**EARLY IDENTIFICATION OF VISUAL IMPAIRMENT AND MACULAR PROTEIN DEGENERATION USING CONVOLUTIONAL NEURAL NETWORKS AND DEEP LEARNING FRAMEWORK**

**Abstract:**

Visual impairment is the common cause which impacts the independent living for many individuals. Visual impairment is caused by Macular protein degeneration. Macular protein degeneration is a specific eye disease which is common cause of blindness among people. As people age, the degeneration of retina, causes this disease which could potentially result to glaucoma, causing eventual blindness. The early detection of this disease will go a long way to identify the symptoms and do the corrective actions. The paper looks at different neural network models and analyzes the image augmentation techniques to make a clear identification. When combined with the clinical analysis, the trained neural network, when put to practical use will make the identification and early treatment seamless, cost effective and quick which normally takes weeks to reach a conclusion. The paper proposes the complete ecosystem where the data is made available through Mobile application to make the continuous feedback loop possible which helps in making the model to get better.

**Aim:**

The aim of this exercise to train the predictive neural network model to identify the early symptoms of macular protein degeneration disease from the prior knowledge of retina fundus images and labelled information. By applying the trained model on the test data, we can study how well the model does in real world situations. The proposed model is planned to use Convolutional neural network for the image analysis and prediction.

**Background:**

Macular protein degeneration is deterioration of the macula, which is the small central area of the retina, which is solely responsible for the visual activity. Macula's health plays a vital role in the eye activity of reading, recognition and identifying the finer details. Macular protein degeneration is the leading cause of vision for much of the US population [2]. As per the study of US center of disease control and prevention (CDC), the symptoms start as early as the age of 40. Macular protein degeneration is diagnosed either as dry (non -neovascular) or wet (neovascular). The growth of new blood vessels in the area of macula which are not supposed to be there, is an early indication of this possible potential disease. Macular protein degeneration impacts mainly the central vision, causing blind spots. The neovascular disease is more common and accounts for 85 to 90% of the cases diagnosed [2].

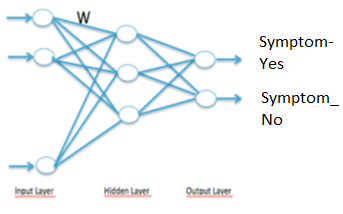
The possible causes for the dry macular protein degeneration arises from thinning of macular tissues, and deposition of pigments in the macula. The yellowish spots accumulate in and around the macula providing some early indication of this disease.

This is caused by diabetes damaging the small blood vessels in the retina. The leakage of the fluids from the damaged vessels causes the swelling of retina and infection in the macula, leading to the vision loss. If the symptoms are not treated early, it may lead to potential long term vision loss. The current process of detecting retinal degeneration including Diabetic Retinopathy is very time consuming and laborious process.

It requires trained technician to examine and evaluate digital fundus photographs of the retina. The results need to be manually examined and it results in delayed results and treatment. The advent of image recognition combined with deep learning can help to detect the lesions associated with the vascular abnormalities and help to identify the onset of the disease early in the diagnosis.

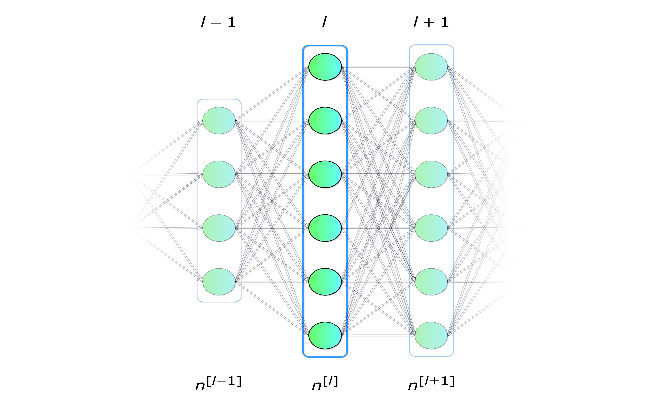
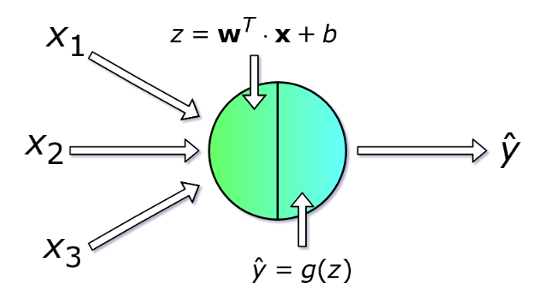
Using color fundus photography as input, the automated detection system can help to realistic clinical potential Image analysis with Convolutional neural network [9].

The input values (x-values) are received by neuron and it predicts the y value. Each of the unit has its own set of parameters weights and bias which changes during the learning process. In each iteration, the neuron calculates a weighted average of the values of the vector x, based on its current weight vector w and added bias which changes during the learning process. The result of the calculation is passed through a non-linear activation function.



https://miro.medium.com/max/451/1*hJjnKS-Kp5mBKzI6_kd3xg.gif

Single Neuron Neuron Layers



The loss function is designed to show how far away from the ideal solution the actual solution is, and the process is designed to minimize the loss function. As part of the epoch in the Keras, and through each part of the iteration, the aim is to minimize the loss function and at the same time increasing the accuracy of the predicted results. The learning process is through the changing the value of weight and the bias. Gradient descent function is used to find the progress towards minimum value.

**Dataset:**

The dataset is obtained from Kaggle, public dataset repository. The dataset includes large set of high-resolution retina images taken under a variety of imaging conditions. A left and right field is provided for every subject. Images are labeled with a subject id as well as either left or right (e.g. 1\_left.jpeg is the left eye of patient id 1). The dataset also has the rating done by the clinician for the presence of diabetic retinopathy in each image on the scale of 0 to 4, according to the following scale:

0 - Normal   
1 - Mild  
2 - Moderate  
3 - Severe  
4 - Proliferative visual impairment

The aim of the model is to do automated analysis using convolutional neural network system, which is capable of assigning a score based on the classification scale. The images of the dataset come from multiple sources with varying degree of image sizes. As with any real word examples, there are noises in the dataset in both the images and labels. Images might be out of focus, underexposed or overexposed.

Datasets available and the descriptions:

|  |  |
| --- | --- |
| Name | Description |
| Train.zip | Training dataset holds close to 35K imgaes |
| Test.zip | Test data for the prediction analysis |
| Sample.zip | Sample set of images for prediction and validation |
| trainLabels.csv | Contains subject identification, subject image name, and the clinical ranking for the level identification |

**Dataset initial analysis:**

The train Labels was loaded into pandas data frame and the initial data analysis of the data.

|  |  |
| --- | --- |
| <class 'pandas.core.frame.DataFrame'>  RangeIndex: 35126 entries, 0 to 35125  Data columns (total 2 columns):  image 35126 non-null object  level 35126 non-null int64  dtypes: int64(1), object(1)  memory usage: 548.9+ KB | Shape: (35126, 2)  35126 train images with left and right eye with the clinical ranking |
| Frequency distribution of Levels in the training | Histogram of the Diabetes retinopathy with levels. As expected close to 73% is normal eye. Leaving to highly unbalanced distribution of the training data.  Lvl Count  0 25810  1 2443  2 5292  3 873  4 708  Lvl % Distribution  0 73.478335  2 15.065763  1 6.954962  3 2.485338  4 2.015601 |

**Image resize and noise reduction:**

As the images size widely vary with varying degree of exposures and variation, the images were resized to standard size of 1024 and again cropped to reduce the noises in the images. To bring the uniformity the black space is removed by identifying the center and radius of the circle of the fundus image. If the image is turned out to be fully black they were manually removed and cleaned out before the training dataset was filtered. For easier classification using Keras, Tensor flow module, [“flow\_from\_directory”], each of the classification was identified under the separate folder named Level0, Level1, Level2, Level3 and Level4.

Folder structure for the Convolutional Neural network model using Keras:

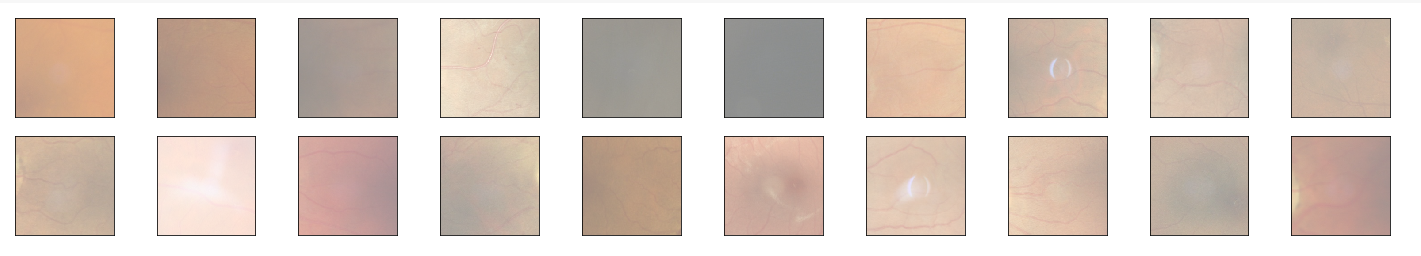
|  |  |  |
| --- | --- | --- |
| Train   * Level0 * Level1 * Level2 * Level3 * Level4 | Valid   * Level0 * Level1 * Level2 * Level3 * Level4 | Test   * Test |

**Adding weight constraints to reduce the overfitting of the model:**

As the majority (close to 73%) is of Level 0 classification, the classes need to be normalized to avoid the model doing the overfitting on the level 0 classification. One of the ways this can be done using the weight constraint for each of the classification labels. This is based on the % distribution of the classes in the training dataset. Weight constraints applied in the model to avoid the overfitting the uniform classification of the training dataset is as follows:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Level 0 | Level 1 | Level 2 | Level 3 | Level 4 | Total |
| Count | 25810 | 2443 | 5292 | 873 | 708 | 35126 |
| % | 73.48% | 6.95% | 15.07% | 2.49% | 2.02% | 100.00% |
| Normalization | 36 | 3 | 7 | 1 | 1 |  |

Example: each images in the Level 0 is weighted as 36 times as of Level 3 or Level 4.



**Image rescaling and classes identification for the CNN:**

Train Data gen : ImageDatagenerator was used to normalize the images by diving the RGB values by 255

Target size of (200,200) and batch size of 128 with the explicit definition of classes were identified for the training data generator.

Valid Data gen: Same as Train Data gen with the flow directory from the valid folder

**Convolutional Neural Network Model: (Model-1)**

In Keras, sequential model layer was selected for the Neural network model. The details of the CNN model is outlined as below:

* Conv2D with 16 neurons, with the filter size of 3,3 with the activation function = RELU
* The output of the layer was max pooled with the size of 2,2
* The above layer was repeated with the same 32 neurons
* Conv2D with 32 neurons with the filter size of 3,3 with the activation function = RELU
* The output of the layer was max pooled with the size of 2,2
* Conv2D with 64 neurons with the filter size of 3,3 with the activation function = RELU
* The output of the layer was max pooled with the size of 2,2
* Conv2D with 64 neurons with the filter size of 3,3 with the activation function = RELU
* The output of the layer was max pooled with the size of 2,2
* Flatter the layer
* Dense layer with Activation of RELU
* Finally dense layer with the SOFTMAX activation for the probably distribution to identify the classification label

**Model Summary:**

Model: "sequential"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

=================================================================

conv2d (Conv2D) (None, 198, 198, 16) 448

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

max\_pooling2d (MaxPooling2D) (None, 99, 99, 16) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

conv2d\_1 (Conv2D) (None, 97, 97, 32) 4640

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

max\_pooling2d\_1 (MaxPooling2 (None, 48, 48, 32) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

conv2d\_2 (Conv2D) (None, 46, 46, 64) 18496

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

max\_pooling2d\_2 (MaxPooling2 (None, 23, 23, 64) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

conv2d\_3 (Conv2D) (None, 21, 21, 64) 36928

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

max\_pooling2d\_3 (MaxPooling2 (None, 10, 10, 64) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

conv2d\_4 (Conv2D) (None, 8, 8, 64) 36928

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

max\_pooling2d\_4 (MaxPooling2 (None, 4, 4, 64) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

flatten (Flatten) (None, 1024) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense (Dense) (None, 128) 131200

**Convolutional Neural Network Model: ( Model-2)**

As the model 1 performance was less than satisfactory, multiple convolutional layers with the batch normalization were added to improve the training predictions.

The 2nd model expanded upon the first one by adding multiple Conv2D layer with batch normalizations.

* Conv2D with 16 neurons, with the filter size of 3,3 with the activation function = RELU
* Conv2D with 16 neurons, with the filter size of 3,3 with the activation function = RELU
* Batch Normalization ( to normalize the weight values)
* Zero padding
* Conv2D with 32 neurons, with the filter size of 3,3 with the activation function = RELU
* Conv2D with 32 neurons, with the filter size of 3,3 with the activation function = RELU
* Batch Normalization ( to normalize the weight values)
* Zero padding
* Conv2D with 64 neurons, with the filter size of 3,3 with the activation function = RELU
* Conv2D with 64 neurons, with the filter size of 3,3 with the activation function = RELU
* Batch Normalization ( to normalize the weight values)
* Zero padding
* Maxpooling (2, 2)
* Conv2D with 128 neurons, with the filter size of 3,3 with the activation function = RELU
* Conv2D with 128 neurons, with the filter size of 3,3 with the activation function = RELU
* Batch Normalization ( to normalize the weight values)
* Conv2D with 64 neurons, with the filter size of 3,3 with the activation function = RELU
* Conv2D with 64 neurons, with the filter size of 3,3 with the activation function = RELU
* Batch Normalization ( to normalize the weight values)
* Conv2D with 64 neurons, with the filter size of 3,3 with the activation function = RELU
* Conv2D with 64 neurons, with the filter size of 3,3 with the activation function = RELU
* Batch Normalization ( to normalize the weight values)
* Max Pooling (2,2)
* Conv2D with 128 neurons, with the filter size of 3,3 with the activation function = RELU
* Conv2D with 128 neurons, with the filter size of 3,3 with the activation function = RELU
* Batch Normalization
* Flatten
* Dense ( Fully Connected Layer)
* Dropout
* Dense(with Sigma activation)

**Model Summary:**

Model: "model\_1"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

=================================================================

input\_1 (InputLayer) (None, 256, 256, 3) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

conv2d\_1 (Conv2D) (None, 254, 254, 16) 448

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

conv2d\_2 (Conv2D) (None, 252, 252, 16) 2320

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

batch\_normalization\_1 (Batch (None, 252, 252, 16) 64

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

zero\_padding2d\_1 (ZeroPaddin (None, 254, 254, 16) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

conv2d\_3 (Conv2D) (None, 252, 252, 32) 4640

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

conv2d\_4 (Conv2D) (None, 250, 250, 32) 9248

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

batch\_normalization\_2 (Batch (None, 250, 250, 32) 128

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

conv2d\_5 (Conv2D) (None, 248, 248, 64) 18496

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

conv2d\_6 (Conv2D) (None, 246, 246, 64) 36928

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

batch\_normalization\_3 (Batch (None, 246, 246, 64) 256

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

max\_pooling2d\_1 (MaxPooling2 (None, 123, 123, 64) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

zero\_padding2d\_2 (ZeroPaddin (None, 125, 125, 64) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

conv2d\_7 (Conv2D) (None, 123, 123, 128) 73856

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

conv2d\_8 (Conv2D) (None, 122, 122, 128) 65664

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

batch\_normalization\_4 (Batch (None, 122, 122, 128) 512

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

conv2d\_9 (Conv2D) (None, 120, 120, 64) 73792

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

conv2d\_10 (Conv2D) (None, 118, 118, 64) 36928

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

batch\_normalization\_5 (Batch (None, 118, 118, 64) 256

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

conv2d\_11 (Conv2D) (None, 116, 116, 64) 36928

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

conv2d\_12 (Conv2D) (None, 114, 114, 64) 36928

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

batch\_normalization\_6 (Batch (None, 114, 114, 64) 256

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

max\_pooling2d\_2 (MaxPooling2 (None, 57, 57, 64) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

conv2d\_13 (Conv2D) (None, 55, 55, 64) 36928

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

conv2d\_14 (Conv2D) (None, 53, 53, 64) 36928

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

batch\_normalization\_7 (Batch (None, 53, 53, 64) 256

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

conv2d\_15 (Conv2D) (None, 51, 51, 128) 73856

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

conv2d\_16 (Conv2D) (None, 50, 50, 128) 65664

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

batch\_normalization\_8 (Batch (None, 50, 50, 128) 512

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

flatten\_1 (Flatten) (None, 320000) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_1 (Dense) (None, 32) 10240032

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dropout\_1 (Dropout) (None, 32) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_2 (Dense) (None, 32) 1056

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dropout\_2 (Dropout) (None, 32) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_3 (Dense) (None, 5) 165

=================================================================

Total params: 10,853,045

Trainable params: 10,851,925

Non-trainable params: 1,120

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Optimizer Used in the model: (Stochastic Gradient Descent / RMS Optimizer / Adam optimizer)**

The most common optimizer used in the CNN models is gradient descent optimizer. It is an effective method to find the optimal values for the neural network. The objective of all the optimizers is to reach the global minima where the cost function attains the least possible value.

The idea behind the gradient descent optimizer Algorithm is

* For each example in the data  
   - find the value predicted by the neural network   
   - calculate the loss from the loss function   
   - find partial derivatives of the loss function, these partial derivatives produce gradients  
   - use the gradients to update the values of weights and biases

Learning rate is the important aspect of the gradient descent which dictates the fastness of the learning and how effectively it approaches the solution. Larger learning rate, will lead to wandering around the global minima with the chances of convergence being very low. Too low learning rate will make the convergence slower, and in this, the risk of overshooting the minima always exists.

Another option for the optimizer is gradient descent with momentum, by increasing the momentum in the x direction, the oscillation on the y direction is decreased resulting in faster convergence. RMS optimizer is the form of gradient descent optimizer with momentum. The difference between the RMS optimizer and the gradient descent optimizer is on how the gradients are calculated.

**Results**:

**Accuracy details for:** Image size of 200 x 200, minimum data augmentation, batch size of 128 with RMS optimizer

Epoch 1/30

19/20 [===========================>..] - ETA: 43s - loss: 1.6758 - acc: 0.4038 Epoch 1/30

20/20 [==============================] - 889s 44s/step - loss: 1.6683 - acc: 0.4024 - val\_loss: 1.4135 - val\_acc: 0.5469

Epoch 2/30

19/20 [===========================>..] - ETA: 41s - loss: 1.4588 - acc: 0.4511 Epoch 1/30

20/20 [==============================] - 840s 42s/step - loss: 1.4595 - acc: 0.4504 - val\_loss: 1.3733 - val\_acc: 0.5469

Epoch 3/30

19/20 [===========================>..] - ETA: 13s - loss: 1.4856 - acc: 0.4370Epoch 1/30

20/20 [==============================] - 271s 14s/step - loss: 1.4861 - acc: 0.4366 - val\_loss: 1.3837 - val\_acc: 0.5469

|  |  |
| --- | --- |
|  |  |

**Accuracy details for**: Image size of 256 x 256, with data augmentation of horizontal flip, vertical flip and rotation, batch size of 32 with stochastic gradient optimizer

Epoch 1/5

176/176 [==============================] - 1612s 9s/step - loss: 7.7480 - acc: 0.7746 - val\_loss: 2.6659 - val\_acc: 0.8000

Epoch 2/5

176/176 [==============================] - 1585s 9s/step - loss: 2.6306 - acc: 0.7963 - val\_loss: 2.0110 - val\_acc: 0.8000

Epoch 3/5

176/176 [==============================] - 1618s 9s/step - loss: 1.3899 - acc: 0.7951 - val\_loss: 0.9826 - val\_acc: 0.7994

Epoch 4/5

176/176 [==============================] - 1610s 9s/step - loss: 0.7342 - acc: 0.7977 - val\_loss: 0.5871 - val\_acc: 0.8000

Epoch 5/5

176/176 [==============================] - 1589s 9s/step - loss: 0.5864 - acc: 0.7988 - val\_loss: 0.5006 - val\_acc: 0.8000

|  |  |
| --- | --- |
|  |  |

**Adam optimizer**

Epoch 1/2

176/176 [==============================] - 1596s 9s/step - loss: 0.7576 - acc: 0.7918 - val\_loss: 0.5438 - val\_acc: 0.8000

Epoch 2/2

176/176 [==============================] - 1575s 9s/step - loss: 0.5387 - acc: 0.7998 - val\_loss: 0.4704 - val\_acc: 0.8000

|  |  |
| --- | --- |
|  |  |

**Results Summary:**

|  |  |  |  |
| --- | --- | --- | --- |
| Model | Image Size | Optimizer | Predictions |
| Model -1 | 200 x 200  256 x 256 | RMSProp  Stochastic Gradient  Adam | Max accuracy : 48%  Valid accuracy:54% |
| Model -2 | 256 x 256 | RMSProp  Stochastic Gradient  Adam | Max accuracy : 80%  Valid accuracy: 81% |
| RestNet | 256 X 256 | Adam | Mac accuracy: 79.79%  Valid accuracy: 76.84% |
| RestNet -2 | 256 X 256 | Adam | Accuracy: 77.34%  Val Accuracy: 77.72% |
| VGG16 Model | 256 X 256 | RMS Optimizer | Accuracy: 76.29%  Val Accuracy: 78.89% |

Using Transfer Learning to train the images and predict the results:

Model: ResNet50 ( Keras trained with imagenet dataset for initial weights)

Epoch 1/2

Learning rate: 0.001

176/176 [==============================] - 5542s 31s/step - loss: 1.0134 - acc: 0.7777 - val\_loss: 1.8952 - val\_acc: 0.7684

Epoch 00001: val\_acc improved from -inf to 0.76842, saving model to /content/gdrive/My Drive/ML /data/DiabeticRetinopathy/model\_name

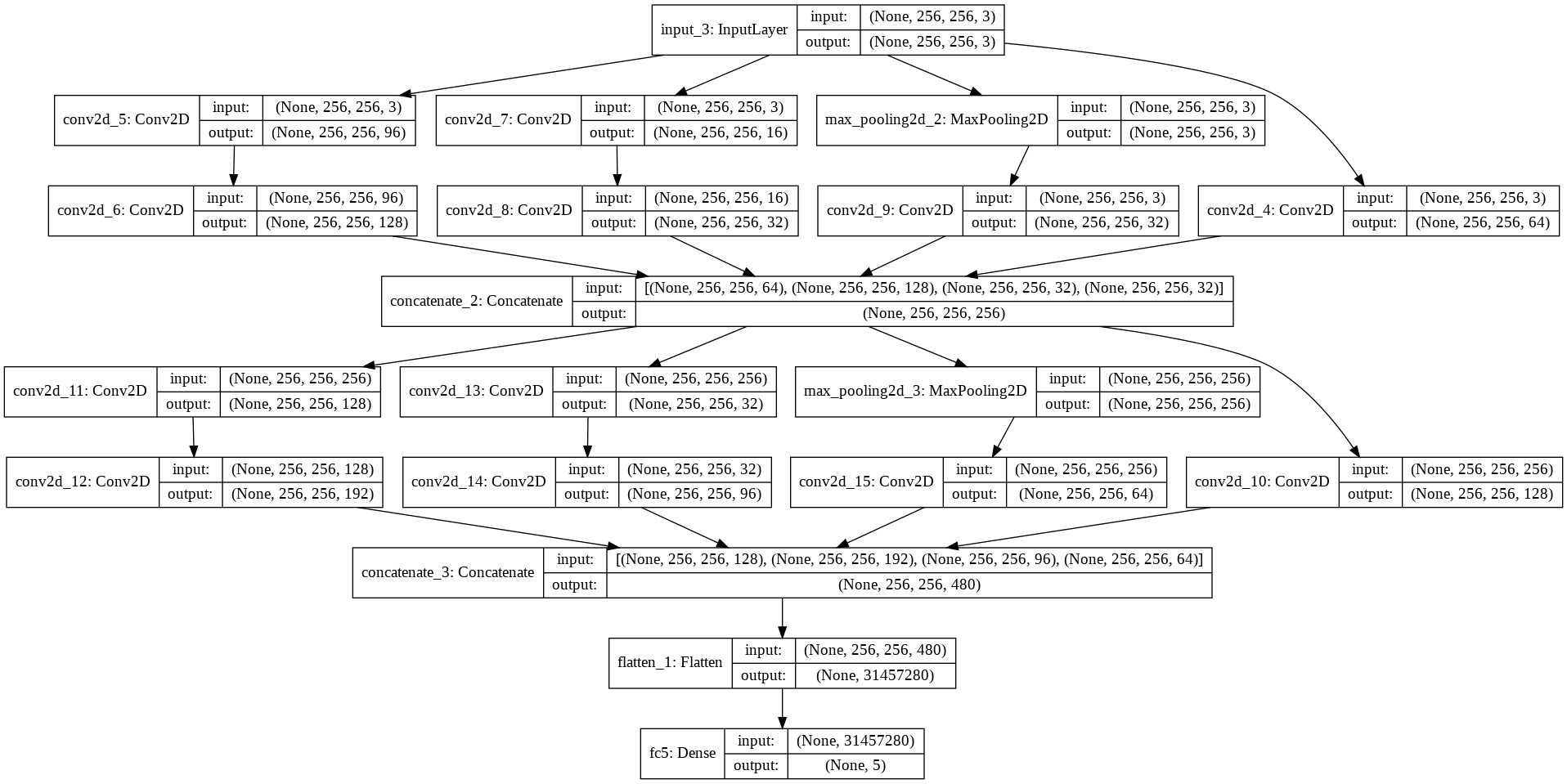
Epoch 2/2

Learning rate: 0.001

176/176 [==============================] - 5586s 32s/step - loss: 0.6046 - acc: 0.7979 - val\_loss: 2.6624 - val\_acc: 0.7762

Epoch 00002: val\_acc improved from 0.76842 to 0.77618, saving model to /content/gdrive/My Drive/ML /data/DiabeticRetinopathy/model\_name

**Resnet – version 2**



Total params: 157,799,525

Trainable params: 157,799,525

Non-trainable params: 0

**Pre Trained Resnet50 model:**

* Loads the model from Keras pretrained model resnet50 with imagenet weights
* Kept the top layers for training as false
* Adds the bottom layer for flattening and set the classifiers for the class labels

Due to time and processing constraints Only one EPOCH was run.

20/20 [==============================] - 808s 40s/step - loss: 0.7778 - acc: 0.7334 - val\_loss: 1.1006 - val\_acc: 0.7368

**Pre Trained VGG16 model:** Also produced similar results.

Using the pre trained models make the runtime faster, but the basic issues still remain on the pre processing required for the image datasets and identifying the individual classes. The model accuracy lies in the low 50 range if we are trying to predict the individual classes accurately, but if the aim was to identify the normal or possible issue, then the accuracy definitely improves in the range of high 80%.

**Conclusion:** Pre Trained model ran faster, but due to only limited number of epoch runs and also since the original training was done against the imagenet dataset, which doesn’t have any of the visually impaired images, the initial accuracy rate is 73%. This is good considering only one epoch run was done with the limited number of images that was available.

**Using the model with mobile application to create the eco System for continuous feedback loop:**

Once the model is trained with the neural network, the model was saved and exposed through REST API, with the mobile interface, to make the independent living possible. This was combined with the image recognition for identifying the friends and family using the cloud cognitive service APIs. Analyzing and uploading the new images, helps to make more data available for the model, which helps in better training of the model to improve the accuracy. The plan proposed require subjects to share the information to get the better results with improved accuracy.

**Conclusion:**

As the fundus images closely vary apart, the model requires multiple convolutional layer to reach the reasonable level of accuracy. The initial model started with 6 convolutional layer wasn’t enough to reach the higher accuracy level. Image augmentation, optimizer selections, image size do play a role, but the important criteria in reaching the higher prediction level is the number of convolutional layers.

By identifying the individual classes, the model accuracy comes in the range of 50%. But by identifying the normal vs possible issue with binary classification, the model performs much better.

It requires lot more image augmentation, cropping and focusing on the noise reduction to help improve the image classification.

But considering overall extent of the work involved, the ability to identify the normal eye vs possible issue is a major step in the right direction.

There are numerous studies done on the image cropping, normalizing the image around Green scale to make the features predominant, and to make the distinction, in order to make the model accuracy better.

**Software packages used:**

* Python 3.6
* Tensor Flow
* Keras
* PyTorch
* Pickle
* Standard python libraries incuding numpy, pandas, matplotlib, seaborn
* The model was run using the Google Colab research laboratory edition

**Bibliography**

1. Background Diabetic Retinopathy. Kellogg Eye Center, www.med.umich.edu/1libr/Ophthalmology/Retina/BackgroundDiabeticRetinopathy.pdf.
2. Marilyn Haddrill, Charles Slonim. All About Vision; What Is Age Related Macular Degeneration? 2017
3. Bourne, R.R.; Stevens, G.A.; White, R.A.; Smith, J.L.; Flaxman, S.R.; Price, H.; Jonas, J.B.; Keeffe, J.; Leasher, J.; Naidoo, K.; et al. Causes of vision loss worldwide, 1990–2010: A systematic analysis. Lancet Glob. Health 2013, 1, e339–e349
4. Wong, T.Y.; Cheung, C.M.G.; Larsen, M.; Sharma, S.; Simó, R. Diabetic Retinopathy. Nat. Rev. Disease Prim. 2016, 2, 16012.
5. Abràmoff,M.D.; Garvin,M.K.; Sonka,M. Retinal imaging and image analysis. IEEE Rev. Biomed. Eng. 2010, 3, 169–208
6. . Porwal, P.; Pachade, S.; Kokare, M.; Deshmukh, G.; Sahasrabuddhe, V. Automatic Retinal Image Analysis for the Detection of Diabetic Retinopathy. In Biomedical Signal and Image Processing in Patient Care; IGI Global: Hershey, PA, USA, 2018; pp. 146–161.
7. Diabetic Retinopathy, Iris Health.com, http://www.irishhealth.com/article.html?con=205
8. Harry Pratta, , Frans Coenenb, Deborah M Broadbentc , Simon P Hardinga,c, Yalin Zhenga, International conference on Medical Imaging understanding and Analysis, MIUA 2016 Loughborough, UK
9. Piotr Skalski, Mysteries of Neural Network, Deep dive into math behind neural network, 2018
10. Muhammad Rizwan, LeNet-5 A Classic CNN Architecture, engMrk, Sep 2018
11. Dataset Source: <https://www.kaggle.com/c/diabetic-retinopathy-detection>
12. Neural Network Model Host RunTime: Google Colab
13. Gandhi, Rohith. “A Look at Gradient Descent and RMSprop Optimizers.” Medium, Towards Data Science, 19 June 2018, towardsdatascience.com/a-look-at-gradient-descent-and-rmsprop-optimizers-f77d483ef08b.
14. Tanlikesmath. “Diabetic Retinopathy, ResNet50, Binary, Cropped.” Kaggle, Kaggle, 30 June 2019, www.kaggle.com/tanlikesmath/diabetic-retinopathy-resnet50-binary-cropped.