 10)                10010
=================================================================
Total params: 27,214,402
Trainable params: 27,214,402
Non-trainable params: 0
```

Table 3: Figure of VGG16

The block diagram of the model is:

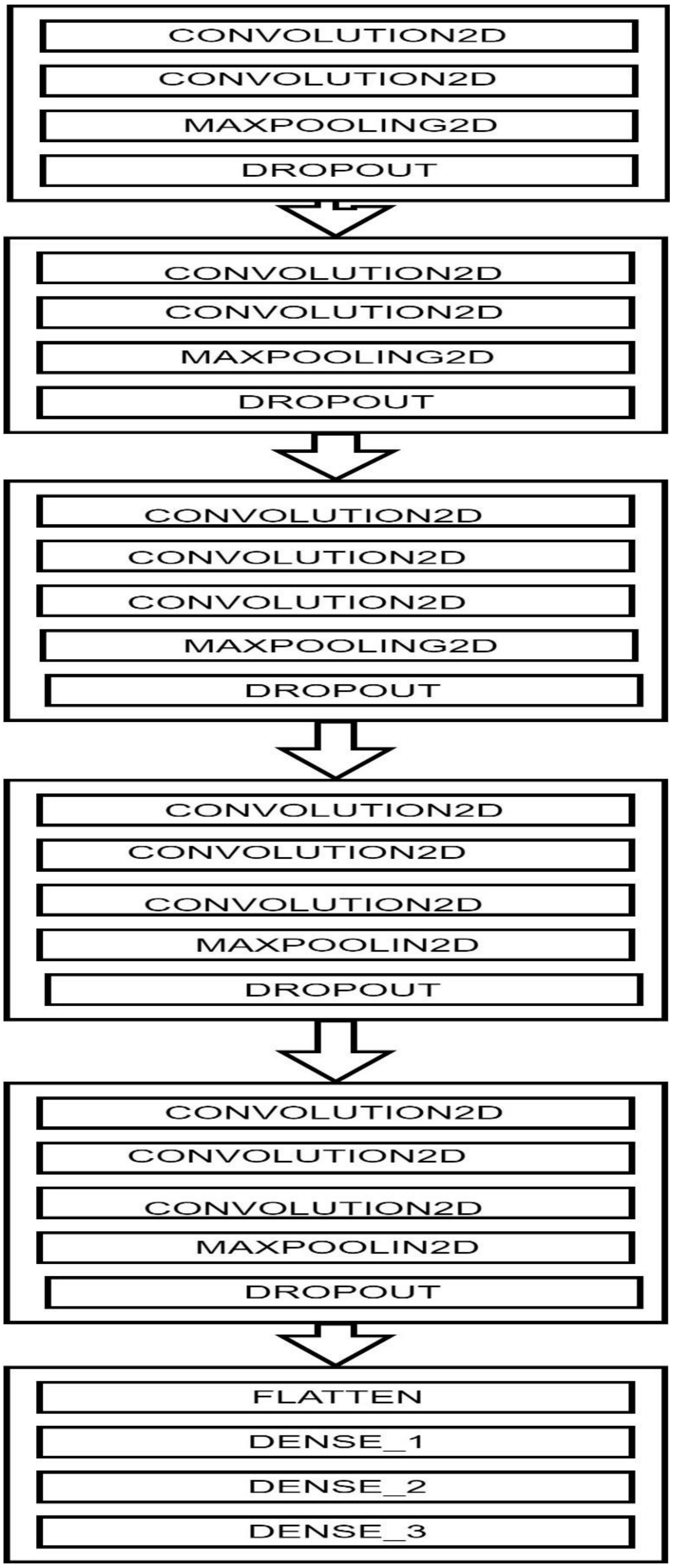
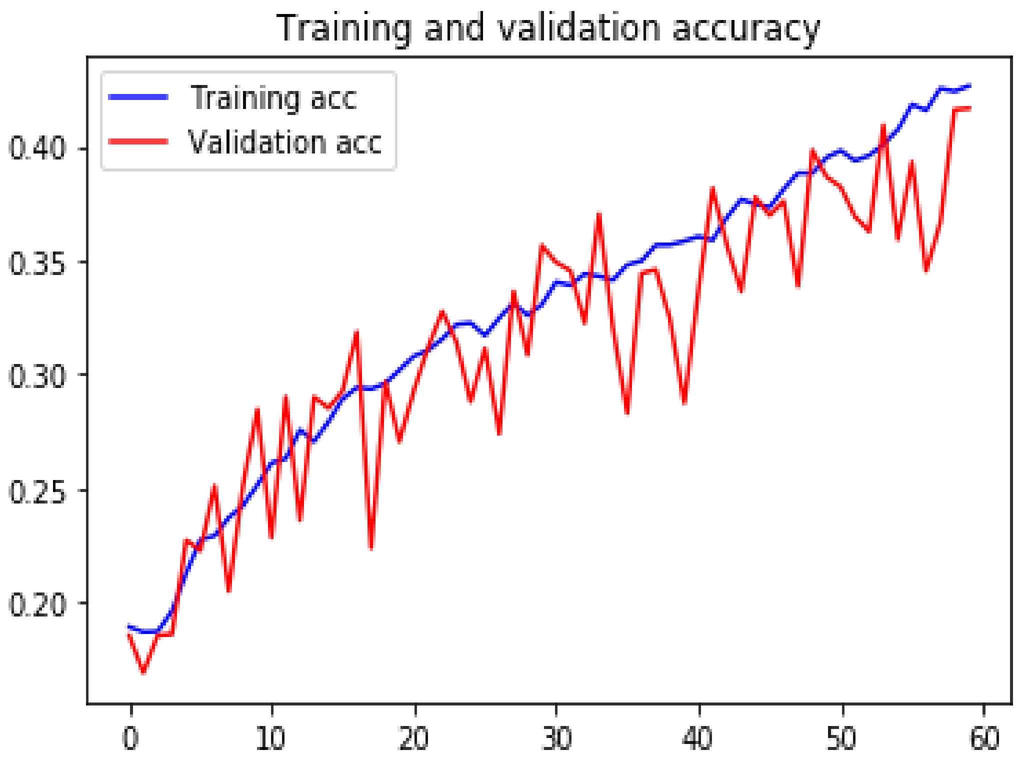


Figure 17: Block diagram of VGG16 model

The training versus validation accuracy and loss so obtained by this architecture is:



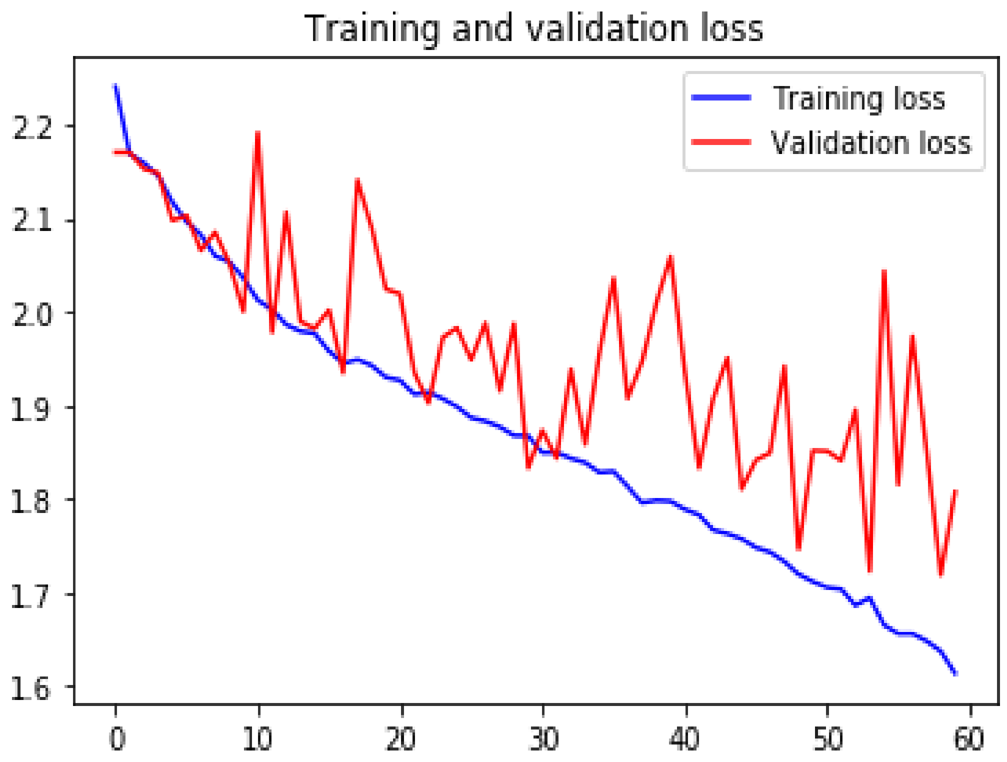


Figure 18: Figure of training versus validation accuracy and loss

3.4 Feature Extraction

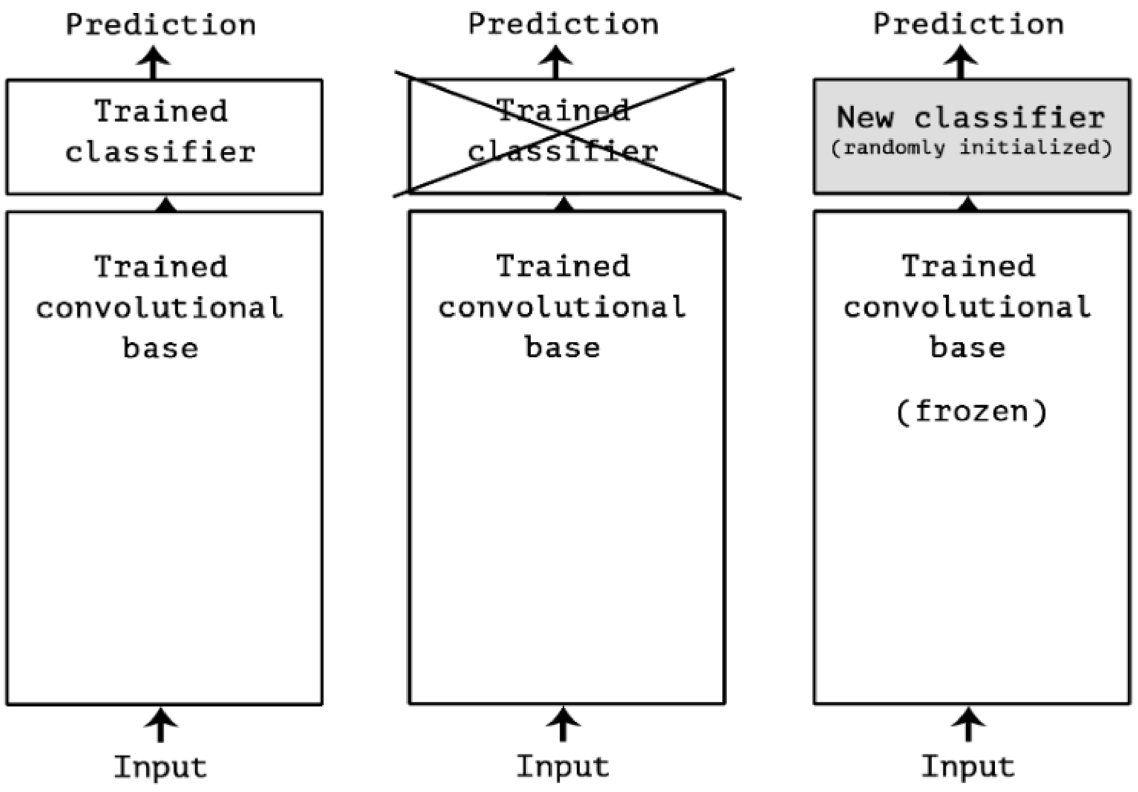
Information extraction from the dataset was the core part of our program and for this we performed training on several models. Some of them were just useless and some of them gave nice result. After many hits and trials, we could sort out for the model which best fitted our training dataset. VGG16 model was the mode which nearly gave best fitting of our dataset. However, to get even better performance we have done tuning by adding the dropout followed by maxpooling at some convolution block and after dense layer to reduce the overfitting due to huge parameters.

3.4.1 Using pre-trained covnet

A commonly and highly effective approach to deep learning on small image datasets is to leverage a pre-trained network. A pre-trained network is simply a saved network previously trained on a large dataset, typically on a large-scale image classification task. If this original dataset is large enough and general enough, then the spatial feature hierarchy learned by the pre-trained network can effectively act as a generic model of our visual world, and hence its features can prove useful for many different computer vision problems, even though these new problems might involve completely different classes from those of the original task. For instance, one might train a network on ImageNet (where classes are mostly animals and everyday objects) and then re-purpose this trained network for something as remote as identifying furniture items in images. Such portability of learned features across different problems is a key advantage of deep learning compared to many older shallow learning approaches, and it makes deep learning very effective for small-data problems.

Feature extraction consists of using the representations learned by a previous network to extract interesting features from new samples. These features are then run through a new classifier, which is trained from scratch.

As we saw previously, convnets used for image classification comprise two parts: they start with a series of pooling and convolution layers, and they end with a densely-connected classifier. The first part is called the "convolutional base" of the model. In the case of convnets, "feature extraction" will simply consist of taking the convolutional base of a previously-trained network, running the new data through it, and training a new classifier on top of the output.



```
from keras.applications import VGG16

pre_trained = VGG16(weights='imagenet',
                    include_top=False,
                    input_shape=(150, 150, 3))
```

We passed three arguments to the constructor:

- weights, to specify which weight checkpoint to initialize the model from
- include_top, which refers to including or not the densely-connected classifier on top of the network. By default, this densely-connected classifier would correspond to the 1000 classes from ImageNet. Since we intend to use our own densely-connected classifier (with only two classes, cat and dog), we don't need to include it.
- input_shape, the shape of the image tensors that we will feed to the network. This argument is purely optional: if we don't pass it, then the network will be able to process inputs of any size.

for layer in pre_trained_model.layers[:11]:

  layer.trainable = False.

Here, we freezed the upper layer upto 11 to make the weights value unchanged.

```
model = models.Sequential()
model.add(pre_trained)
model.add(layers.Flatten())
model.add(layers.Dense(512, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))
```

Parameters before freezing the layers
Total params: 14,714,688
Trainable params: 14,714,688
Non-trainable params: 0

Parameters after Freezing the layers
Total params: 14,714,688
Trainable params: 12,979,200

Non-trainable params: 1,735,488

we then trained the model by adding the on the top the pretrained base model.

Layer (type)	Output Shape	Param #
vgg16 (Model)	(None, 4, 4, 512)	14714688
flatten_2 (Flatten)	(None, 8192)	0
dense_3 (Dense)	(None, 512)	4194816
dense_4 (Dense)	(None, 10)	5130

Parameters after Freezing the layers and adding on the top of the layers

Total params: 18,914,634
Trainable params: 17,179,146
Non-trainable params: 1,735,488

Tools and TechnologiesWe used different tools and technologies to complete our project. Different tools were used for project management, version control management, administrative tool for database, etc. Similarly, technologies for dataset collection, training, user interface development were used. Different python libraries were used for information extraction techniques, machine learning, web scrapping, etc.Extending the model we have (conv_base) by adding Dense layers on top, and running the whole thing end-to-end on the input data. This allows us to use data augmentation, because every input image is going through the convolutional base every time it is seen by the model. However, for this same reason, this technique is far more expensive than the first one.

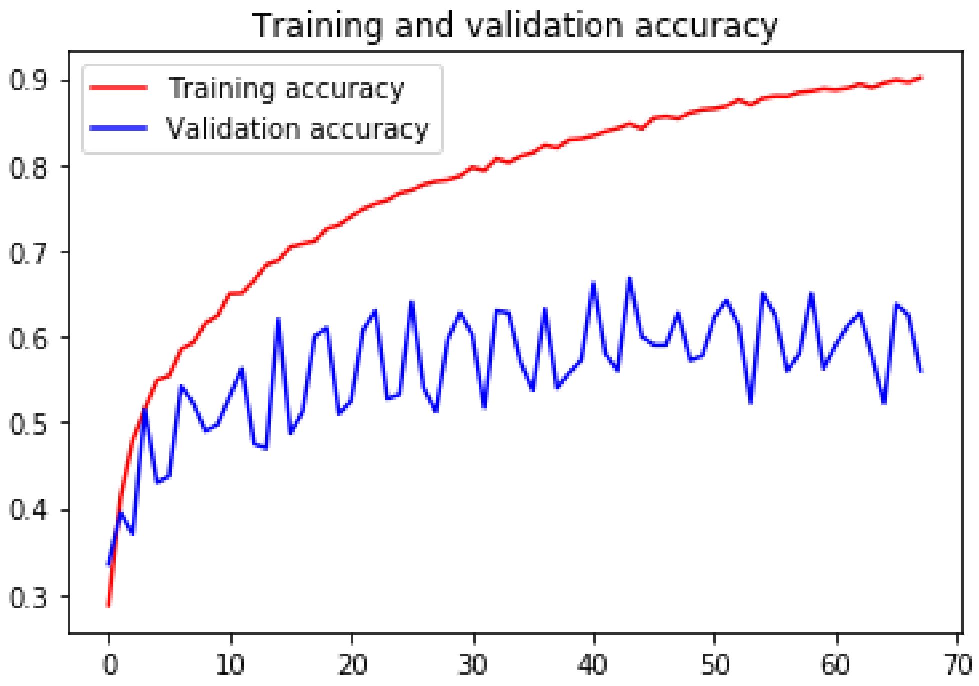


Figure 19:Training vs validation accuracy after freezing

3.4.2 Fine tuning

Another widely used technique for model reuse, complementary to feature extraction, is fine-tuning. Fine-tuning consists in unfreezing a few of the top layers of a frozen model base used for feature extraction, and jointly training both the newly added part of the model (in our case, the fully-connected classifier) and these top layers. This is called "fine-tuning" because it slightly adjusts the more abstract representations of the model being reused, in order to make them more relevant for the problem at hand.

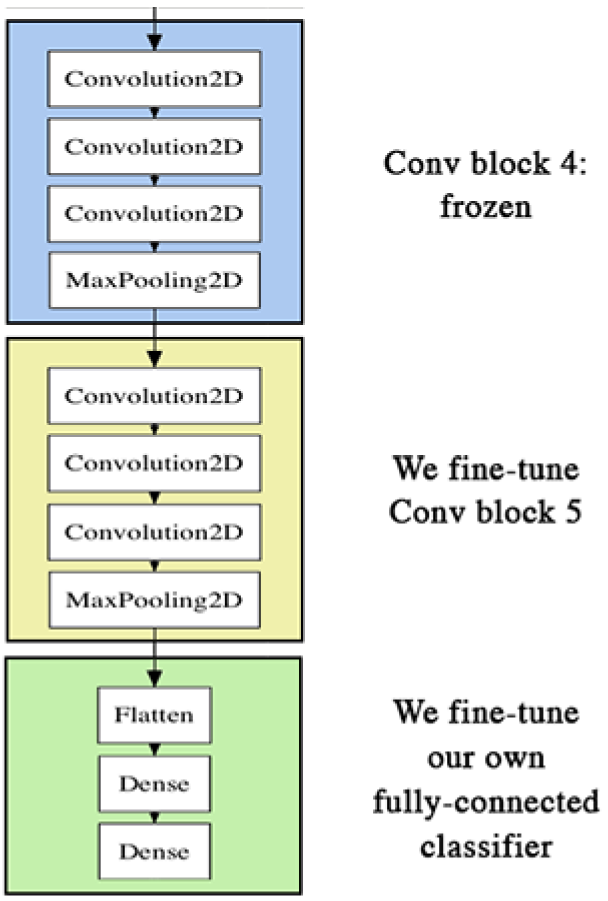


Figure 20:Model of fine Tuning

We have stated before that it was necessary to freeze the convolution base of VGG16 in order to be able to train a randomly initialized classifier on top. For the same reason, it is only possible to fine-tune the top layers of the convolutional base once the classifier on top has already been trained. If the classified wasn't already trained, then the error signal propagating through the network during training would be too large, and the representations previously learned by the layers being fine-tuned would be destroyed. Thus the steps for fine-tuning a network are as follow:

1) Add your custom network on top of an already trained base network.
2) Freeze the base network.
3) Train the part you added.
4) Unfreeze some layers in the base network.
5) Jointly train both these layers and the part you added.

We have already completed the first 3 steps when doing feature extraction. Let's proceed with the 4th step: we will unfreeze our conv_base, and then freeze individual layers inside of it.

Some of the tools and technologies used are briefly described below.

```
conv_base.trainable = True

set_trainable = False
for layer in conv_base.layers:
    if layer.name == 'block5_conv1':
        set_trainable = True
    if set_trainable:
        layer.trainable = True
    else:
        layer.trainable = False
```

Now we can start fine-tuning our network. We will do this with the RMSprop optimizer, using a very low learning rate. The reason for using a low learning rate is that we want to limit the magnitude of the modifications we make to the representations of the 3 layers that we are fine-tuning. Updates that are too large may harm these representations.

```
model.compile(loss='binary_crossentropy', optimizer=optimizers.RMSprop(lr=00000.3), metrics=['acc'])
```

3.5. Classification

The final stage where the actual information to the user is obtained through this classification stage. The feature extractions are done after applying to different layers of deep learning. Here based on the features extracted from the image is match to the actual applied image to classify.

3.6 Analysis of training output:

After training various models and analyzing the outputs we finalize to work with VGG16 model with rmsprop optimizer. After this we started modifying various parameters and tried to fine tune the parameters with hyperparameter tuning techniques.

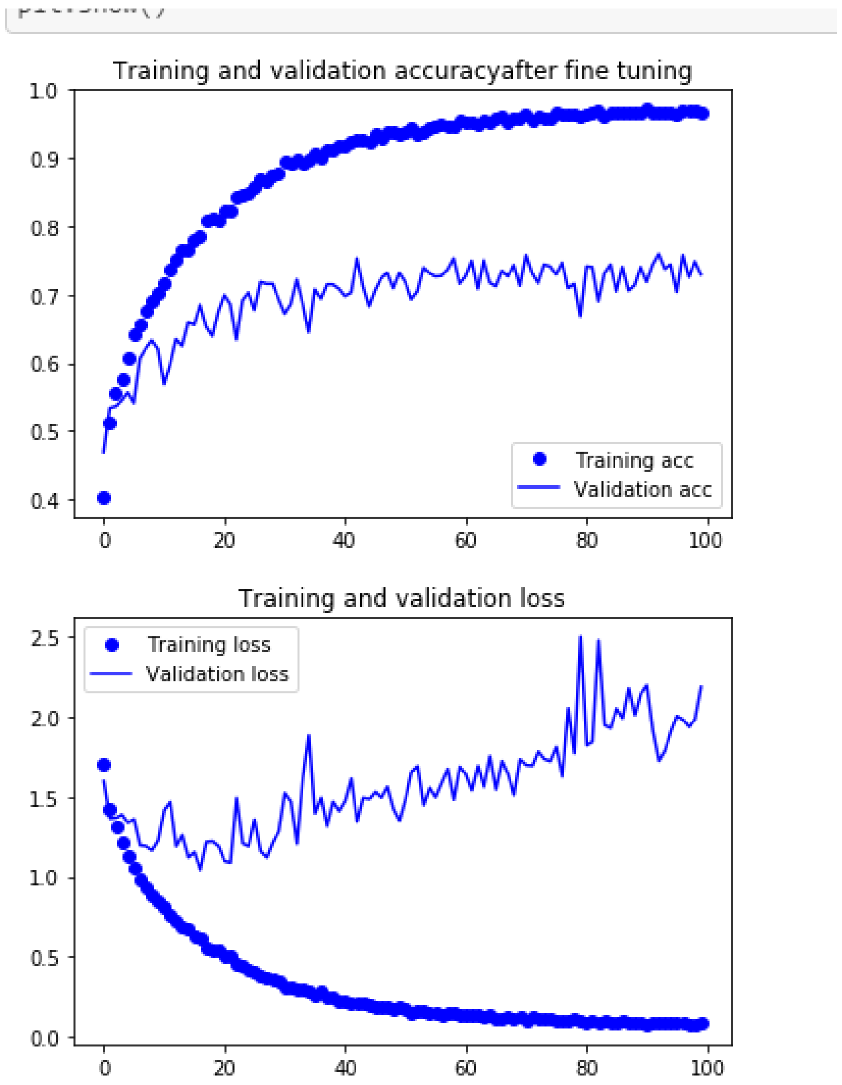
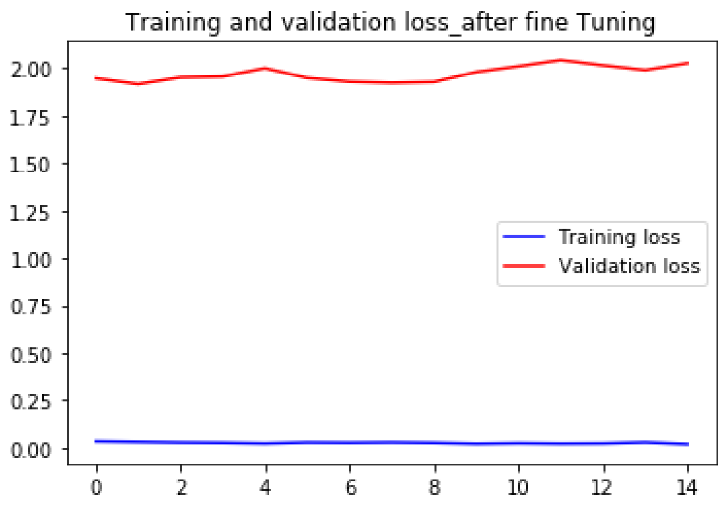


Figure 21:Training vs validation accuracy and loss after freezing

```
4400 images belonging to 10 classes.
1100 images belonging to 10 classes.
Epoch 1/100
69/68 [==============================] - 53s 762ms/step loss: 1.7023 - acc: 0.4040 - val_loss:
1.5960 - val_acc: 0.4691
Epoch 10/100
69/68 [==============================] - 53s 763ms/step - loss: 0.8505 - acc: 0.7009 - val_loss:
1.2272 - val_acc: 0.6200
Epoch 20/100
69/68 [==============================] - 52s 757ms/step - loss: 0.5402 - acc: 0.8089 - val_loss:
1.1914 - val_acc: 0.6755
Epoch 30/100
```

```
69/68 [==============================] - 53s 764ms/step - loss: 0.3502 - acc: 0.8776 - val_loss: 1.2829 - val_acc: 0.6927
Epoch 40/100
69/68 [==============================] - 52s 748ms/step - loss: 0.2303 - acc: 0.9181 - val_loss: 1.4104 - val_acc: 0.7082
Epoch 50/100
69/68 [==============================] - 52s 760ms/step - loss: 0.1868 - acc: 0.9355 - val_loss: 1.3493 - val_acc: 0.7318
Epoch 60/100
69/68 [==============================] - 52s 751ms/step - loss: 0.1322 - acc: 0.9561 - val_loss: 1.6878 - val_acc: 0.7155
Epoch 70/100
69/68 [==============================] - 52s 759ms/step - loss: 0.1231 - acc: 0.9565 - val_loss: 1.7333 - val_acc: 0.7127
Epoch 80/100
69/68 [==============================] - 52s 748ms/step - loss: 0.1004 - acc: 0.9598 - val_loss: 2.4983 - val_acc: 0.6682
Epoch 90/100
69/68 [==============================] - 52s 752ms/step - loss: 0.0894 - acc: 0.9672 - val_loss: 2.1429 - val_acc: 0.7400
Epoch 100/100
69/68 [==============================] - 52s 755ms/step - loss: 0.0863 - acc: 0.9650 - val_loss: 2.1843 - val_acc: 0.7291
```

After fine Tuning of the model:



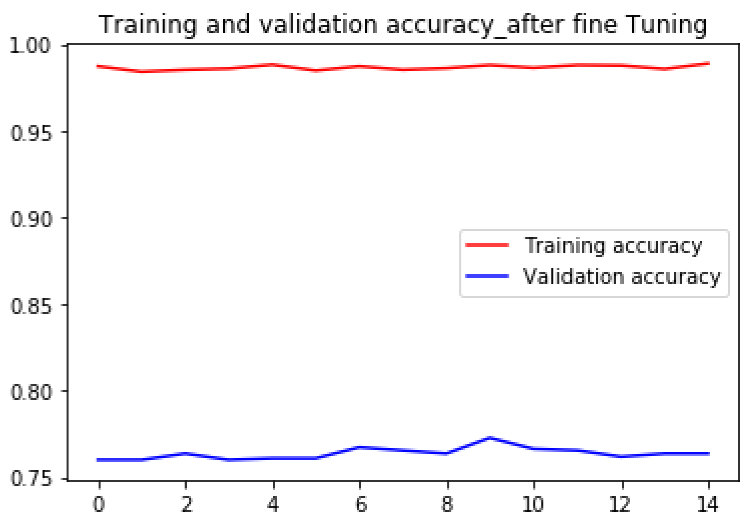


Figure 22: Training and validation accuracy and loss after fine Tuning

```
Epoch 1/50
69/68 [==============================] - 55s 798ms/step - loss: 0.0346 - acc: 0.9872 - val_loss
: 1.9476 - val_acc: 0.7600
Epoch 2/50
69/68 [==============================] - 51s 739ms/step - loss: 0.0312 - acc: 0.9841 - val_loss
: 1.9173 - val_acc: 0.7600
Epoch 3/50
69/68 [==============================] - 51s 735ms/step - loss: 0.0282 - acc: 0.9853 - val_loss
: 1.9524 - val_acc: 0.7636
Epoch 4/50
69/68 [==============================] - 51s 740ms/step - loss: 0.0266 - acc: 0.9859 - val_loss
: 1.9566 - val_acc: 0.7600
Epoch 5/50
69/68 [==============================] - 51s 734ms/step - loss: 0.0232 - acc: 0.9882 - val_loss
: 1.9999 - val_acc: 0.7609
Epoch 6/50
69/68 [==============================] - 51s 733ms/step - loss: 0.0287 - acc: 0.9846 - val_loss
: 1.9496 - val_acc: 0.7609
Epoch 7/50
69/68 [==============================] - 51s 734ms/step - loss: 0.0271 - acc: 0.9873 - val_loss
: 1.9298 - val_acc: 0.7673
Epoch 8/50
```

```
69/68 [==============================] - 51s 734ms/step - loss: 0.0287 - acc: 0.9853 - val_loss
: 1.9243 - val_acc: 0.7655
Epoch 9/50
69/68 [==============================] - 50s 731ms/step - loss: 0.0264 - acc: 0.9862 - val_loss
: 1.9285 - val_acc: 0.7636
Epoch 10/50
69/68 [==============================] - 51s 740ms/step - loss: 0.0217 - acc: 0.9880 - val_loss
: 1.9800 - val_acc: 0.7727
Epoch 11/50
69/68 [==============================] - 50s 729ms/step - loss: 0.0245 - acc: 0.9863 - val_loss
: 2.0100 - val_acc: 0.7664
Epoch 12/50
69/68 [==============================] - 50s 731ms/step - loss: 0.0224 - acc: 0.9880 - val_loss
: 2.0424 - val_acc: 0.7655
Epoch 13/50
69/68 [==============================] - 51s 734ms/step - loss: 0.0235 - acc: 0.9877 - val_loss
: 2.0150 - val_acc: 0.7618
Epoch 14/50
69/68 [==============================] - 51s 737ms/step - loss: 0.0289 - acc: 0.9857 - val_loss
: 1.9903 - val_acc: 0.7636
Epoch 15/50
69/68 [==============================] - 52s 748ms/step - loss: 0.0197 - acc: 0.9889 - val_loss
: 2.0260 - val_acc: 0.7636
```

Reached >**98.850%** accuracy so cancelling training!

3.7 Tools and Technologies

We used different tools and technologies to complete our project. Different tools were used for project management, version control management, administrative tool for database, etc. Similarly, technologies for dataset collection, training, user interface development were used. Different python libraries were used for information extraction techniques, machine learning, web scrapping, etc.

Some of the tools and technologies used are briefly described below.

Google Spreadsheet:

It was used as a project management tool. Responsibility of individual team members were assigned in the assignment sheet. We used it to keep track of the completed task and the pending tasks.

GitHub:

GitHub was used for distributed version control and source code management. We created a public repository and updated our source code there.

Android studio:

The mobile interface is being created in android studio and the task is under progress.

Jupyter notebook:

Jupyter notebook was used inside virtual environment in order to perform all the task relate to coding the training and image preprocessing.

CUDA Toolkit, cuDNN:

We used NVDIA GETFORCE GTX gpu accelerated laptop to train our model.

The NVIDIA® CUDA® Toolkit provides a development environment for creating high performance GPU-accelerated applications. With the CUDA Toolkit, you can develop, optimize and deploy your applications on GPU-accelerated embedded systems, desktop workstations, enterprise data centers, cloud-based platforms and HPC supercomputers. The toolkit includes GPU-accelerated libraries, debugging and optimization tools, a C/C++ compiler and a runtime library to deploy your application.

The NVIDIA CUDA® Deep Neural Network library (cuDNN) is a GPU-accelerated library of primitives for deep neural networks. CuDNN provides highly tuned implementations for standard routines such as forward and backward convolution, pooling, normalization, and activation layers.

Google collab:

Since all of the team members didn’t have gpu supported laptop, rest of them trained and performed the model testing in google collab itself.

3.8 software application in project:

3.8.1 Android Studio

Android Studio is the official integrated development environment for Google's Android operating system, built on JetBrains' IntelliJ IDEA software and designed specifically for Android development. We plan to make the android application using android studio so as to make the app more user friendly by simply deploying the deep learning model on this app.

Regarding the android application that we have planned to integrate with our major project for the better portability and easiness. We started the work for the UI dated where we started with the stuffs like knowing about the existence of similar application and its offered option. Firstly, we decided to work on the Java platform but had problem in the installation of android Studio IDE so after a lot of research for the application platform and more reliable source for the completion of task we changed our previous option and then started to work on Flutter which is a Google's portable UI toolkit for building beautiful, native applications for mobile, web, and desktop from a single codebase. For the IDE we first thought of using Visual Studio Code but finally switched to android Studio again as for the programming language we have decided to work with Dart programming language, which is a client-optimized programming language for fast apps on multiple platforms. It is developed by Google and is used to build mobile, desktop, backend and web applications. Thus, we completed the initial planning and requirement analysis phase.

Moreover, we moved to the next phase i.e. design phase but before starting we began with studying the tutorials and learning to use application required for the further process. For the tutorials we followed various videos and official website which are mentioned in the reference part. Furthermore, for the wire framing of application, we used Adobe XD windows application and started designing the welcome page, homepage and toggle bar on the homepage at the started of eighth semester and still we are working on it. The design will be refined in the days to come according to the requirement if needed. Currently, we are now working on the initial development phase of UI and it's a new platform for us initial phase is taking some time.

3.8.1 Python

Python is a widely used general-purpose, high-level programming language. Its design philosophy emphasizes code readability, and its syntax allows programmers to express concepts in fewer lines of code than would be possible in languages such as C++ or Java. The language provides constructs intended to enable clear programs on both a small and large scale. Python supports multiple programming paradigms, including object oriented, imperative and functional programming or procedural styles. It features a dynamic type system and automatic memory management and has a large and comprehensive standard library.

Convolutional Neural network (CNN) used for our problem. For the implementation of CNN, we have used Keras library as TensorFlow as backend. The language used is python. The software used to code the program was Jupiter notebook and the interpreter used is of python 3.6 version.

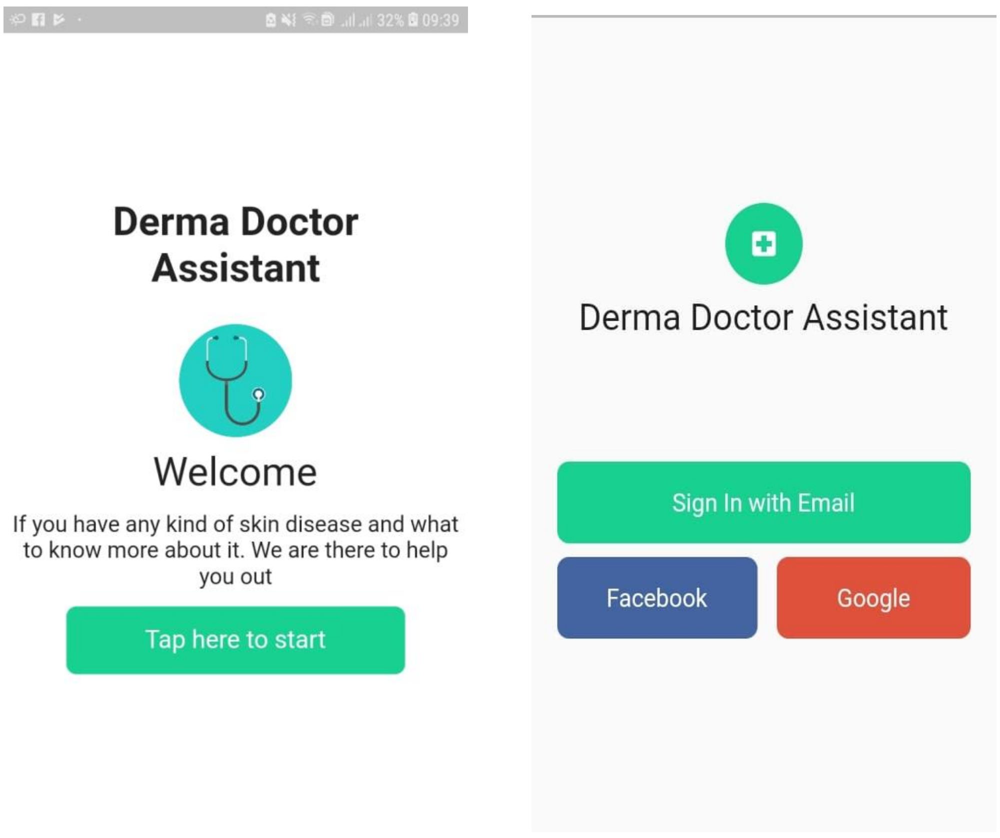
3.8.2 NumPy:

NumPy is the fundamental package for scientific computing with Python. It contains among other things:

A powerful N-dimensional array object Sophisticated (broadcasting) functions Tools for integrating Visual C/C++. Useful linear algebra, Fourier transform, and random number capabilities. Besides its obvious scientific uses, NumPy can also be used as an efficient multi-dimensional container of generic data. Arbitrary data-types can be defined. This allows NumPy to seamlessly and speedily integrate with a wide variety of databases.

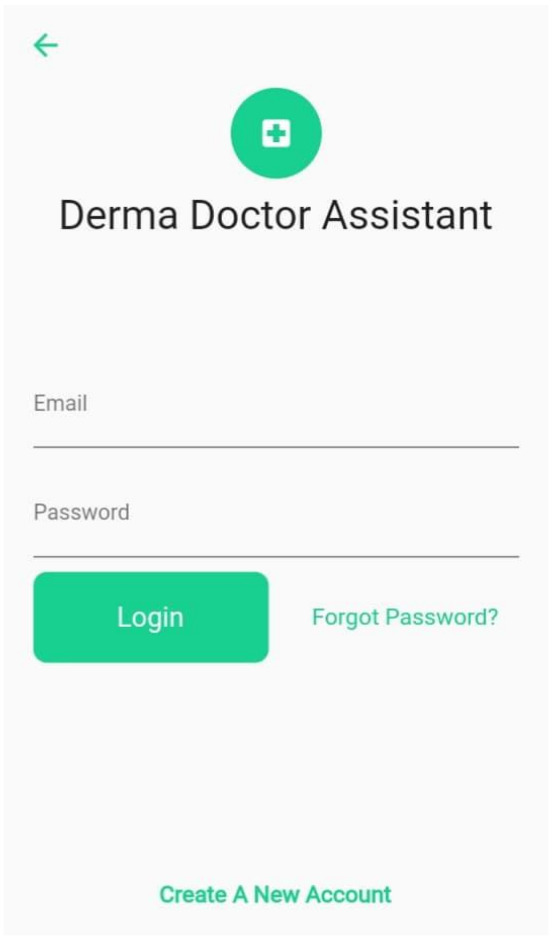
4. OUTPUT

The user interface that we have made till now are:



If you have any kind of skin disease and what to know more about it. We are there to help you out

Figure 23: Welcome page of UI



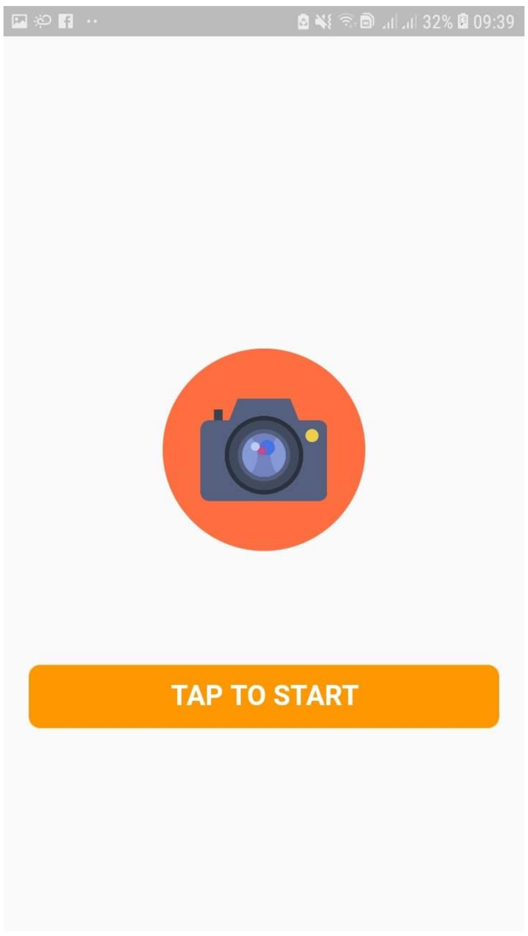
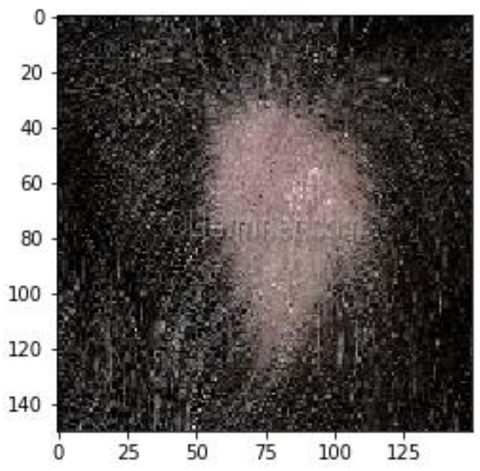


Figure 24:Login and Camera access page

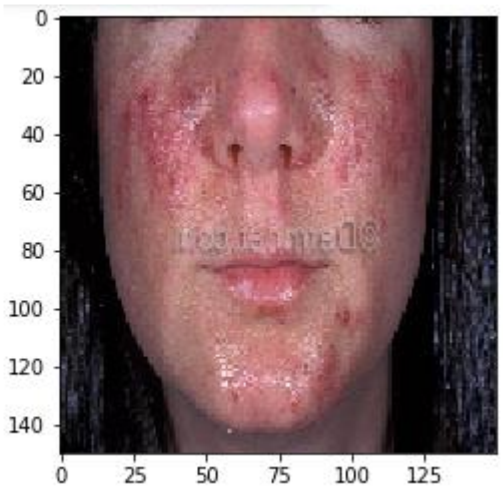
The final stage where the actual information to the user is obtained through the classification stage. The feature extractions are done after applying to different layers of deep learning. Here based on the features extracted from the image is match to the actual applied image to classify.

```
CATEGORIES =["Acne-Rosacea", "Basal cell carcinoma"," Hair Loss Alopecia and other Hair Diseases",
          " Herpes"," Melanoma Skin Cancer Nevi and Moles",
         " Nail Fungus and other Nail Disease" , "Urticaria Hives"," Vasculitis"," Warts","seborrh eic-keratoses"]
```

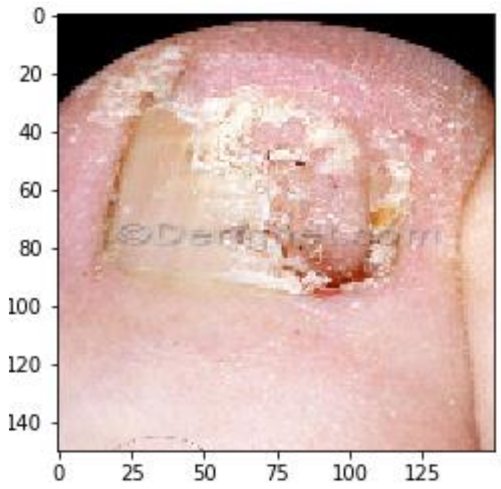
```
Choose Files  No file chosen  Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving alopecia-areata-17.jpg to alopecia-areata-17 (4).jpg
[0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]
The predicted disease found to be Hair Loss Alopecia and other Hair Diseases.
```



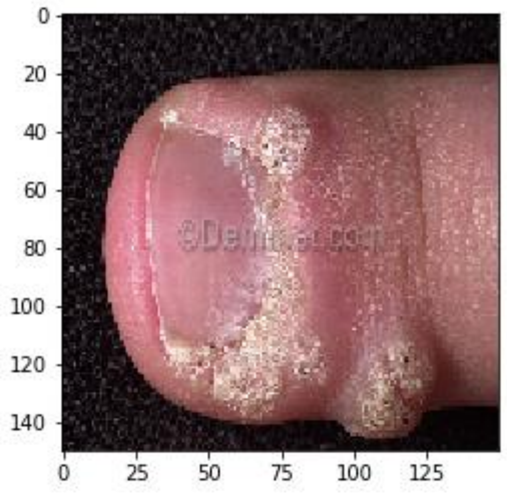
```
Saving acne-cystic-121f.jpg to acne-cystic-121f (4).jpg
[1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]

The predicted disease found to be Acne-Rosacea
```



```
Saving biting-excoriation-14.jpg to biting-excoriation-14.jpg

[0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 1.193549e-34
 1.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00]

The predicted disease found to be Nail Fungus and others Nail Disease
```



```
Saving 12PeriungualWart.jpg to 12PeriungualWart.jpg

[1.8924947e-33 0.0000000e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00
 0.0000000e+00 0.0000000e+00 0.0000000e+00 1.0000000e+00 0.0000000e+00]



The predicted disease found to be Warts
```



5. CONCLUSION

To conclude our work done till date, we remained busy collecting dataset and training them to find the best fit. Also, we have been working to increase the percent of accuracy. We have created different model to fit our project and to predict the more precise class or result. After testing many model the VGG16 model with the Rmsprop optimizer gives the best result and hence we conclude to use the very model in our project. As we need to increase the accuracy we simply perform the transfer learning where we did freezing certain pre trained portion of model and training the remaining part of model. This yields the better accuracy and is able to predict the result more precisely. Similarly we have also done image augmentation to make our data set more so that it will be sufficient to train our model with various image.

We have used Android Studio for making our user interface by installing flutter package. As the flutter provides many widgets which are easy to implement we decided to use it and designed the welcome, login and camera access page. The camera access page allow us to access our cell phone camera and the capture the image. The data base part is to be done along with the model deploy in cell phone using tensor flow lite. We planned this task to be done within a week.

6. REFERENCES

1. *"Efstratios Gavves"A Brief History of Computer Vision," 2014.*
2. *M.H. Yang (2003). "Object recognition" [online].*
3. *https://adeshpande3.github.io/adeshpande3.github.io/The-9-Deep-Learning-Papers-You-Need-To-Know-About.html[online]*
4. *https://www.youtube.com/watch?v=m8pOnJxOcqY&t=19s[media]*
5. *https://www.youtube.com/watch?v=FTr3n7uBIuE [media]*
6. *https://adeshpande3.github.io/adeshpande3.github.io/ [online].*
7. *https://github.com/llSourcell/Convolutional_neural_network [online].*
8. *http://www.deeplearningbook.org[online]*
9. *https://cambridgespark.com/content/tutorials/convolutional-neural-networks-with-keras[online]*
10. *3. https://www.tensorflow.org/tutorials/ [online].*
11. *4. https://github.com/llSourcell/Convolutional_neural_network [online].*
12. *5. https://www.tensorflow.org/tutorials/estimators/cnn[online].*
13. *6. https://www.coursera.org/learn/machine-learning[online].*
14. *7. https://www.coursera.org/specializations/deep-learning[online].*
15. *ager T. (2017). 8 effective and useful data visualization tools for mapping.*

Retrieved from: http://bigdata-madesimple.com/8-effective-and-useful-data-visualization-tools-for-mapping/

16. Beautiful soup. Retrieved from: https://wiki.python.org/moin/beautiful%20soup