### Methodology:

We have tried out several usual approaches to find the best way to detect diabetic retinopathy. We have explored different datasets with different compression rates of the dataset images. We established that jpeg compression did not store sufficient information while png compression enabled us to achieve higher accuracies with the same dataset.

The main idea behind this was to use pre-trained models and try to mould it to our problem’s requirement. Since we are also try to identify objects in a retina image, pretrained algorithms on imagenet dataset work perfectly well for us.

We have tried exploring the dataset with famous ***VGG16***, ***Resnet50***, ***InceptionV3*** and ***DenseNet121.*** The idea behind it was to segment out the parts of a retina image that can be classified as indicators to Diabetic Retinopathy i.e Aneurysms, Wool and cloud of blood vessels forming in the retina.

After the successful extraction of features from the images, we have tried to train our own GAP layer to classify the features according to us and then give us results in the form of 5 output classes.

0 - No DR

1 - Mild

2 - Moderate

3 - Severe

4 - Proliferative DR

### Experimentation:

We have explored various avenues with the images data. We have taken the data from two separate repositories and performed pre-processing techniques on them.

#### Dataset:

We used two different datasets across 4 different algorithms. Our algorithm is implemented in Kears and was trained on Nvidia 1080-ti 12gb of memory. It took 6 hours to train a single model with 100 epochs. Accuracy, sensitivity and specificity are measured using the confusion matrix for each classification result.

The dataset comprised of 35676 images which were then split first into train and validation where 10 percent of the data was randomly chosen for validation. The train data was split into training and testing data using train\_test\_split from sklearn with the test\_size being 15% of the remaining data with the following code script.

train\_x, valid\_x, train\_y, valid\_y = train\_test\_split(x, y, test\_size=0.15,stratify=y, random\_state=8)

#### Preprocessing:

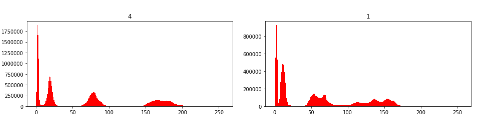
For each image we have had in our dataset we have performed the following image augmentations sometimes and randomly:

1. horizontally flip 50% of all images
2. vertically flip 20% of all images
3. scale images to 80-120% of their size, individually per axis
4. translate by -20 to +20 percent (per axis)
5. rotate by -45 to +45 degrees
6. shear by -16 to +16 degrees
7. use nearest neighbour or bilinear interpolation (fast)
8. convert images into their superpixel representation
9. blur images with a sigma between 0 and 3.0
10. blur image using local means with kernel sizes between 2 and 7
11. blur image using local medians with kernel sizes between 2 and 7
12. sharpen images
13. emboss images
14. search either for all edges or for directed edges, blend the result with the original image using a blobby mask
15. add gaussian noise to images
16. randomly remove up to 10% of the pixels
17. invert color channels
18. change brightness of images (by -10 to 10 of original value)
19. change hue and saturation either change the brightness of the whole image (sometimes per channel) or change the brightness of subareas
20. move pixels locally around (with random strengths)
21. sometimes move parts of the image around

We have also removed any ovals from the dataset and kept only the circular ones so it was easier for the algorithm to learn the difference between diseased eyes.

#### Histogram Equalisation:

We have also plotted histograms for images and found there was little or no difference in between different classes.



Class 1 & Class 4 histogram look almost the same and henceforth it might lead to good training accuracy but we could not produce similar results in validation accuracy.

#### 

#### Training:

For an individual algorithm, we have downloaded its pretrained weights on ‘imagenet’ dataset. We extracted features from those algorithms and then trained our own final layers to classify the image into their separate classes.

We removed the initial input layer of the model to suit our needs of a customized image shape. The image shape was kept (300, 300, 3) with the 3 channels.

After performing image augmentations, the images were fed to the model & the output layer was trained on the pre-labelled dataset.

Model Architecture:

We compared 4-different models for our experimentation which were **VGG16**,**DenseNet121**,**Resnet50**,**Inceptionv3**. Pretrained weights on **imagenet**. The last layer was modified we introduced **GAP-layer**(Global average pooling) to minimize overfitting by reducing the total number of parameters in the model. Similar to **max pooling layers**, GAP layers are used to reduce the spatial dimensions of a 3-d tensor.And then **Dropout** with 0.5 as regularizer.

During training we freeze the first layers and trained last 4 layers including classifier for transfer learning.Feature extractor was trained on imagenet.

After experimentations **DenseNet121** outperformed the other networks with best scores.

Architecture is given as follows.

### 

### Results:

We have been able to achieve state of the art results (Kappa Score) using pretrained models and customizing them to our needs:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| image\_type\model | DenseNet121 | VGG16 | Resnet50 | InceptionV3 |
| .jpeg | 80.08 | 55.831 | 58.23 | 71.28 |
| .png | 90.25 | 79.01 | 73.01 | 80.98 |

We believe the significant jump in the kappa score has been a result of .png files having greater information than .jpeg compressions. Also DenseNet161 performed exceptionally well than the other novel algorithms because of its complex architecture. Thereby it was able to retain more information in its weights while being able to remember the minute details (even pixel wise) which helped return better classification results.

**Before Preprocessing:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Models** | **Accuracy** | **Precision** | **Recall** | **Sensitivity** | **Specificity** | **Kappa Score** | **ROC** |
| DenseNet121 | 84.5 | 76 | 84 | 84 | 27 | 66.25 | 75.23 |
| VGG16 | 55.3 | 69 | 55 | 55 | 33 | 63.01 | 63.3 |
| Resnet50 | 49.2 | 64 | 58 | 58 | 40 | 54.01 | 56.45 |
| InceptionV3 | 41.5 | 58 | 71 | 71 | 37 | 48.78 | 48.78 |

Our accuracies were quite low while using as it is rgb images into model.Then we applied some preprocessing techniques like **transformation,scaling** and also changed optimizer from **SGD** to **Adam** and tweaked **learning rate** also which improved our results exponentially as given below.

**After Preprocessing & Optimization:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Models** | **Accuracy** | **Precision** | **Recall** | **Sensitivity** | **Specificity** | **Kappa Score** | **ROC** |
| DenseNet121 | 90.5 | 93 | 90 | 90 | 87 | 90.25 | 91.4 |
| VGG16 | 79.3 | 91 | 88 | 88 | 70 | 79.01 | 79.78 |
| Resnet50 | 84.7 | 85 | 83 | 83 | 40 | 73.01 | 77.39 |
| InceptionV3 | 85.8 | 88 | 83 | 83 | 37 | 80.98 | 72.10 |

### Conclusions

The research community can continue from here and tweak DenseNet121 architecture with GAP layers while using .png images of the retinal images to further improve the accuracy of the trained model and thus moving one step closer to the early identification of this disease.