CHAPTER 1

INTRODUCTION

Skin cancer, the most common human malignancy, is primarily diagnosed visually, beginning with an initial clinical screening and followed potentially by dermoscopic analysis, a biopsy and histopathological examination. Automated classification of skin lesions using images is a challenging task owing to the fine-grained variability in the appearance of skin lesions. Deep convolutional neural networks (CNNs) show potential for general and highly variable tasks across many fine-grained object categories.

There are 5.4 million new cases of skin cancer in the United States every year. One in five Americans will be diagnosed with a cutaneous malignancy in their lifetime. Although melanomas represent fewer than 5% of all skin cancers in the United States, they account for approximately 75% of all skin-cancer-related deaths, and are responsible for over 10,000 deaths annually in the United States alone. Early detection is critical, as the estimated 5-year survival rate for melanoma drops from over 99% if detected in its earliest stages to about 14% if detected in its latest stages. We developed a computational method which may allow medical practitioners and patients to proactively track skin lesions and detect cancer earlier. By creating a novel disease taxonomy, and a disease-partitioning algorithm that maps individual diseases into training classes, we are able to build a deep learning system for automated dermatology.

The skin lesions are categorized into two classes:

1): Melanocytic (melanoma)

2): Non- melanocytic (nevus and seborrheic keratosis)

1.1 Machine Learning

Machine learning is a subset of artificial intelligence in the field of computer science that often uses statistical techniques to give computers the ability to “learn” (i.e., progressively improve performance on a specific task) with data, without being explicitly programmed every single time. [Wikipedia https://en.wikipedia.org/wiki/Machine_learning]

The name *machine learning* was coined in 1959 by Arthur Samuel. Evolved from the study of pattern recognition and computational learning theory in artificial intelligence, machine learning explores the study and construction of algorithms that can learn from and make predictions on data- such algorithms overcome following strictly static program instructions by making data-driven predictions or decisions through building a model from sample inputs.

1.2 Convolution Neural Network

Convolutional Neural Network is a class of deep neural network that is used for Computer Vision or analyzing visual imagery. Convolutional networks were inspired by biological processes in that the connectivity pattern between neurons resembles the organization of the animal visual cortex.

A CNN consists of an input and an output layer, as well as multiple hidden layers. The hidden layers of a CNN typically consist of convolutional layers, pooling layers, fully connected layers and normalization layers.

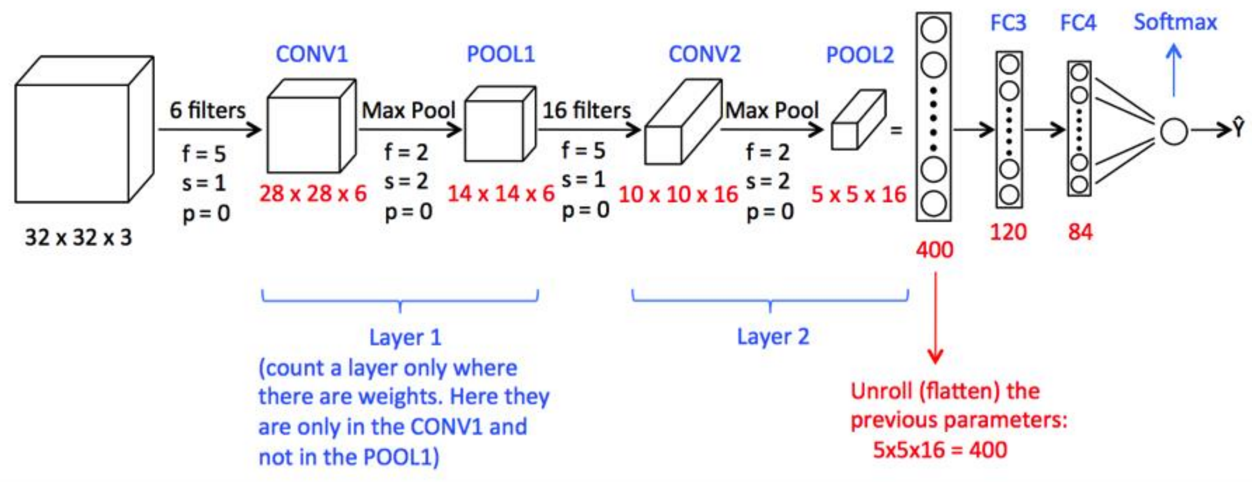


Fig: 2 Convolution neural network process

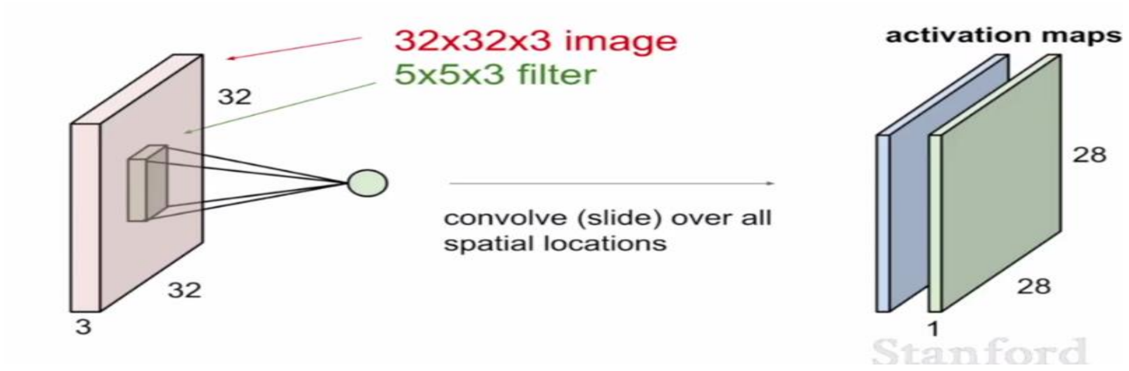
1.2.1 Convolution Layer

Computers read images as pixels and it is expressed as matrix (NxNx3)—(height by width by depth). Images makes use of three channels (RGB), so that is why we have a depth of 3.

The Convolutional Layer makes use of a set of learnable filters. A filter is used to detect the presence of specific features or patterns present in the original image (input). It is usually expressed as a matrix (MxMx3), with a smaller dimension but the same depth as the input file.

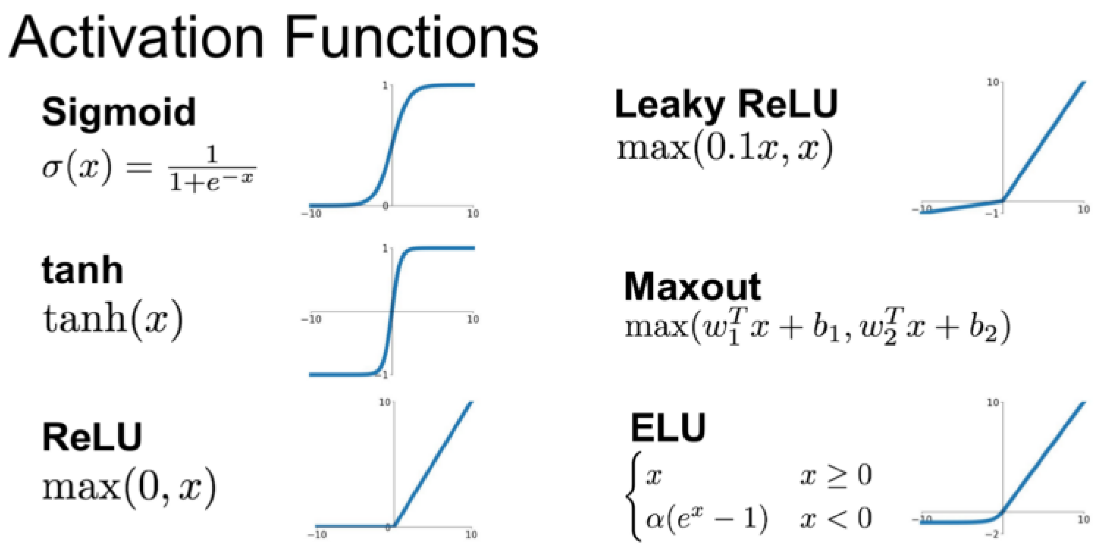
This filter is convolved (slided) across the width and height of the input file, and a dot product is computed to give an activation map.

Different filters which detect different features are convolved on the input file and a set of activation maps is outputted which is passed to the next layer in the CNN.



1.2.2 Activation Function

Activation function is a node that is put at the end of or in between Neural Networks. They help to decide if the neuron would fire or not.



ReLU function is the most widely used activation function in neural networks today. One of the greatest advantages ReLU has over other activation functions is that it does not activate all neurons at the same time. From the image for ReLU function above, we'll notice that it converts all negative inputs to zero and the neuron does not get activated. This makes it very computational efficient as few neurons are activated per time. It does not saturate at the positive region. In practice, ReLU converges six times faster than tanh and sigmoid activation functions.

Some disadvantage ReLU presents is that it is saturated at the negative region, meaning that the gradient at that region is zero. With the gradient equal to zero, during back propagation all the weights will not be updated, to fix this, we use **Leaky ReLU**. Also, ReLU functions are not zero-centered. This means that for it to get to its optimal point, it will have to use a zig-zag path which may be longer.

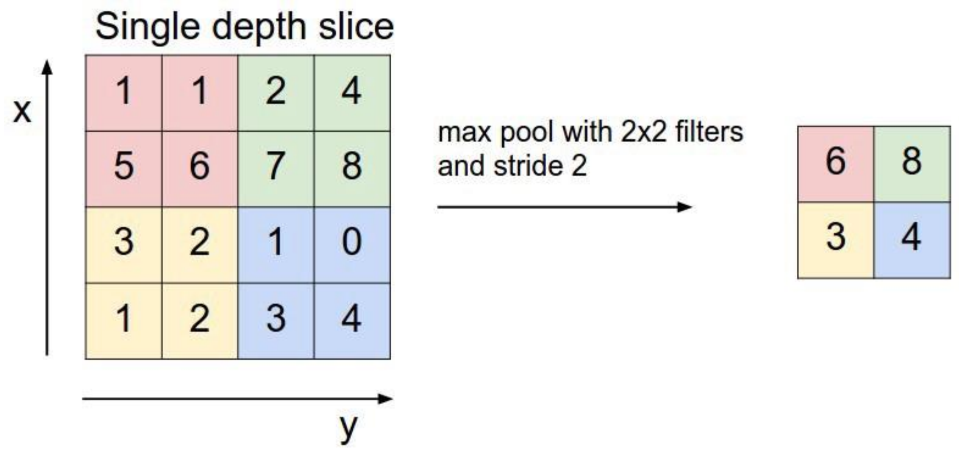
1.2.3 Pooling Layer

The Pooling layer can be seen between Convolution layers in a CNN architecture. This layer basically reduces the amount of parameters and computation in the network, controlling overfitting by progressively reducing the spatial size of the network.

There are two operations in this layer; Average pooling and Maximum pooling.

1.2.3.1 Max-pooling, like the name states; will take out only the maximum from a pool. This is actually done with the use of filters sliding through the input; and at every stride, the maximum parameter is taken out and the rest is dropped. This actually down-samples the network.

Unlike the convolution layer, the pooling layer does not alter the depth of the network, the depth dimension remains unchanged.



1.2.5 Fully-Connected Layer

In this layer, the neurons have complete connection to all the activations from the previous layers. Their activations can hence be computed with a matrix multiplication followed by a bias offset. This is the last phase for a CNN network.

The Convolutional Neural Network is actually made up of **hidden layers** and **fully-connected layer(s)**.

1.2.6 Dropout

Simply put, dropout refers to ignoring units (i.e. neurons) during the training phase of certain set of neurons which is chosen at random. By “ignoring”, I mean these units are not considered during a particular forward or backward pass.

More technically, At each training stage, individual nodes are either dropped out of the net with probability *1-p* or kept with probability *p*, so that a reduced network is left; incoming and outgoing edges to a dropped-out node are also removed.

A fully connected layer occupies most of the parameters, and hence, neurons develop co-dependency amongst each other during training which curbs the individual power of each neuron leading to over-fitting of training data.
That's why we need dropout layer.

1.2.7 Batch Normalization

Batch normalization reduces the amount by what the hidden unit values shift around (covariance shift) and keep the output value of activation function around the center.

Also, batch normalization allows each layer of a network to learn by itself a little bit more independently of other layers.

We can use higher learning rates because batch normalization makes sure that there’s no activation that’s gone really high or really low.

It reduces overfitting because it has a slight regularization effects. Similar to dropout, it adds some noise to each hidden layer’s activations.

1.2.8 Back Propagation

Backpropagation is a method used in artificial neural networks to calculate a gradient that is needed in the calculation of the weights to be used in the network. Backpropagation is shorthand for "the backward propagation of errors," since an error is computed at the output and distributed backwards throughout the network’s layers. It is commonly used to train deep neural networks, a term referring to neural networks with more than one hidden layer.

Backpropagation is a special case of a more general technique called automatic differentiation. In the context of learning, backpropagation is commonly used by the gradient descent optimization algorithm to adjust the weight of neurons by calculating the gradient of the loss function.

1.2.9 Receiver Operating Characteristic Curve

AUC - ROC curve is a performance measurement for classification problem at various thresholds settings. ROC is a probability curve and AUC represents degree or measure of separability. It tells how much model is capable of distinguishing between classes. Higher the AUC, better the model is at predicting 0s as 0s and 1s as 1s. By analogy, Higher the AUC, better the model is at distinguishing between patients with disease and no disease.

The ROC curve is plotted with TPR against the FPR where TPR is on y-axis and FPR is on the x-axis.

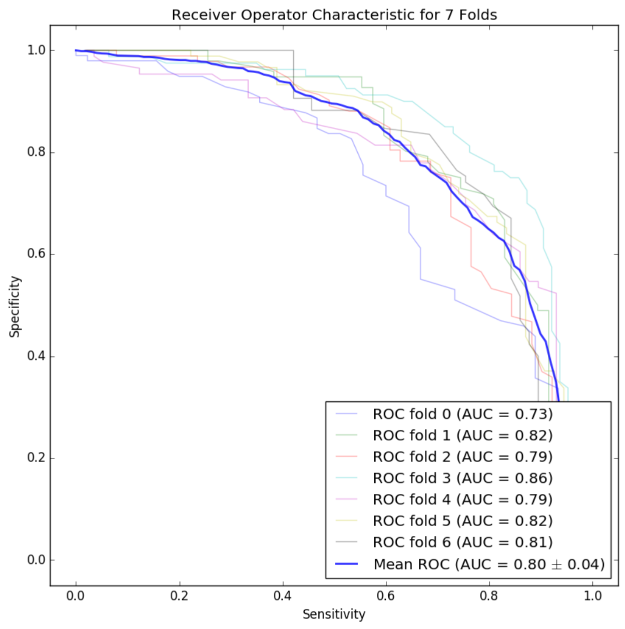


Fig 3: ROC curve

CHAPTER 2

METHODOLOGY

2.1 Tools Required

Sl. No.	Tool	Description
1.	**Google Cloud Platform**	Used to train model
2.	**Jupyter Notebook**	Python IDE
3.	**Heroku**	Platform on which flask script has been loaded
4.	**Android Studio**	Ide for android application

2.2 Libraries Used

Sl. No.	Library
1.	Keras
2.	Sklearn
3.	Skimage
4.	Theano
5.	Pillow
6.	Flask
7.	Gunicorn
8.	Postman
9.	Volley (REST API)

2.3 Dataset Used

Our dataset comes from a combination of open-access dermatology repositories, the ISIC Dermoscopic Archive, the Edinburgh Dermofit Library and data from the Stanford Hospital. The images from the online open-access dermatology repositories are annotated by dermatologists, not necessarily through biopsy. The ISIC Archive data used are composed strictly of melanocytic lesions that are biopsy-proven and annotated as malignant or benign. The Edinburgh Dermofit Library and data from the Stanford Hospital are biopsy-proven and annotated by individual disease names (that is, actinic keratosis). In our test sets, melanocytic lesions include malignant melanomas—the deadliest skin cancer—and benign nevi. Epidermal lesions include malignant basal and squamous cell carcinomas, intraepithelial carcinomas, pre-malignant actinic keratosis and benign seborrheic keratosis.

2000 images are provided as training data, including 374 "melanoma", 254 "seborrheic keratosis", and the remainder as benign nevi (1372). The training data is provided as a ZIP file, containing dermoscopic lesion images in JPEG format and a CSV file with some clinical metadata for each image.

All images are named using the scheme ISIC_<image_id>.jpg, where <image_id> is a 7-digit unique identifier. EXIF tags in the images have been removed; any remaining EXIF tags should not be relied upon to provide accurate metadata.

Ground Truth Data

The Training Ground Truth file is a single CSV (comma-separated value) file, containing 3 columns:

The first column of each row contains a string of the form ISIC_<image_id>, where <image_id> matches the corresponding Training Data image.

The second column of each row pertains to the first binary classification task (melanoma vs. nevus and seborrheic keratosis) and contains the value 0 or 1.

The number 1 = lesion is melanoma.

The number 0 = lesion is nevus or seborrheic keratosis.

The third column of each row pertains to the second classification task (seborrheic keratosis vs. melanoma and nevus) and contains the value 0 or 1.

The number 1 = lesion is seborrheic keratosis.

The number 0 = lesion is melanoma or nevus.

Malignancy diagnosis data were obtained from expert consensus and pathology report information. Participants are not strictly required to limit development to the training data, and are free to train their algorithm using external data sources. However, any other sources of data in system development must be properly cited in the abstract.

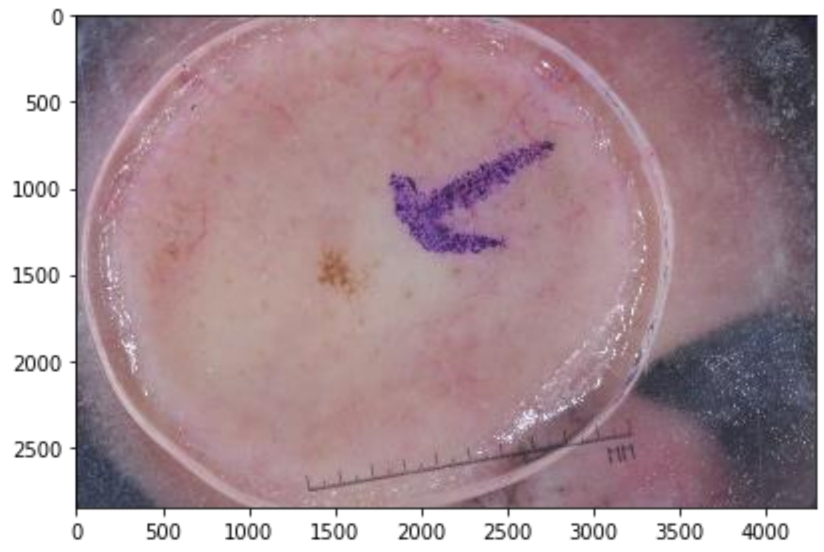


Fig :- 4 Seborrheic Keratosis Lesion

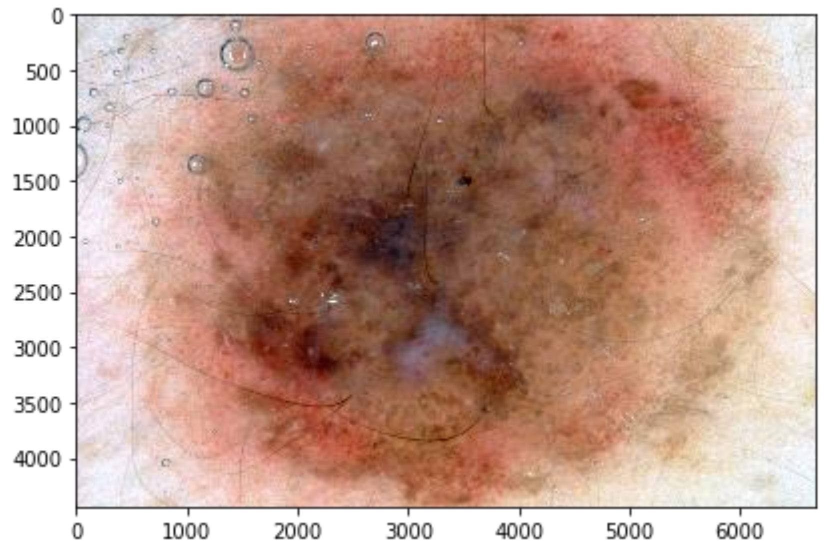


Fig :- 5 Nevus Lesion

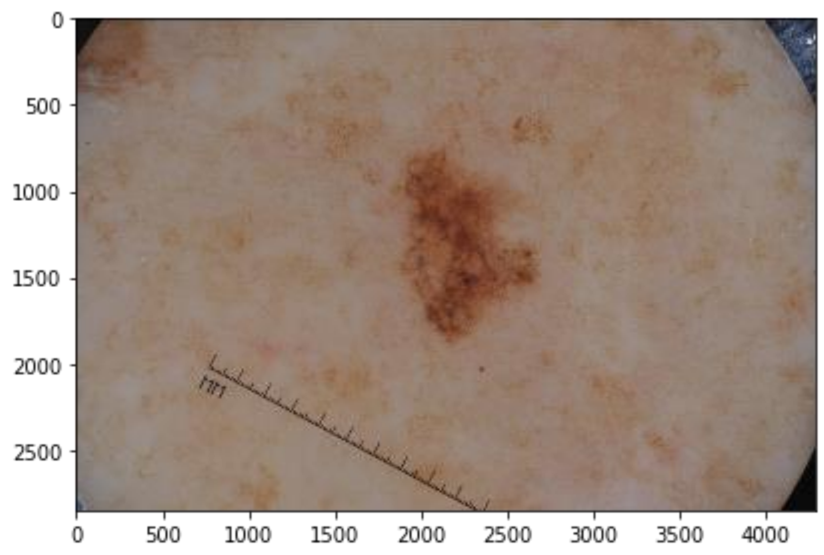


Fig :- 6 Melanoma Lesion

For model training we have divided the ISIC dataset into two parts 48% for training and 4% for cross validation, and remaining 48% is for testing.

Training Set: This data set is used to adjust the weights on the neural network.

Validation Set: This data set is used to minimize overfitting. You're not adjusting the weights of the network with this data set, you're just verifying that any increase in accuracy over the training data set actually yields an increase in accuracy over a data set that has not been shown to the network before, or at least the network hasn't trained on it (i.e. validation data set). If the accuracy over the training data set increases, but the accuracy over then validation data set stays the same or decreases, then you're overfitting your neural network and you should stop training.

Testing Set: This data set is used only for testing the final solution in order to confirm the actual predictive power of the network.

2.4 CONVOLUTION NEURAL NETWORK MODELING

During model training we have used modified google resnet model. We have popped the last layer of resnet model and append relu layer followed by fully connected layer1 having 2048 input neurons and 1024 output neurons, followed by batch normalization layer then final fully connected layer having 1024 input neurons and 3 output neurons.

The google resnet model has 4 layers each with multiple bottleneck layers each consists of some combination of convolution layer, pooling layer, batch normalization layer.

Model is trained on Google Colab that has a session time limit of 12 hours and a GPU capacity of 12 GB. This makes training a model with a more complex architecture or a greater batch size very challenging.

Dataset was obtained from the repository created by the original paper publishing team, https://github.com/udacity/dermatologist-ai. Given the limited storage capacity on Google Drive, the dataset was downloaded every time the Colab environment was setup for training.

The framework used is Pytorch.

Batch size = 64, greater batch couldn't be loaded due to limited GPU memory and and a smaller one wasn't enough for regularization.

We are using transforms to augment the images. The transform include random rotations (with a maximum limit of 5 degrees with respect to the center of the image), random horizontal flip with a probability of 10% and the same for a vertical flip.

All the images are rescaled/cropped to the dimension of 224*224.

We tried creating a model from scratch, combining dense layers and convolution layers. But it we soon ran out of GPU space and even our best models couldn't come close to the accuracy we were getting using pre-trained models, so we decided to go with transfer learning.

We are using a pre-trained resnet152 model and removing its last layer, the classifying layer and replacing it with our own sequence of layers, viz.

Leaky Relu : with a negative slope of 0.06. The value was fine tuned after testing multiple times. This activation function was chosen to make sure gradients flow back to change the pre-trained weights t conform it to the new dataset.

Dropout : This is the first dropout layer in the sequence with a probability of 20%. This value too was obtained after fine tuning.

Dense Layer : This is the first fully connected layer which takes the input from the pre-trained layers of the resnet152 model and outputs a layer of 512 nodes.

Batch Normalization : Since the resnet152 model utilizes batch normalization multiple times we added one after the first dense layer and as expected it helped in decreasing the validation loss.

Leaky Relu : Once again we used leaky relu to make sure the previous layer gets gradients to update their weights, although the negative slope value we use here is lower given the lesser number of layers it’s supposed to update.

Dropout : another dropout layer added to avoid overfitting with a 50% probability of dropping a node.

Dense Layer : this final fully connected layer is used for the final classification. It classifies the 1024 output nodes from the previous dense layer into 3 categories, i.e., melanoma, nevus and seborrheic_keratosis.

We initialize the weights of the two dense layers added separately using normal xavier distribution

We also froze some initial layers of the resnet152 model to prevent its weight from changing because the high-level features are almost similar for both datasets, the one we’re using and the imagenet dataset, plus our dataset is not as large as the imagenet.

The model is trained for 25 epochs with the initial learning rate of 0.05.

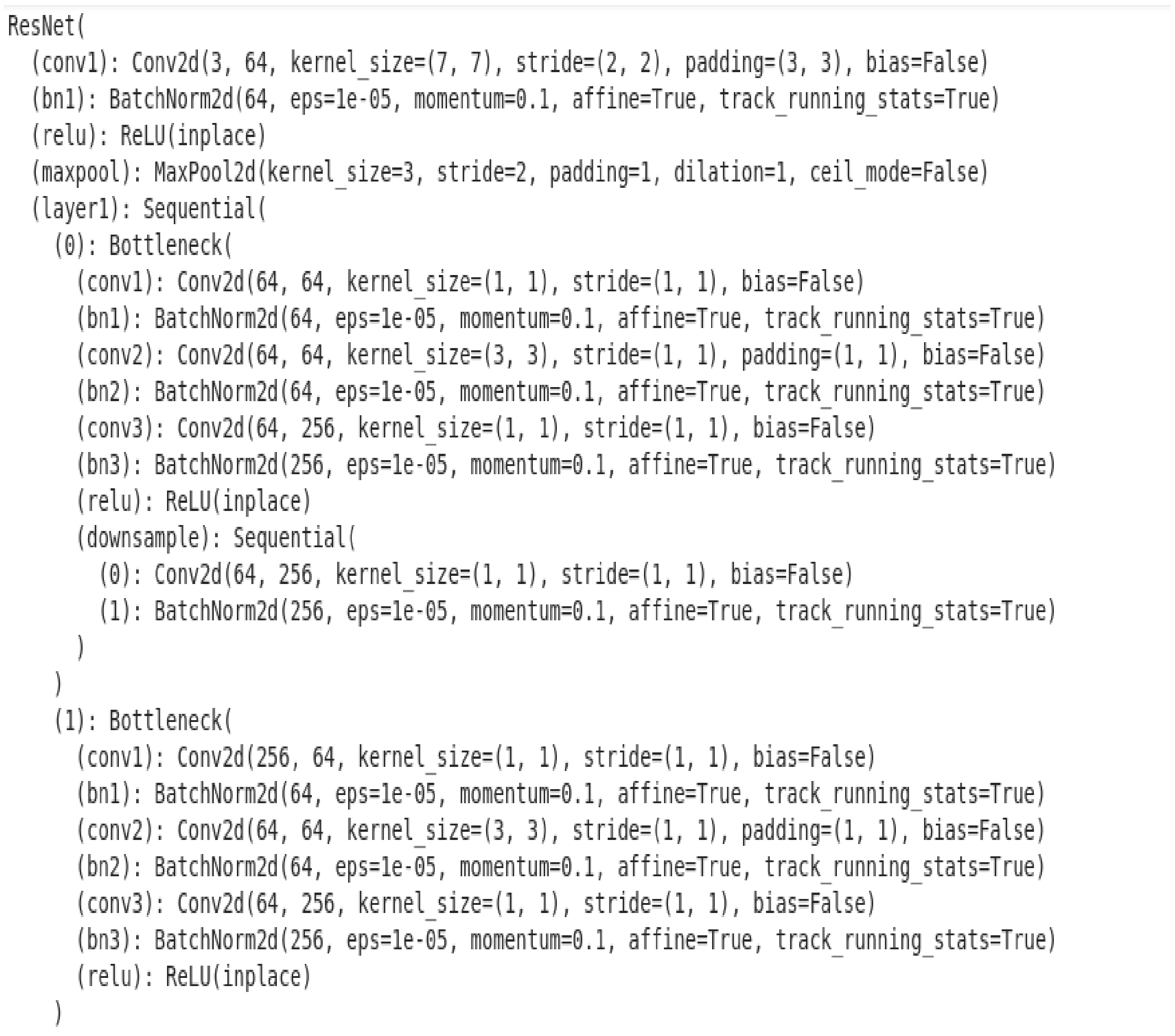
The criterion we’re using is the Cross Entropy Loss as it’s the most popular and of the most efficient loss function for classification problems.

We’re using the Stochastic Gradient Descent optimizer to update weights of the model. We’ve included a momentum with value 0.85 to prevent the loss function from getting stuck in a local minima.

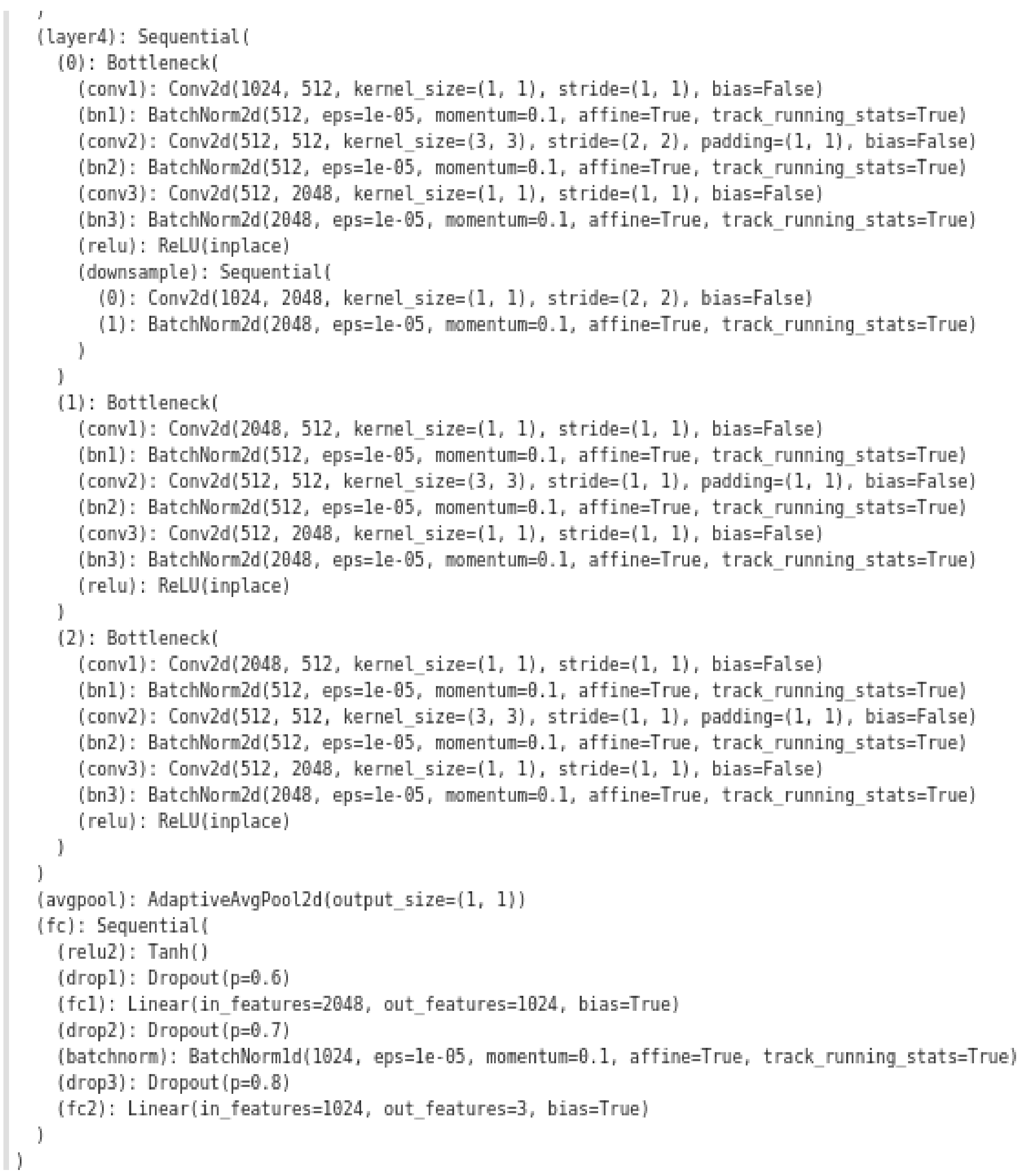
We’ve used a learning rate scheduler to prevent oscillating around a minima. The scheduler reduces learning rate by a factor of 5 if the validation loss doesn’t go below the lowest validation loss for three epochs consecutively.

During the training we save the model with the minimum validation loss as a checkpoint, i.e., we save it’s weights in a file.

After the training is complete, we load the checkpoint with the minimum loss into the model



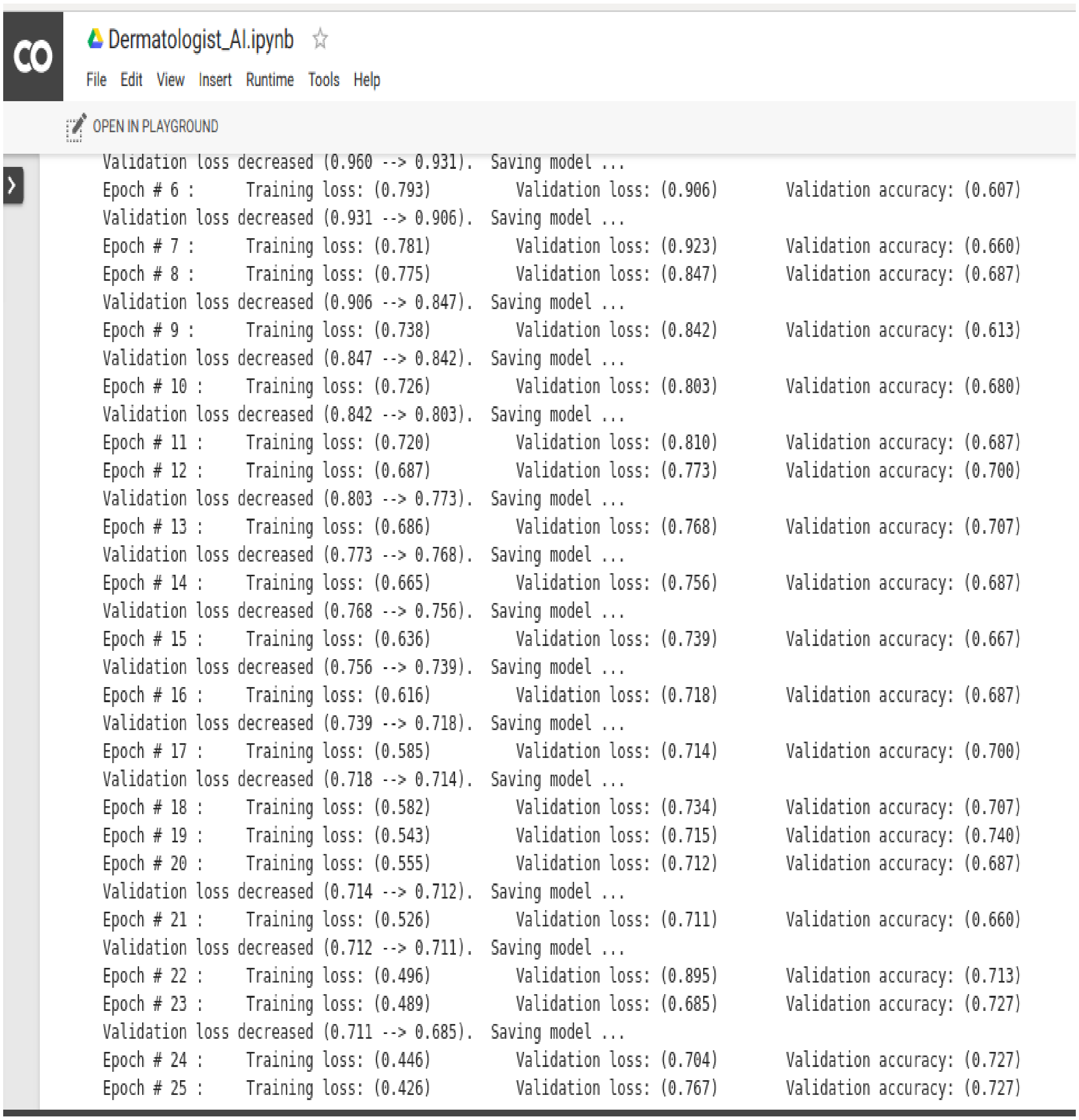
```
ResNet(
  (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu): ReLU(inplace)
  (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
  (layer1): Sequential(
    (0): Bottleneck(
      (conv1): Conv2d(64, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace)
      (downsample): Sequential(
        (0): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (1): Bottleneck(
      (conv1): Conv2d(256, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace)
    )
```



```
  ,
  (layer4): Sequential(
    (0): Bottleneck(
      (conv1): Conv2d(1024, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace)
      (downsample): Sequential(
        (0): Conv2d(1024, 2048, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (1): Bottleneck(
      (conv1): Conv2d(2048, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace)
    )
    (2): Bottleneck(
      (conv1): Conv2d(2048, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace)
    )
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
  (fc): Sequential(
    (relu2): Tanh()
    (drop1): Dropout(p=0.6)
    (fc1): Linear(in_features=2048, out_features=1024, bias=True)
    (drop2): Dropout(p=0.7)
    (batchnorm): BatchNorm1d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (drop3): Dropout(p=0.8)
    (fc2): Linear(in_features=1024, out_features=3, bias=True)
  )
)
```

Fig 7: Skin cancer detection model architecture

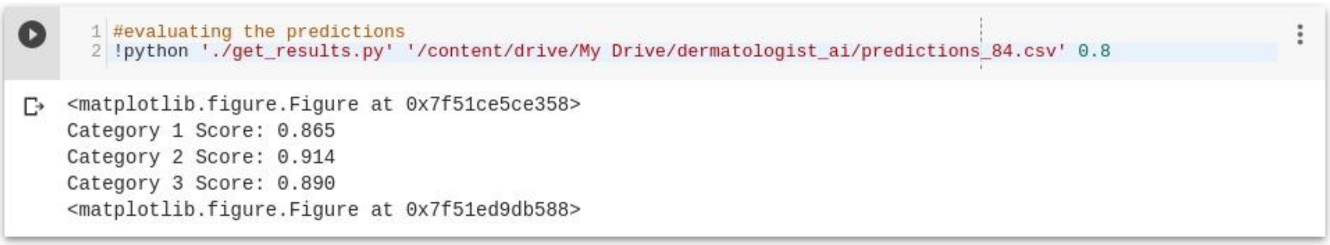
2.4.1 Training Summary



```
Validation loss decreased (0.960 --> 0.931).  Saving model ...
Epoch # 6 :     Training loss: (0.793)        Validation loss: (0.906)       Validation accuracy: (0.607)
Validation loss decreased (0.931 --> 0.906).  Saving model ...
Epoch # 7 :     Training loss: (0.781)        Validation loss: (0.923)       Validation accuracy: (0.660)
Epoch # 8 :     Training loss: (0.775)        Validation loss: (0.847)       Validation accuracy: (0.687)
Validation loss decreased (0.906 --> 0.847).  Saving model ...
Epoch # 9 :     Training loss: (0.738)        Validation loss: (0.842)       Validation accuracy: (0.613)
Validation loss decreased (0.847 --> 0.842).  Saving model ...
Epoch # 10 :    Training loss: (0.726)        Validation loss: (0.803)       Validation accuracy: (0.680)
Validation loss decreased (0.842 --> 0.803).  Saving model ...
Epoch # 11 :    Training loss: (0.720)        Validation loss: (0.810)       Validation accuracy: (0.687)
Epoch # 12 :    Training loss: (0.687)        Validation loss: (0.773)       Validation accuracy: (0.700)
Validation loss decreased (0.803 --> 0.773).  Saving model ...
Epoch # 13 :    Training loss: (0.686)        Validation loss: (0.768)       Validation accuracy: (0.707)
Validation loss decreased (0.773 --> 0.768).  Saving model ...
Epoch # 14 :    Training loss: (0.665)        Validation loss: (0.756)       Validation accuracy: (0.687)
Validation loss decreased (0.768 --> 0.756).  Saving model ...
Epoch # 15 :    Training loss: (0.636)        Validation loss: (0.739)       Validation accuracy: (0.667)
Validation loss decreased (0.756 --> 0.739).  Saving model ...
Epoch # 16 :    Training loss: (0.616)        Validation loss: (0.718)       Validation accuracy: (0.687)
Validation loss decreased (0.739 --> 0.718).  Saving model ...
Epoch # 17 :    Training loss: (0.585)        Validation loss: (0.714)       Validation accuracy: (0.700)
Validation loss decreased (0.718 --> 0.714).  Saving model ...
Epoch # 18 :    Training loss: (0.582)        Validation loss: (0.734)       Validation accuracy: (0.707)
Epoch # 19 :    Training loss: (0.543)        Validation loss: (0.715)       Validation accuracy: (0.740)
Epoch # 20 :    Training loss: (0.555)        Validation loss: (0.712)       Validation accuracy: (0.687)
Validation loss decreased (0.714 --> 0.712).  Saving model ...
Epoch # 21 :    Training loss: (0.526)        Validation loss: (0.711)       Validation accuracy: (0.660)
Validation loss decreased (0.712 --> 0.711).  Saving model ...
Epoch # 22 :    Training loss: (0.496)        Validation loss: (0.895)       Validation accuracy: (0.713)
Epoch # 23 :    Training loss: (0.489)        Validation loss: (0.685)       Validation accuracy: (0.727)
Validation loss decreased (0.711 --> 0.685).  Saving model ...
Epoch # 24 :    Training loss: (0.446)        Validation loss: (0.704)       Validation accuracy: (0.727)
Epoch # 25 :    Training loss: (0.426)        Validation loss: (0.767)       Validation accuracy: (0.727)
```

Fig :- 8 Training Summary

2.5 Prediction Results



```
#evaluating the predictions
!python './get_results.py' '/content/drive/My Drive/dermatologist_ai/predictions_84.csv' 0.8
```

```
<matplotlib.figure.Figure at 0x7f51ce5ce358>
Category 1 Score: 0.865
Category 2 Score: 0.914
Category 3 Score: 0.890
<matplotlib.figure.Figure at 0x7f51ed9db588>
```

Fig :- 9 Evaluating the predictions

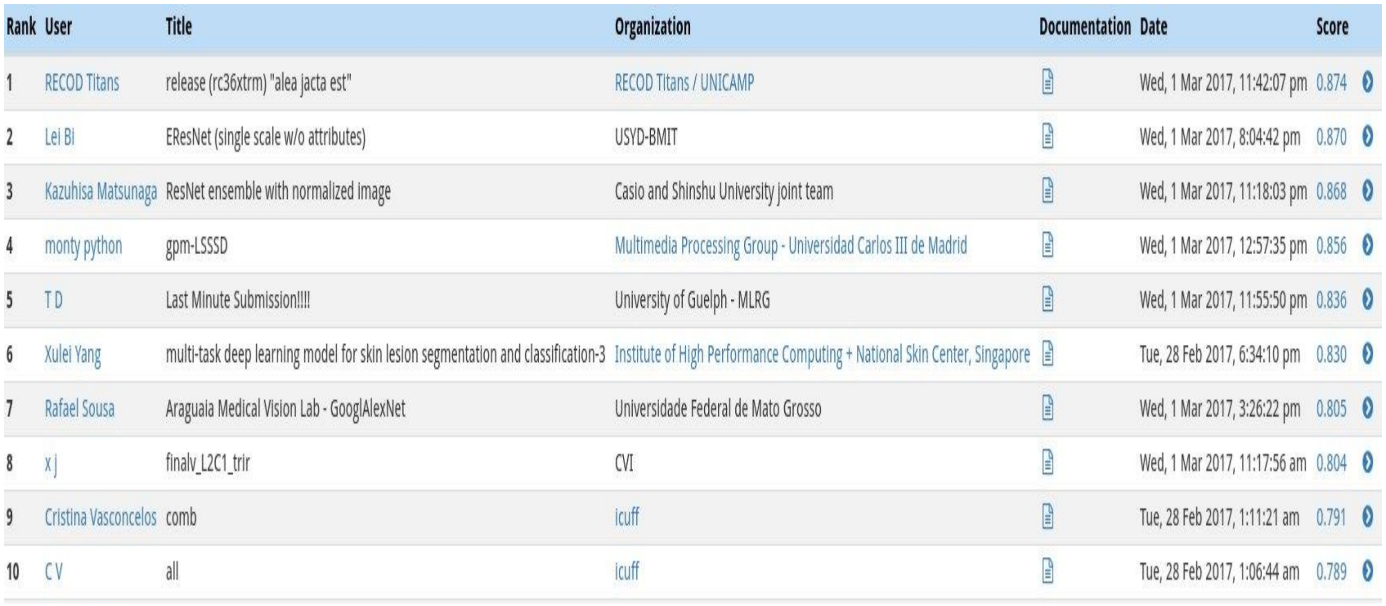
2.6 ISIC Competition Leadership Board

Since we have chosen this project from udacity skin cancer detection competition, so our model has been ranked according to separate three categories:-

Category 1: ROC AUC for Melanoma Classification

In the first category, we will gauge the ability of your CNN to distinguish between malignant melanoma and the benign skin lesions (nevus, seborrheic keratosis).

The top scores (from the ISIC competition) in this category can be found in the image below.



Rank	User	Title	Organization	Documentation	Date	Score
1	RECOD Titans	release (rc36xtrm) "alea jacta est"	RECOD Titans / UNICAMP	📄	Wed, 1 Mar 2017, 11:42:07 pm	0.874
2	Lei Bi	EResNet (single scale w/o attributes)	USYD-BMIT	📄	Wed, 1 Mar 2017, 8:04:42 pm	0.870
3	Kazuhisa Matsunaga	ResNet ensemble with normalized image	Casio and Shinshu University joint team	📄	Wed, 1 Mar 2017, 11:18:03 pm	0.868
4	monty python	gpm-LSSSD	Multimedia Processing Group - Universidad Carlos III de Madrid	📄	Wed, 1 Mar 2017, 12:57:35 pm	0.856
5	T D	Last Minute Submission!!!!	University of Guelph - MLRG	📄	Wed, 1 Mar 2017, 11:55:50 pm	0.836
6	Xulei Yang	multi-task deep learning model for skin lesion segmentation and classification-3	Institute of High Performance Computing + National Skin Center, Singapore	📄	Tue, 28 Feb 2017, 6:34:10 pm	0.830
7	Rafael Sousa	Araguaia Medical Vision Lab - GooglAlexNet	Universidade Federal de Mato Grosso	📄	Wed, 1 Mar 2017, 3:26:22 pm	0.805
8	x j	finalv_L2C1_trir	CVI	📄	Wed, 1 Mar 2017, 11:17:56 am	0.804
9	Cristina Vasconcelos	comb	icuff	📄	Tue, 28 Feb 2017, 1:11:21 am	0.791
10	C V	all	icuff	📄	Tue, 28 Feb 2017, 1:06:44 am	0.789

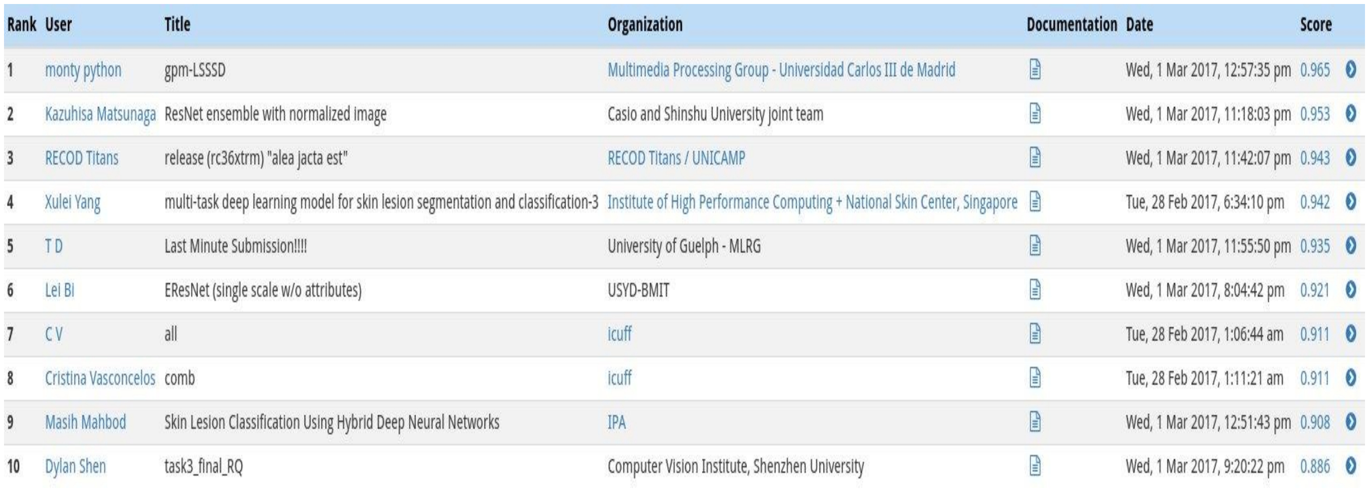
Our model's score in category 1 is 0.865.

Category 2: ROC AUC for Melanocytic Classification

All of the skin lesions that we will examine are caused by abnormal growth of either melanocytes or keratinocytes, which are two different types of epidermal skin cells. Melanomas and nevi are derived from melanocytes, whereas seborrheic keratoses are derived from keratinocytes.

In the second category, we will test the ability of your CNN to distinuish between melanocytic and keratinocytic skin lesions.

The top scores in this category (from the ISIC competition) can be found in the image below.



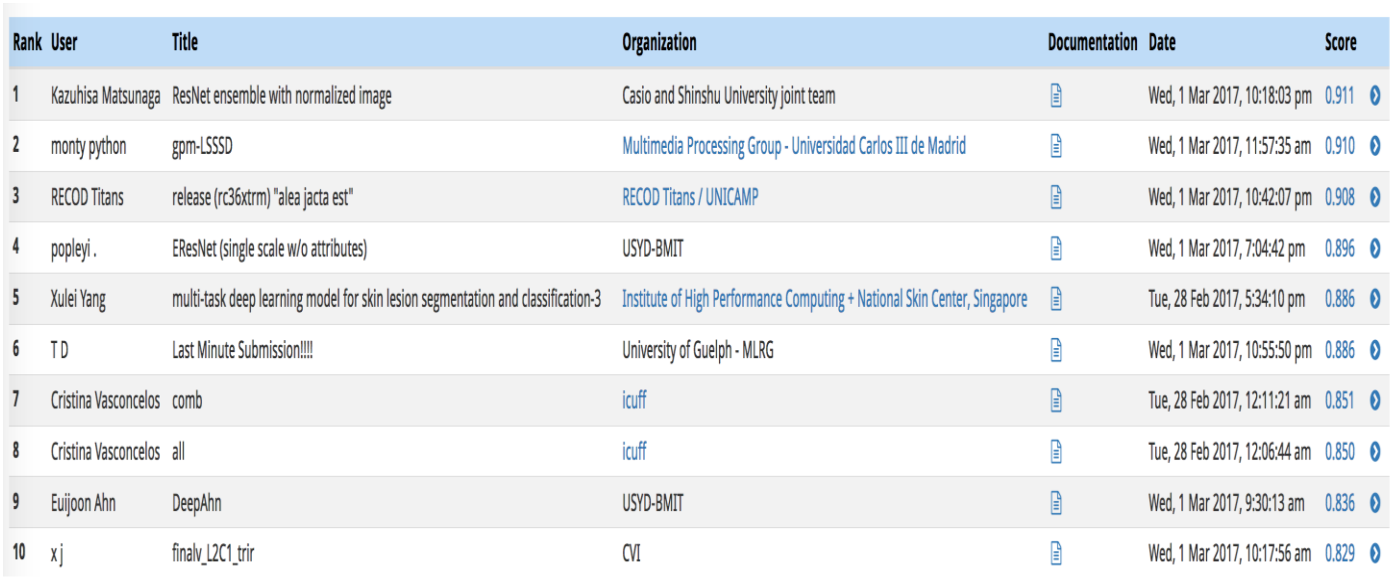
Rank	User	Title	Organization	Documentation	Date	Score
1	monty python	gpm-LSSSD	Multimedia Processing Group - Universidad Carlos III de Madrid	📄	Wed, 1 Mar 2017, 12:57:35 pm	0.965
2	Kazuhisa Matsunaga	ResNet ensemble with normalized image	Casio and Shinshu University joint team	📄	Wed, 1 Mar 2017, 11:18:03 pm	0.953
3	RECOD Titans	release (rc36xtrm) "alea jacta est"	RECOD Titans / UNICAMP	📄	Wed, 1 Mar 2017, 11:42:07 pm	0.943
4	Xulei Yang	multi-task deep learning model for skin lesion segmentation and classification-3	Institute of High Performance Computing + National Skin Center, Singapore	📄	Tue, 28 Feb 2017, 6:34:10 pm	0.942
5	T D	Last Minute Submission!!!!	University of Guelph - MLRG	📄	Wed, 1 Mar 2017, 11:55:50 pm	0.935
6	Lei Bi	EResNet (single scale w/o attributes)	USYD-BMIT	📄	Wed, 1 Mar 2017, 8:04:42 pm	0.921
7	C V	all	icuff	📄	Tue, 28 Feb 2017, 1:06:44 am	0.911
8	Cristina Vasconcelos	comb	icuff	📄	Tue, 28 Feb 2017, 1:11:21 am	0.911
9	Masih Mahbod	Skin Lesion Classification Using Hybrid Deep Neural Networks	IPA	📄	Wed, 1 Mar 2017, 12:51:43 pm	0.908
10	Dylan Shen	task3_final_RQ	Computer Vision Institute, Shenzhen University	📄	Wed, 1 Mar 2017, 9:20:22 pm	0.886

Our CNN model's category 2 score is 0.914.

Category 3: Mean ROC AUC

In the third category, we will take the average of the ROC AUC values from the first two categories.

The top scores in this category (from the ISIC competition) can be found in the image below.



Rank	User	Title	Organization	Documentation	Date	Score
1	Kazuhisa Matsunaga	ResNet ensemble with normalized image	Casio and Shinshu University joint team	📄	Wed, 1 Mar 2017, 10:18:03 pm	0.911
2	monty python	gpm-LSSSD	Multimedia Processing Group - Universidad Carlos III de Madrid	📄	Wed, 1 Mar 2017, 11:57:35 am	0.910
3	RECOD Titans	release (rc36xtrm) "alea jacta est"	RECOD Titans / UNICAMP	📄	Wed, 1 Mar 2017, 10:42:07 pm	0.908
4	popleyi .	EResNet (single scale w/o attributes)	USYD-BMIT	📄	Wed, 1 Mar 2017, 7:04:42 pm	0.896
5	Xulei Yang	multi-task deep learning model for skin lesion segmentation and classification-3	Institute of High Performance Computing + National Skin Center, Singapore	📄	Tue, 28 Feb 2017, 5:34:10 pm	0.886
6	T D	Last Minute Submission!!!!	University of Guelph - MLRG	📄	Wed, 1 Mar 2017, 10:55:50 pm	0.886
7	Cristina Vasconcelos	comb	icuff	📄	Tue, 28 Feb 2017, 12:11:21 am	0.851
8	Cristina Vasconcelos	all	icuff	📄	Tue, 28 Feb 2017, 12:06:44 am	0.850
9	Euijoon Ahn	DeepAhn	USYD-BMIT	📄	Wed, 1 Mar 2017, 9:30:13 am	0.836
10	x j	finalv_L2C1_trir	CVI	📄	Wed, 1 Mar 2017, 10:17:56 am	0.829

Our CNN model's category 3 score is 0.890.

Overall we stand among top 10 participants of the ISIC competition.

2.7 Backend Script using Flask on Heroku

The application uses a back-end server deployed on Heroku, that provides cloud platform as a service (PaaS). The server runs a python script that accepts incoming HTTP requests, in our case, POST requests, using the Flask web micro-framework.

2.7.1 The server has the following files stored:

The exported trained Keras model, named gender_sa.h5

A python script that accepts incoming HTTP POST requests by implementing the Flask micro-framework and returns a predicitons as json, named predictor.py.

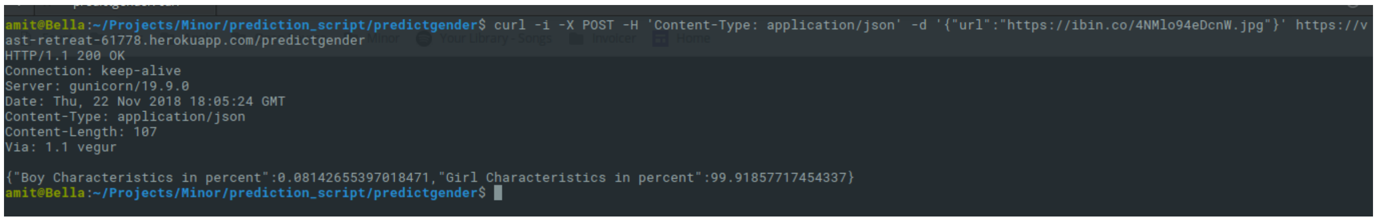
A Procfile that contains commnds to be run by the Heroku dyno, named Procfile

A text file that contains all the required libraries that need to be installed in order to run the script, named requirements.txt

A text file that mentions the runtime environment and its version for the script.

2.7.2 Flowchart:

1. -The android app sends HTTP POST requests
2. -The python script receives that request by implementing the Flask micro-framework
3. -The script calls the loadthemodel() function which loads the trained Keras model. This function is called only once and that is when the first POST request is received.
4. -Now the make_predict() function is called which json sent through the POST request.
5. -Now the image url is extraced from the json and then the image is downloaded.
6. -The downloaded image is resized into a 100*100 image and then transformed into grayscale.
7. -Now pixel data of this resized grayscale image is extracted and then saved in a 100*100 matrix.
8. -The matrix mentioned above is reshaped into a ndarray of shape (1, 100, 100, 1) as the model is trained using this shape.
9. -Now the reshaped matrix is provided as an argument to the predict () function of keras.models.Model class.
10. -The result returned by the predict () function is stored in a json and returned as the response to the incoming POST request.



```
amit@Bella:~/Projects/Minor/prediction_script/predictgender$ curl -i -X POST -H 'Content-Type: application/json' -d '{"url":"https://ibin.co/4NMlo94eDcnW.jpg"}' https://v
ast-retreat-61778.herokuapp.com/predictgender
HTTP/1.1 200 OK
Connection: keep-alive
Server: gunicorn/19.9.0
Date: Thu, 22 Nov 2018 18:05:24 GMT
Content-Type: application/json
Content-Length: 107
Via: 1.1 vegur

{"Boy Characteristics in percent":0.08142655397018471,"Girl Characteristics in percent":99.91857717454337}
amit@Bella:~/Projects/Minor/prediction_script/predictgender$ 
```

Fig :- 10 Curl script for prediction

2.8 Skin Cancer Detection Android Application

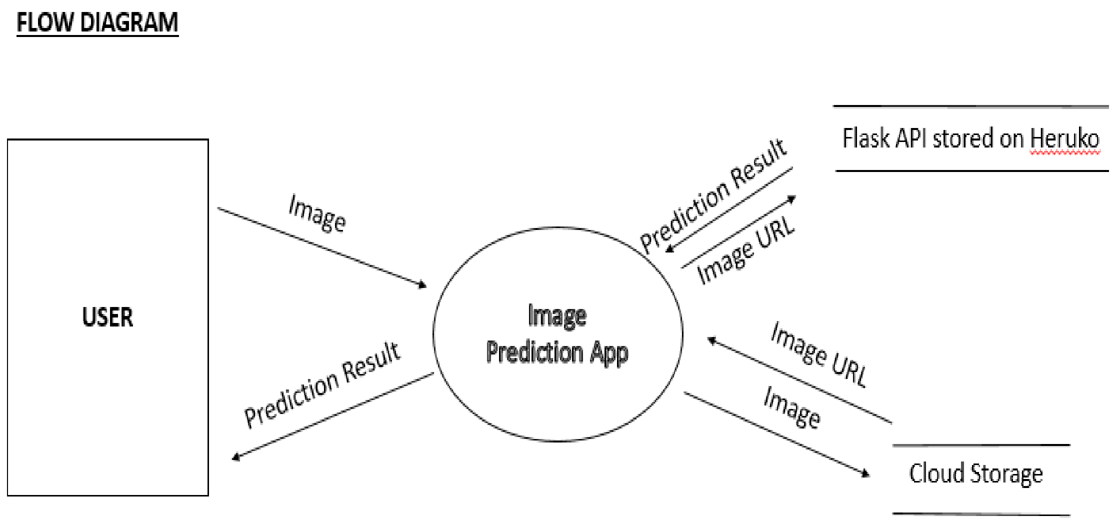


Fig :- 11 Android Application Flowchart

1. User uploads a image into the app. The image can be upload through gallery or can be clicked instantly.
2. On clicking on upload image, the image gets stored in Firebase Storage. The Firebase storage returns the image URL to the app. Cloud Storage for Firebase lets you upload and share user generated content, such as images and video, which allows you to build rich media content into your apps. Your data is stored in a Google Cloud Storage bucket, an exabyte scale object storage solution with high availability and global redundancy. Cloud Storage lets you securely upload these files directly from mobile devices and web browsers, handling spotty networks with ease.
3. The URL of the image is sent to the Flask Script stored n Heroku Server using a post request through Volley. Volley is an HTTP library that makes networking for Android apps easier and most importantly, faster

Volley is an HTTP library that makes networking for Android apps easier and most importantly, faster.

Volley offers the following benefits:

- Automatic scheduling of network requests.
- Multiple concurrent network connections.
- Transparent disk and memory response caching with standard HTTP cache coherence.
- Support for request prioritization.

- Cancellation request API. You can cancel a single request, or you can set blocks or scopes of requests to cancel.
- Ease of customization, for example, for retry and backoff.
- Strong ordering that makes it easy to correctly populate your UI with data fetched asynchronously from the network.
- Debugging and tracing tools.

The Script on the Heruko server loads the model. It downloads the image for given url and then resizes it and converts into grayscale using Pillow library. The gray scaled image is then predicted using the loaded model.

The prediction result is sent in the JSON format to the app.

The app shows the result to the user.

2.9 Snapshots

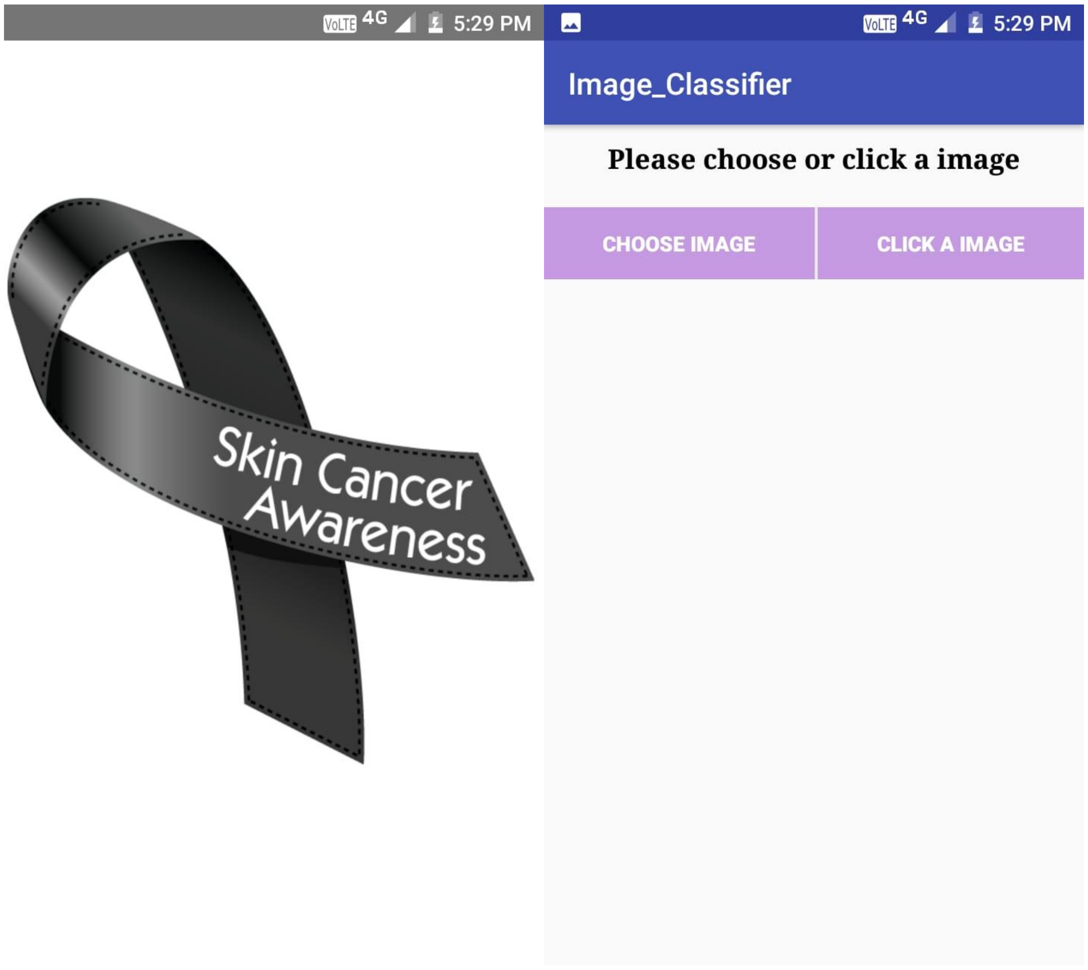
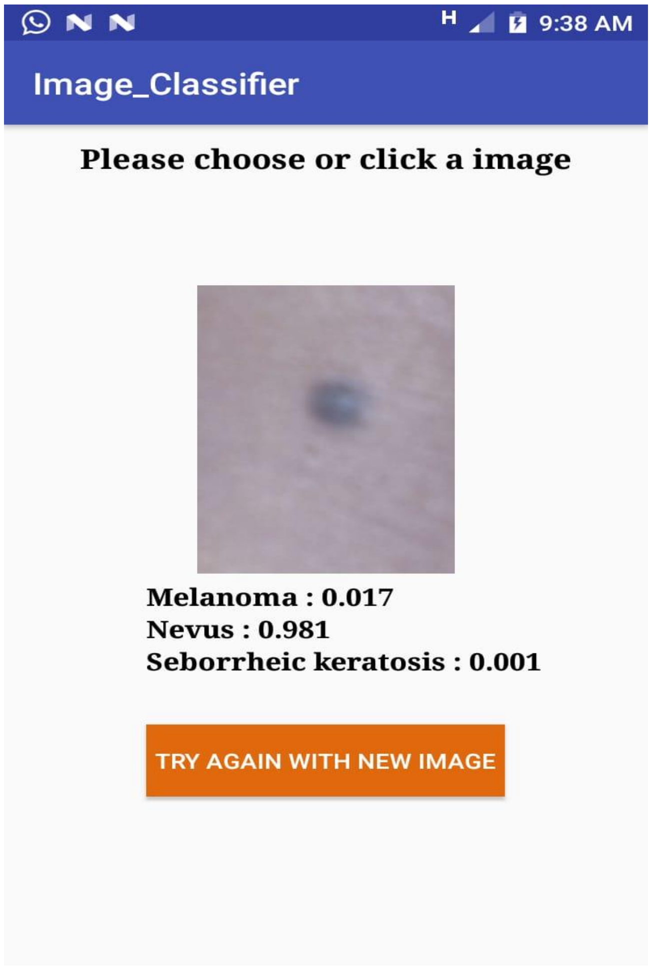


Fig 12: Android Application



Please choose or click a image

Melanoma : 0.017
Nevus : 0.981
Seborrheic keratosis : 0.001

CHAPTER 3

FUTURE ASPECTS AND CONCLUSION

The entire model was trained on high resolution medical images. But in real-life scenario the images will be taken from mobile phones. So, we need to create a dataset of images clicked from mobile phones.

The accuracy of our model is near to 76% and according to a survey a average dermatologist have a accuracy rate of 70%.

The total ratio of patients per dermatologist is quite high and may tend to increase with the increasing rate of pollution. So this feature can prove to be quite helpful, given that the actual accuracies of our model and actual dermatologists is almost same.

A web application can prove to be quite helpful for the same.

CHAPTER 4

REFERENCES

1. Matsunaga K, Hamada A, Minagawa A, Koga H. "Image Classification of Melanoma, Nevus and Seborrheic Keratosis by Deep Neural Network Ensemble". International Skin Imaging Collaboration (ISIC) 2017 Challenge at the International Symposium on Biomedical Imaging (ISBI).
2. Daz IG. "Incorporating the Knowledge of Dermatologists to Convolutional Neural Networks for the Diagnosis of Skin Lesions". International Skin Imaging Collaboration (ISIC) 2017 Challenge at the International Symposium on Biomedical Imaging (ISBI). (**github**)
3. Menegola A, Tavares J, Fornaciali M, Li LT, Avila S, Valle E. "RECOD Titans at ISIC Challenge 2017". International Skin Imaging Collaboration (ISIC) 2017 Challenge at the International Symposium on Biomedical Imaging (ISBI). (**github**)
4. https://github.com/udacity/dermatologist-ai
5. https://www.nature.com/articles/nature21056
6. https://hackernoon.com/machine-learning-for-isic-skin-cancer-classification-challenge-part-1-ccddea4ec44a
7. https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5
8. American Cancer Society. Cancer facts & figures 2016. Atlanta, American Cancer Society2016. http://www.cancer.org/acs/groups/content/@research/documents/document/acspc-047079.pdf
9. Rogers, H. W. *et al.* Incidence estimate of nonmelanoma skin cancer (keratinocyte carcinomas) in the US population, 2012. *JAMA Dermatology* 151.10, 1081–1086 (2015)
10. He, K. Zhang, X., Ren, S. & Sun J. Deep residual learning for image recognition. Preprint at https://arxiv.org/abs/1512.03385 (2015)
11. Pan, S. J. & Yang, Q. A survey on transfer learning. *IEEE Trans. Knowl. Data Eng.* 22, 1345–1359 (2010)