Classification of Various Stages of Diabetic Retinopathy through Deep Convolutional Neural Networks

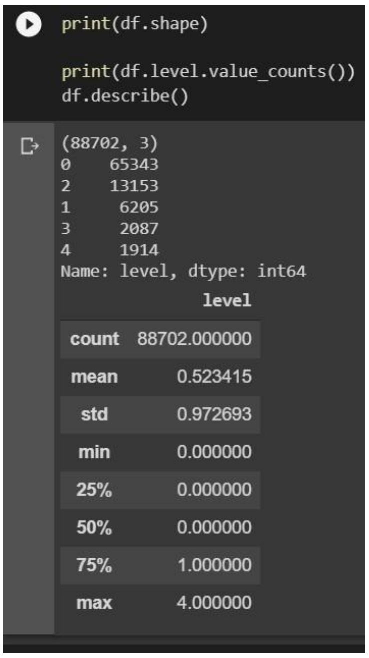
Aditya Jyoti Paul

1. Problem Statement

Diabetic Retinopathy (DR) is a severe complication that provokes retinal vascular damage and is one of the leading causes vision impairment and blindness DR broadly is classified into two stages – proliferative and non-proliferative, non- proliferative (NPDR) where there are almost no symptoms, except a few microaneurysms, while proliferative diabetic retinopathy (PDR) involves a huge number of microaneurysms and hemorrhages, soft and hard exudates, neo-vascularization, macular ischemia or a combination of these, making it easier to detect This project aims to detect NPDR in its early stages through a novel CNN based procedure, which also makes the procedure more explainable and helps in supporting physicians at all stages of the diagnosis This will be able to accurately classify all stages of DR

2. Dataset Description

Kaggle Dataset for Diabetic Retinopathy [1] has been used for this project as the primary source of data There are 88702 images with associated DR levels ranging from 0 to 4 Some dataset specific intuitions are shown in Fig 21 below



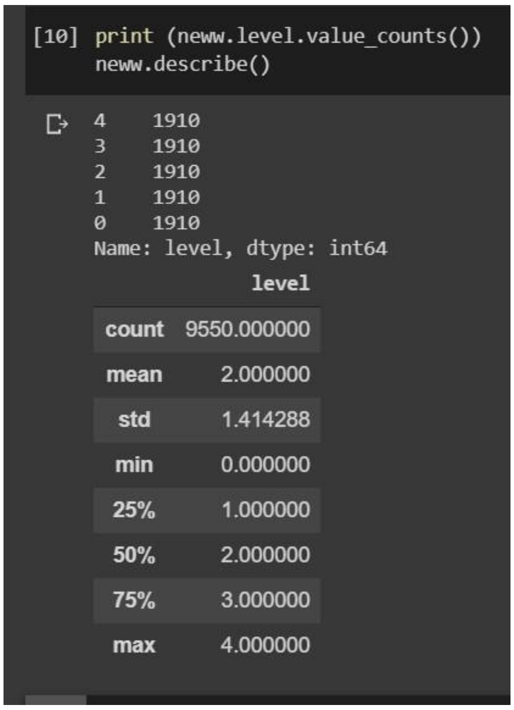
```
print(df.shape)

print(df.level.value_counts())
df.describe()
```

```
(88702, 3)
0    65343
2    13153
1     6205
3     2087
4     1914
Name: level, dtype: int64
```

	level
count	88702.000000
mean	0.523415
std	0.972693
min	0.000000
25%	0.000000
50%	0.000000
75%	1.000000
max	4.000000

Fig 21: Specifics from original dataset

To get rid of the unbalanced outputs, only 1910 images from each level have been use to create a secondary dataset, the specifics of which have been shown in Fig 22 below



```
[10] print (neww.level.value_counts())
     neww.describe()

     4    1910
     3    1910
     2    1910
     1    1910
     0    1910
     Name: level, dtype: int64
```

	level
count	9550.000000
mean	2.000000
std	1.414288
min	0.000000
25%	1.000000
50%	2.000000
75%	3.000000
max	4.000000

Fig 22: Specifics from Balanced Dataset

3. Objective

The objective of this work is the creation of an automated system that can take the image of an eye taken by an optician maybe at a rural area, where healthcare is not very easily available and give them a quick and accurate estimate of whether the patent has Diabetic Retinopathy or not, through Deep convolutional Neural Network

4. Algorithm Explanation

As discussed, the problem is a multi-class single-label classification problem where classes range from 0 to 4, 0 being no DR and 4 being PDR To detect this, Deep CNN model is used Some preliminaries about CNN are mentioned below

A **Convolutional Neural Network (ConvNet/CNN)** is a Deep Learning algorithm which can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other The preprocessing required in a ConvNet is much lower as compared to other classification algorithms While in primitive methods filters are hand-engineered, with enough training, ConvNets have the ability to learn these filters/characteristics in a space and domain-invariant manner

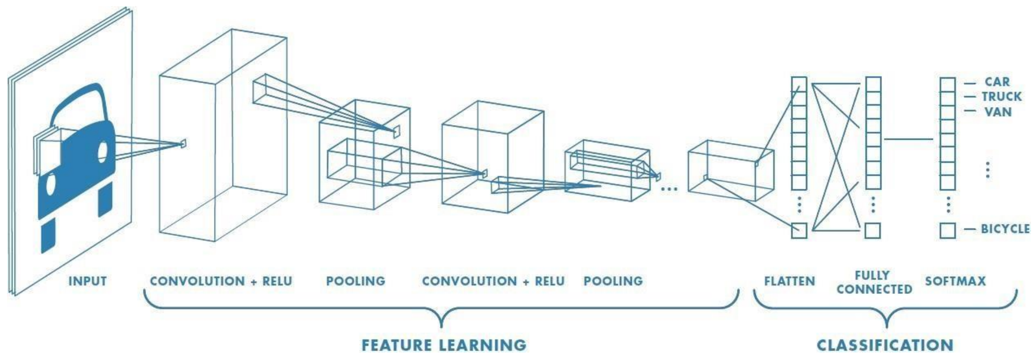


Fig 41 : Pictorial representation of a ConvNet

1) **CONVOLUTIONAL LAYER**: Convolutional layers are the main building blocks of the CNNs. They learn the feature representation of the input images by performing convolutions over the inputs. The convolutional layer consists of several kernels which are used to compute different features from the input images. They ensure that the local connectivity-neurons are connected to a small region of the input which is known as the receptive field. The extracted feature maps are calculated by convolving the input with the kernels and then add the bias parameters to the feature. The convolutional layer has many kernels, and they are applied to the input image to calculate the output feature map. Each kernel is shared by all special locations of the input. The advantage of weight sharing is to reduce the complexity of the model and the training process of the network becomes easier. Mathematically, consider x as the input image, W is the kernel, and b is the bias for the convolutional layer. The feature map z generated from this layer is calculated as:

$$z = Wx + b.$$

Many researchers proposed different types of convolutional layers to improve the feature representation and to learn some kind of invariance.

2) **ACTIVATION FUNCTIONS**: CNNs have some linear components and nonlinear components The activation functions are the nonlinear components which follow the convolutional layers to introduce the nonlinearities to the CNN to detect the nonlinear features and to improve the CNN performance Rectified Linear Unit (ReLU) is one of the most popular activation functions used in CNNs It has been shown that CNNs can be trained efficiently using ReLU. ReLU is defined as:

$$a = \max(z, 0)$$

where z is the input to the activation function and a is the output ReLU keeps the positive part of the input and prunes the negative part to zero Another version of ReLU is leaky ReLU (LReLU) that defines a parameter λ in range (0, 1) to compress the negative part rather than mapping it to zero Mathematically, LReLU is defined as:

$$a = \max(z, 0) + \lambda \min(z, 0)$$

This makes a small and non-zero gradient when the unit is not active (negative value) Exponential Linear Unit (ELU) is another activation function that enables faster learning of the CNN and improves the accuracy of the classification task Like ReLU and LReLU, ELU sets the values to identity and the negative part is used for fast learning and it is robust to noise Mathematically, ELU is defined as:

$$a = \max(z, 0) + \min(\lambda(e\ z - 1), 0)$$

where λ is a controlling parameter to saturate ELU for negative inputs The last activation function we want to talk about is the sigmoid function (σ) Sigmoid function is used often in artificial neural networks to introduce the nonlinearity in the model It takes real numbers and squashes them into range (0, 1) Mathematically, the sigmoid function is defined as:

$$a = \sigma(z) = 1 / (1 + e\hat{}(-z))$$

In general, the activation functions are applied to the output of convolutional layer in the CNN to add the nonlinearity to the output to project the values from some range to a desired range

3) **POOLING LAYER** The purpose of the pooling layer is to ensure the shiftinvariance and lowers the computational burden by reducing the resolution of the feature maps It is usually placed after the convolutional layer It takes the feature map which is generated from the convolutional layer and outputs a single value for each receptive field (pooling window) according to the pooling operation The pooling layer performs max pooling, sum, and mean pooling Figure 4.1 shows different pooling operations Also, there are other versions of pooling layers proposed for some tasks, such as Spatial Pyramid Pooling (SPP) that can generates a fixed length of features regardless of the input size. Fig 4.2 shows some types of pooling.

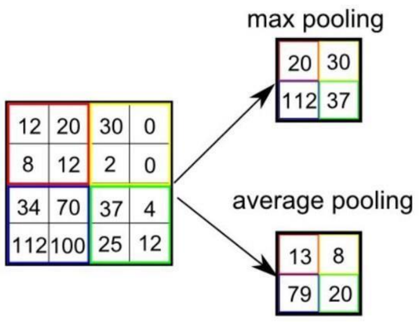


Fig 4.2: Types of pooling

4) **FULLY CONNECTED LAYERS** (FC) In classification tasks, fully connected layers (FC) are used at the end of the CNN after the convolutional layers and the pooling layers Fully connected layers aim to generate specific semantic information The neurons in the fully connected layers have full connection to all neurons in the previous layer It can be considered as a special case of a convolutional layer with the receptive field size is equal to one Usually, dropout is used after the fully connected layers to avoid the CNN from overfitting.

5) **REGULARIZATION** One of the most problematic issues regarding CNN training is overfitting Overfitting happens when the model fits too well to the training dataset, and it cannot generalize to new examples that were not in the training dataset So, overfitting is an unneglectable problem in deep CNNs. There are many proposed solutions to reduce the overfitting effectively, such as Dropout, L1 regularization, and L2 regularization. In deep learning, Dropout is widely used as regularization after the fully connected layers It deletes or deactivates some neurons so that not all connections between the layers are activated at that time during training It can also be applied after the convolutional layers However, it is not preferable to add them in the first layers because dropout causes information to get lost And if the information is lost in the first layers, it will be also lost for the whole network and this will affect the performance of the network During testing time, dropout layers are bypassed and they are not active.

6) **OPTIMIZATION AND LOSS FUNCTIONS** Training a CNN is a problem of global optimization To find the best values of the weights for each layer, a loss function should be selected and minimized To optimize the CNN parameters, Stochastic Gradient Descent (SGD), Momnetum RMSProp, Adam, Nadam etc. are commonly used, here we have used Adadelta.

5. Data Pre-processing Steps

1) **Dataset manipulation**:: Removal of images falling under a specific criterion will be removed, so as to get rid of extremely low-quality images, such as those disturbed by glare. On the other hand, images will be augmented through various techniques like cropping, rotation, flipping, etc, so as to make the model more robust. Batch size for data augmentation should be chosen according to the capacity of the system, this project used a value of 4.

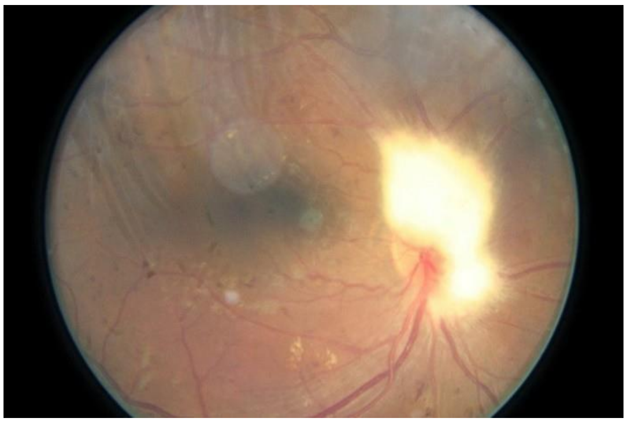


Fig 5.1. Example image to be removed

For eg, Fig. 5.1 should never have made it to the database in the first place, as it involves a blinking artefact, and also the image has too much glare. These images have to be removed; else the eyelash shadow might be misclassified as blood vessels in G channel.

2) **Choosing the appropriate channel**: Green channel was found to be the best channel for training the neural network. It is also apparent to the naked eye how the green channel is superior to the other channels like R, B or L channels.

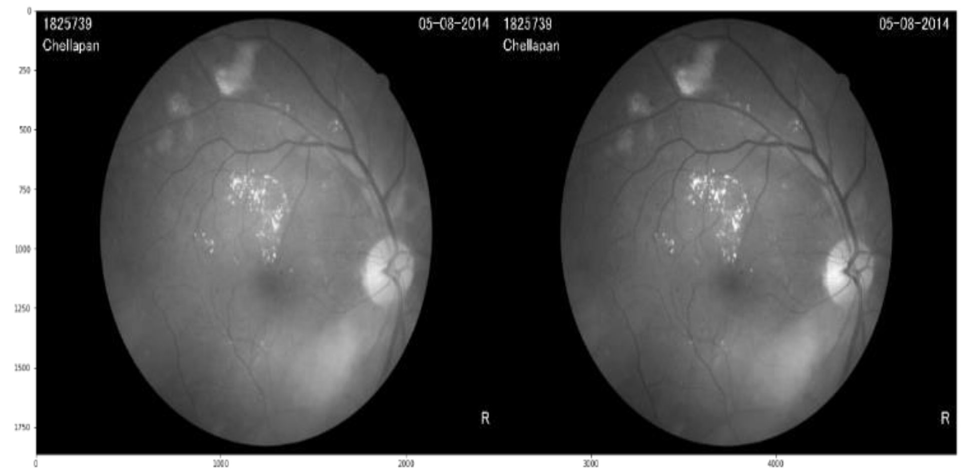


Fig 5.2: Grayscale vs Green-channel image

3) **Adaptive Equalization: CLAHE (Clip Limited Adaptive Histogram Equalization)** has been used to get rid of some of the grain and improve contrast as suggested in [2]. Clip limit (an optimal value of 2 has been chosen) prevents the algorithm from adding extreme features or noise. This image is then sent to the next step.

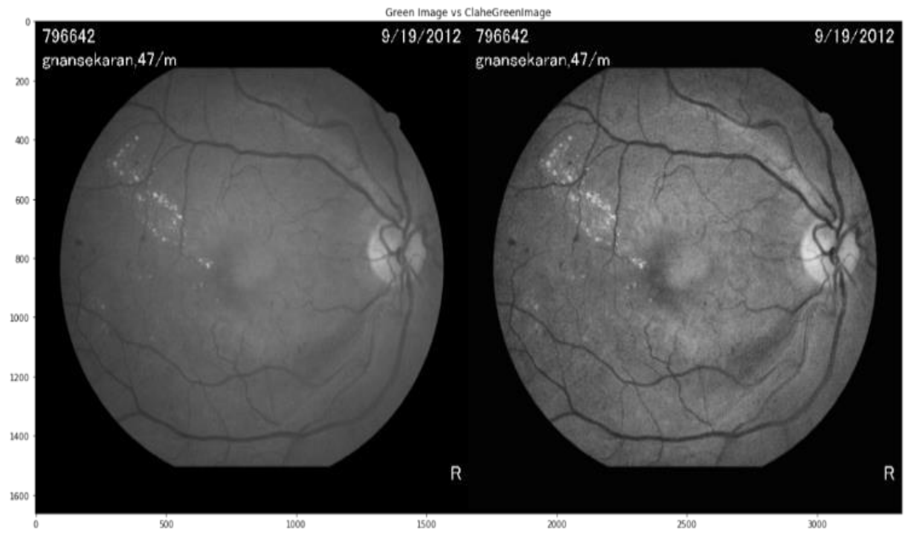


Fig 5.3: Green-channel image before and after CLAHE

4) **Improving Clarity:** A blurred copy of the above image is created with a blur level of 40.

For each pixel p having intensity Xp from the histogram equalization step and the same pixel p in the blurred image having an intensity Yp , Xp is updated accordingly.

$$Xp = 4 * X\ p - 4 * Yp + 128$$

The effect is clear in Fig 5.4 below.

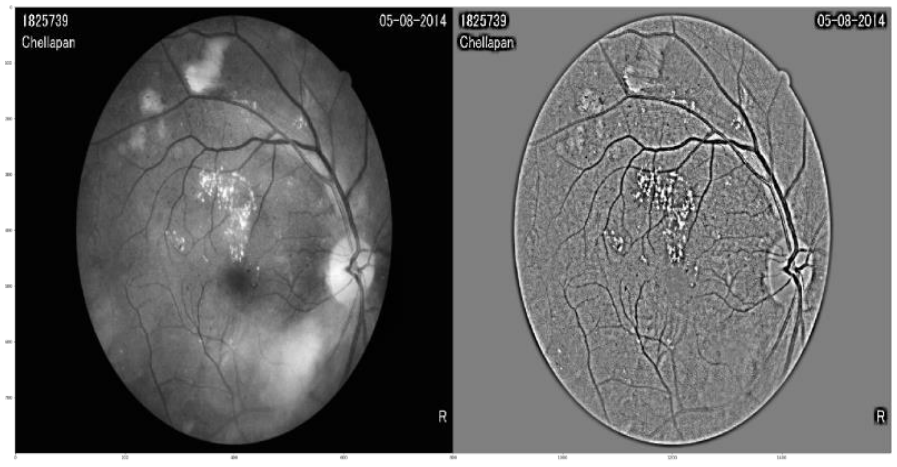


Fig 5.4: Features become more prominent after applying the above method.

6. Implementation

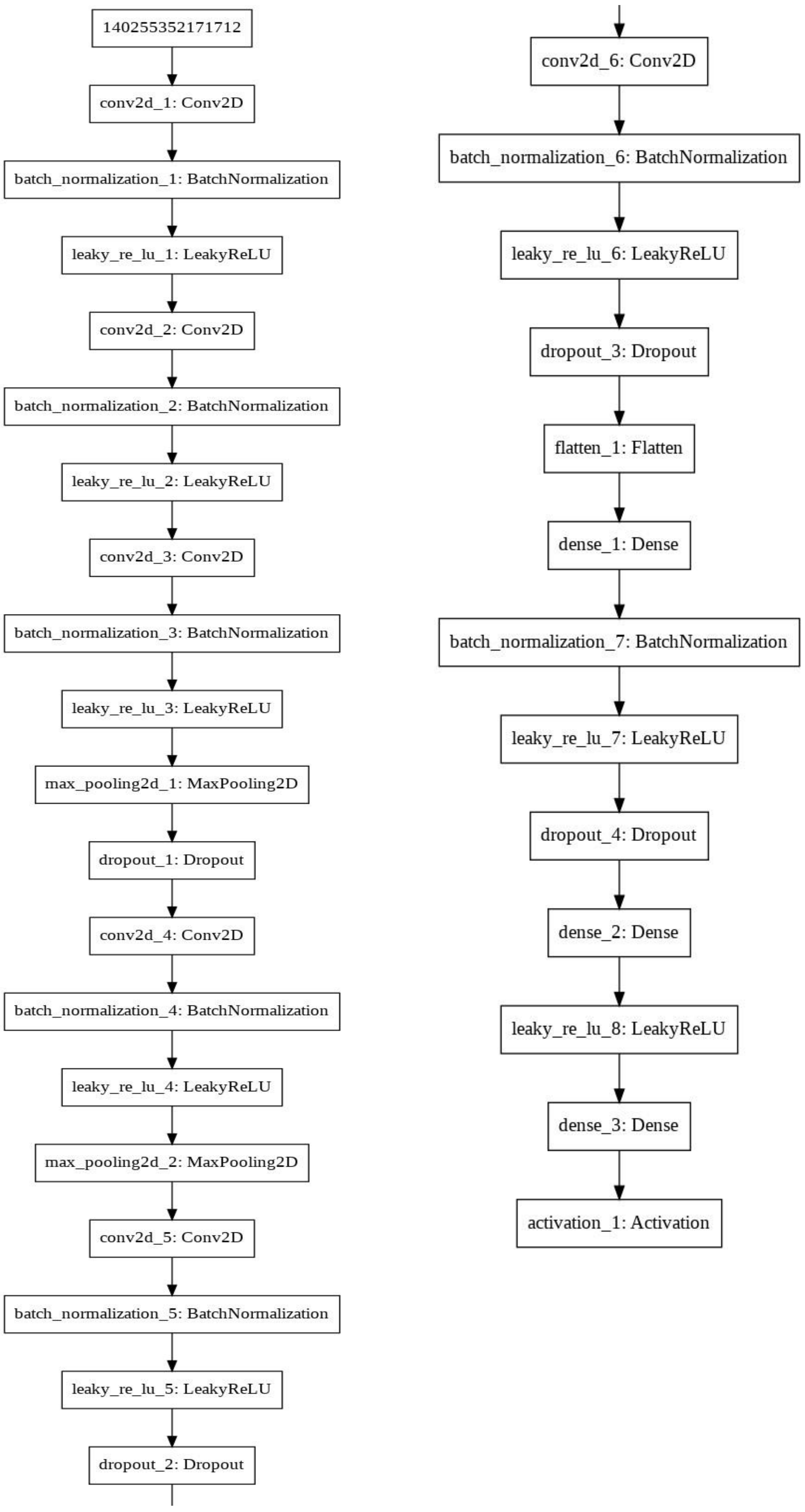
The above-mentioned Deep Convolutional Neural Network has been coded using TensorFlow on Google Colab GPU. The architecture diagram has been described below.

```
Layer (type)                 Output Shape              Param #
==================================================================
conv2d_1 (Conv2D)            (None, 399, 399, 32)      320
__________________________________________________________________
batch_normalization_1 (Batch (None, 399, 399, 32)      128
__________________________________________________________________
leaky_re_lu_1 (LeakyReLU)    (None, 399, 399, 32)      0
__________________________________________________________________ conv2d_2
(Conv2D)            (None, 199, 199, 64)      18496
__________________________________________________________________

batch_normalization_2 (Batch (None, 199, 199, 64)      256
__________________________________________________________________
```

```
leaky_re_lu_2 (LeakyReLU)    (None, 199, 199, 64)      0
_________________________________________________________________ conv2d_3
(Conv2D)            (None, 99, 99, 128)       73856
_________________________________________________________________
batch_normalization_3 (Batch (None, 99, 99, 128)       512
_________________________________________________________________
leaky_re_lu_3 (LeakyReLU)    (None, 99, 99, 128)       0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 49, 49, 128)       0
_________________________________________________________________
dropout_1 (Dropout)          (None, 49, 49, 128)       0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 24, 24, 256)       295168
_________________________________________________________________
batch_normalization_4 (Batch (None, 24, 24, 256)       1024
_________________________________________________________________
leaky_re_lu_4 (LeakyReLU)    (None, 24, 24, 256)       0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 12, 12, 256)       0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 5, 5, 512)         1180160
_________________________________________________________________
batch_normalization_5 (Batch (None, 5, 5, 512)         2048
_________________________________________________________________
leaky_re_lu_5 (LeakyReLU)    (None, 5, 5, 512)         0
_________________________________________________________________
dropout_2 (Dropout)          (None, 5, 5, 512)         0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 2, 2, 512)         2359808
_________________________________________________________________
batch_normalization_6 (Batch (None, 2, 2, 512)         2048
_________________________________________________________________
leaky_re_lu_6 (LeakyReLU)    (None, 2, 2, 512)         0
_________________________________________________________________
dropout_3 (Dropout)          (None, 2, 2, 512)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 2048)              0
_________________________________________________________________
dense_1 (Dense)              (None, 1024)              2098176
_________________________________________________________________
batch_normalization_7 (Batch (None, 1024)              4096
_________________________________________________________________
dropout_4 (Dropout)          (None, 1024)              0
_________________________________________________________________ dense_2
(Dense)              (None, 1024)              1049600
_________________________________________________________________ leaky_re_lu_8
(LeakyReLU)    (None, 1024)              0
```

```
_________________________________________________________________
dense_3 (Dense)              (None, 5)                 5125
_________________________________________________________________
activation_1 (Activation)    (None, 5)                 0
=================================================================
Total params: 7,090,821
Trainable params: 7,085,765
Non-trainable params: 5,056
```

The dataflow diagram is shown below:


140255352171712
conv2d_1: Conv2D
batch_normalization_1: BatchNormalization
leaky_re_lu_1: LeakyReLU
conv2d_2: Conv2D
batch_normalization_2: BatchNormalization
leaky_re_lu_2: LeakyReLU
conv2d_3: Conv2D
batch_normalization_3: BatchNormalization
leaky_re_lu_3: LeakyReLU
max_pooling2d_1: MaxPooling2D
dropout_1: Dropout
conv2d_4: Conv2D
batch_normalization_4: BatchNormalization
leaky_re_lu_4: LeakyReLU
max_pooling2d_2: MaxPooling2D
conv2d_5: Conv2D
batch_normalization_5: BatchNormalization
leaky_re_lu_5: LeakyReLU
dropout_2: Dropout
conv2d_6: Conv2D
batch_normalization_6: BatchNormalization
leaky_re_lu_6: LeakyReLU
dropout_3: Dropout
flatten_1: Flatten
dense_1: Dense
batch_normalization_7: BatchNormalization
leaky_re_lu_7: LeakyReLU
dropout_4: Dropout
dense_2: Dense
leaky_re_lu_8: LeakyReLU
dense_3: Dense
activation_1: Activation

7. Visualization of Results

Training was done in mini batches of size 64 for 100 epochs over the training dataset of 8595 images, for over 6 hrs.

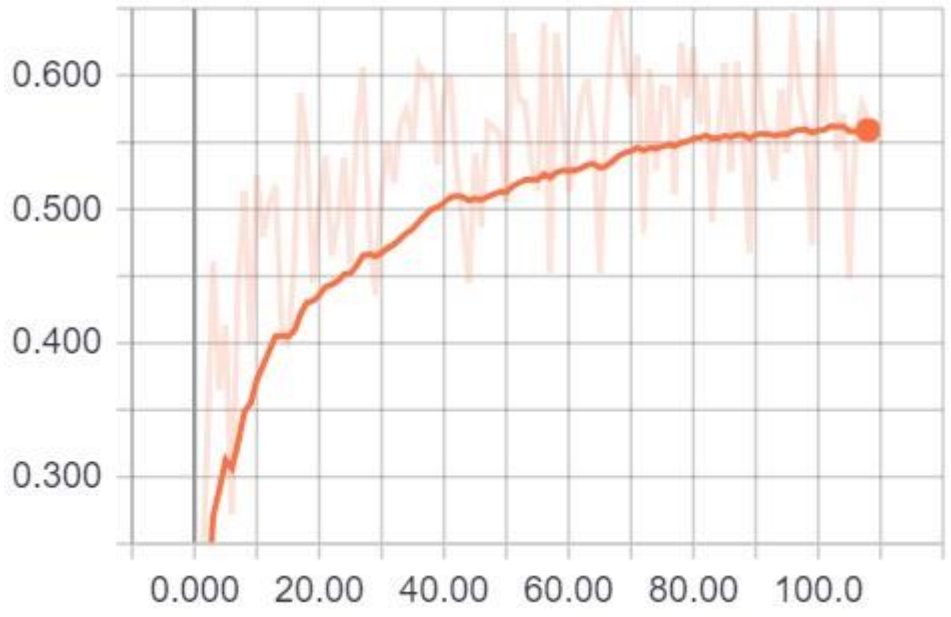


Fig: 7.1: Output from Tensor-board showing validation accuracy against no.of epochs.

8. Performance Metrics

Test Dataset comprised of 955 images. Precision and recall can be calculated for multiclass classification by using the confusion matrix. A confusion matrix is a way of classifying true positives, true negatives, false positives, and false negatives, when there are more than 2 classes. It's used for computing the precision and recall and hence f1-score for multi class problems.

In a confusion matrix (cm) for class x:

- True positive: diagonal position, cm(x, x).
- False positive: sum of column x (without main diagonal), sum(cm(:, x))-cm(x, x).
- False negative: sum of row x (without main diagonal), sum(cm(x, :), 2)-cm(x, x).

All the metrics were calculated with tf.keras.metrics [3].

Accuracy: 0.71232
Precision: 0.81456
Recall: 0.65789
F1 score: 0.72789

Reasonably high accuracy could not be obtained due to limited resources; however, this approach can give higher performance if trained for longer duration.

9. Conclusion

Currently, detection of Diabetic Retinopathy is mostly carried out manually by ophthalmologists which involves checking fundus images and screening, to detect the lesions, which is time consuming, tiring and infeasible for a large number of images. It is crucial to devise an automated system for detection and treatment of DR in its early stages, so that it can prevented sooner and the process be made cheaper. This project takes a step in that direction.

*****