Diabetic Retinopathy using Convoluted Neural Networks and TensorFlow

Digital Image processing

ITE1010

C1+TC1

Winter 2018-19

Prabhukumar R

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

Ayush Bishnoi | 17BIS0066

Naren Rai Gandhi | 17BIS0150

Abstract

Most of the current Diabetic Retinopathy methods are using extremely repetitive methods to detect whether there is a presence of diabetes, hence due to the implementation of a neural network we can train a huge dataset and get a very high accuracy for common standard cases. It is an early-stage detection of the fundus images. We aim to achieve a baseline accuracy of 75%+ but this accuracy can easily be increased by training the model for more epochs. After the model is trained, we can extract some basic features from the image, here we are extracting a map of blood vessels to see any broken or severely damaged vessels. Exudations are for the detection of lipids in the retina.

Introduction

Out of the 30 papers we reviewed for this paper, 23 papers were using Machine Learning/Artificial Intelligence/or Deep Learning. Most of these models were reaching an accuracy of 95%+ with at least 4 hours of training. Hence, we knew a baseline of 75% with minimal training is an extremely good baseline. The TensorFlow model using a robust Keras Backend and OpenCV2 for raw image processing. This leads to feeding the raw images in a standard form to the TensorFlow backend where it can generate a model for training. After the model is trained and the correct classifiers are laid out by the CNN, for extracting its features, it is as simple as a Keras CV2 callback. This lets the CNN decide what is properties to extract, and since it has been trained long and accurately enough, the accuracy is extremely good, and the model is reliable.

Image Processing Pipeline

The input of Images: Who are the Victims?

The dataset we are using is the DIARETDB0 dataset and it is being used as it is a well-used and implemented dataset that will also serve as the basis for comparison for model and implementation. The DIARETDB0 database also has images scored based on 5 levels of the fundus in Diabetic Retinopathy images, in ascending order of severity. namely redsmalldots, haemorrhages, hardexudates, softexudates, and neovascularisation. This helps our model train itself as well as validation of the model.

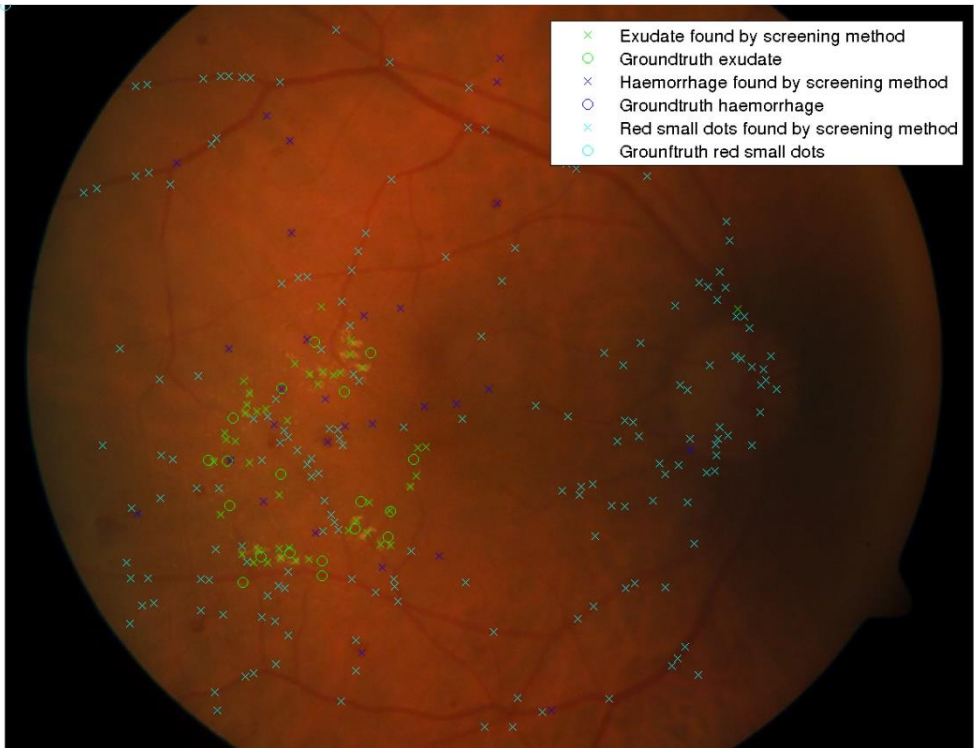

Figure 1: Database Image with marked features

The database has 130 images out of which 104 are being used to train the model and 26 for validation

Pre-processing of Images: Laying the Foundation

The images are all in random sizes that are close to squares, but we need clear grayscale 512x512 images for our CNN to work. (Although it will still work with RGB images these are more reliable and accurate when they are scaled and sized properly. The images are taken in PNG format hence there is minimal compression.

Here is the list of processing that is being carried out on the images:

1. Image is converted into Grayscale using OpenCV
2. Image is resized to 512x512 for the CNN to easily process it.
3. If the image is greater than an 8bit image its histogram is equalized to hold 255 grey levels
4. Image is then thresholded to make it easier for edge detection.
5. The thresholded image then is used to centre the whole image, so that the centre of the image lies roughly in the same area and is the same size for all the dataset images.
6. A bounding box is put around the whole image so that there are no missed spots, if there is clipping of the image it is usually very minimal or not needed as the centred image is more important.
7. Basic image detection is used to find the retinal notch in the image, and this is done in two parts:

a. First, the Hough Circle Transform is applied to find a circle corresponding to the entire eyeball. This circle is subtracted from the thresholded image of the eyeball. Ideally what is left at this point will be either a notch or nothing. Since we will likely pick up "shreds" left from the edges of the subtraction, we contract and dilate at this point to remove leftovers.
b. A region of interest is positioned over where notches appear usually, which is at about the 45-degree mark on the eyeball in the NE quadrant. Blob detection is run over this ROI. If a blob is detected, it is assumed that the blob is a notch.

8. A Hough circle is applied only in the quadrant where the notch can be present. It approximates the notch to be a circle and returns the location of the notch if no notch is present, the image is flipped (as it may be of the other eye) and the notch detection is done again

This pre-processing gives us a normalized image which can be easily worked on by our CNN.

Figure 2: Raw Image

This is an input image, there is less noise in the image and the veins are much more pronounced, the retina is clearer, and the irregularities can be seen clearly. Notch detection was essential to orient the image correctly, this will make sure that there aren't wild variations between images. And the outputted image is a clean 512x512 square with the fundus right in the middle.

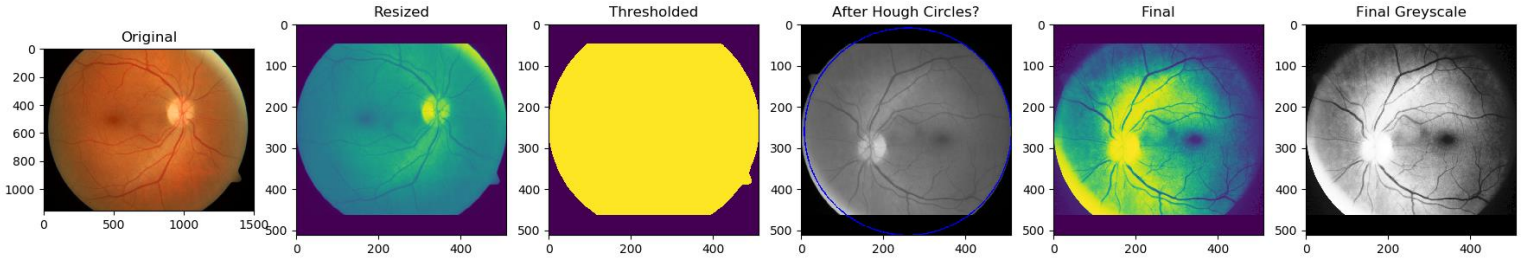

Figure 3: Pipeline Visualized

Deep Learning: Using the Convoluted Neural Network

CNN's are very efficient in Image Processing as the principal that they work on has so many attributes that can be classified into useful/needed etc. Here we are implementing a self-supervised 2-Dimensional CNN in TensorFlow.

In a picture, you have rows and columns of numbers. These numbers are just measuring "how green is this point" or "how blue is this point." Typically, you start with a picture that has some width, some height, and some "depth" where the depth is usually 3 and corresponds to red, green, and blue (or perhaps hue, saturation, and brightness, or other representations).

You could then look at a small patch of this image and ask "how 'blockage' is this patch?" If that small patch (e.g. 7x7 pixels) is composed of a sharp blockage then you would assign a high value to that pixel in your next layer, just like a very red area will have a high value for the red number in its pixels. Then you step over by one pixel and you ask again "how 'blockage' is this patch?" (Noting that you are asking about some of the same pixels that were in the previous patch, too.) You repeat this for patches that cover the whole image. Then you repeat with a new question, perhaps "how 'horizontal line' is this patch" or "how 'smooth texture' is this patch."

This process of coming up with a bit of math to convert one small patch of an input image into a pixel of an output image, then repeating that for every region of the input image, is convolution. Note that while the initial image would likely have 3 channels (red, green, blue) the next layer may have many more layers (smoothness, detecting edges in various orientations, etc). Also, note that the width and height of the output image may be smaller than the input, perhaps because you only check your 7x7 squares every two pixels, or perhaps because you ignore the edges of the image since your 7x7 region would hang off the edge.

That's just one layer. Next, you would start asking more complicated questions like "how 'circle' is this point?" which is an easier question to answer when you know how 'blockage' or 'horizontal line' regions are. You continue building layers like this on top of one another until ultimately, you're asking questions like "How 'elephant' is this picture?" where a picture of an elephant will score very highly but a picture of a rhinoceros or a highway would score lower.

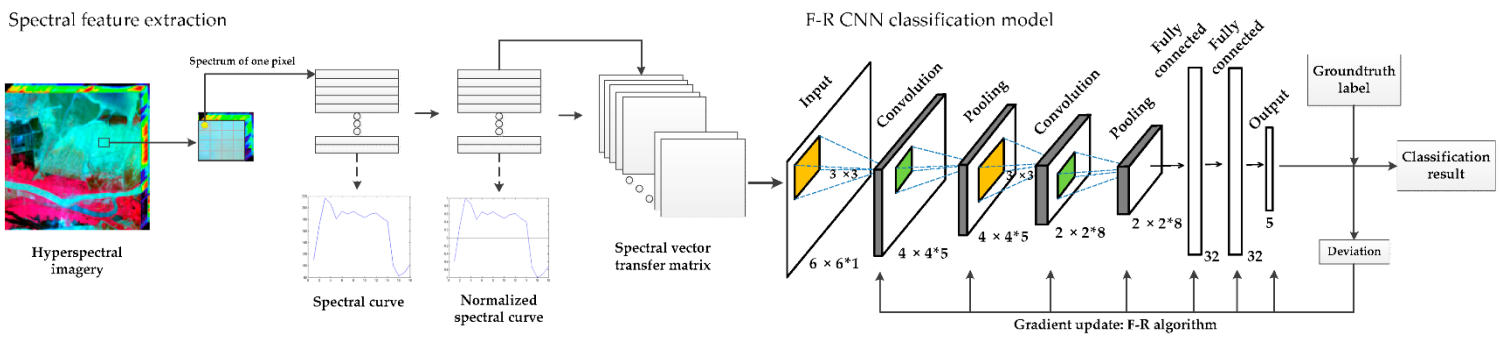

Figure 2: CNN Pipeline

The challenge in all of this is figuring out what questions should be asked at each layer and how to ask them. It isn't that hard to figure out how to check for a blockage, but how do you codify what constitutes an exudate? This is where neural networks come into play. With a neural network, you start with a network that is asking "how 'flarp' is this region" where flarp' is just some random, useless description. You have many layers of this randomness. Then you show the network a picture where you already know what it contains, and you ask the network to figure out what it is. At first it does no better than chance, but by random dumb luck it will get some answers right. When that happens, you modify the math that goes into your individual questions to be a bit more confident, while you reduce the confidence when it gets things wrong. You repeat this a few million times and eventually the 'flarp' classifier gets moulded into, perhaps, a blockage detector, or if it's in one of the later layers perhaps it becomes a 'leathery skin' detector.

Extracting Relevant Features: The Aftermath

1. Blood Vessels: As a simple Keras feature callback, as the attribute is already being defined by our CNN. The attribute is later given a name, and this is the callback that is used.

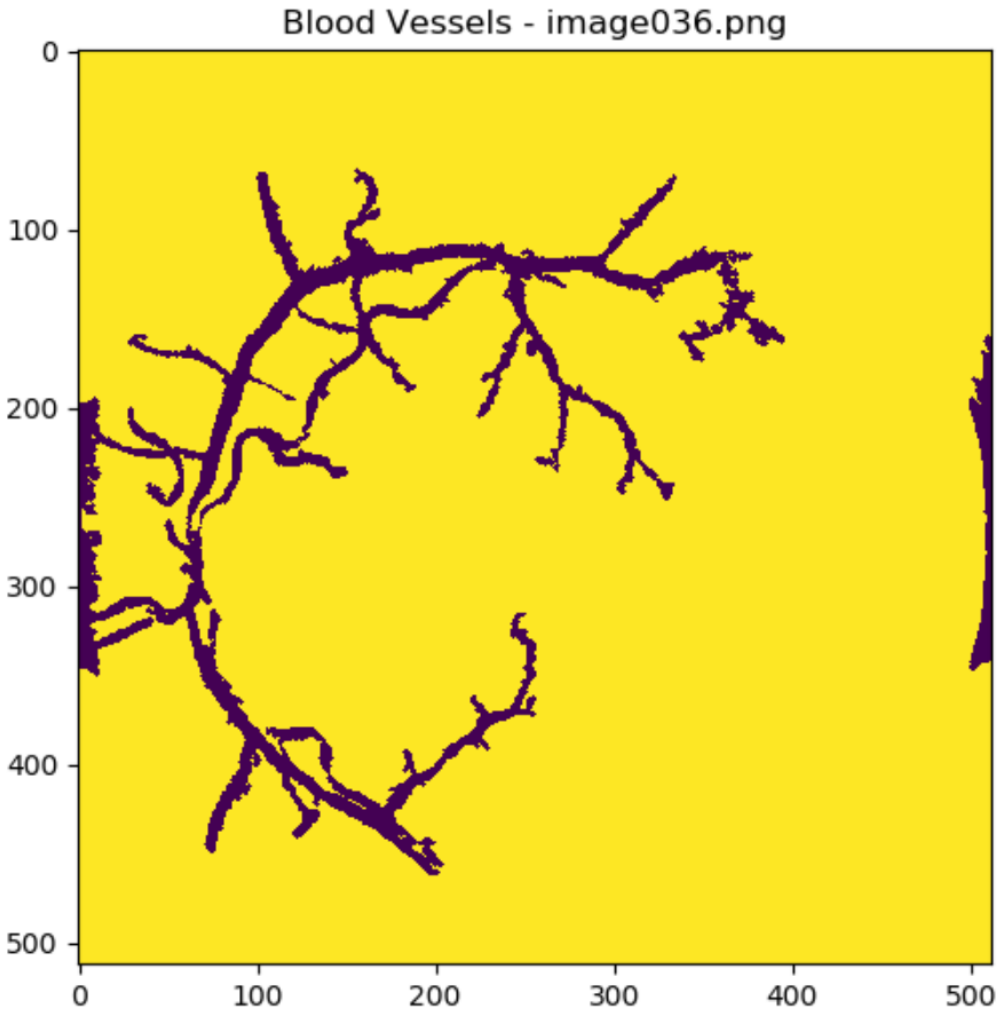

Figure 3: Blood Vessels Extracted

2. Exudations: Same principle as a feature callback, but this one is defined differently.

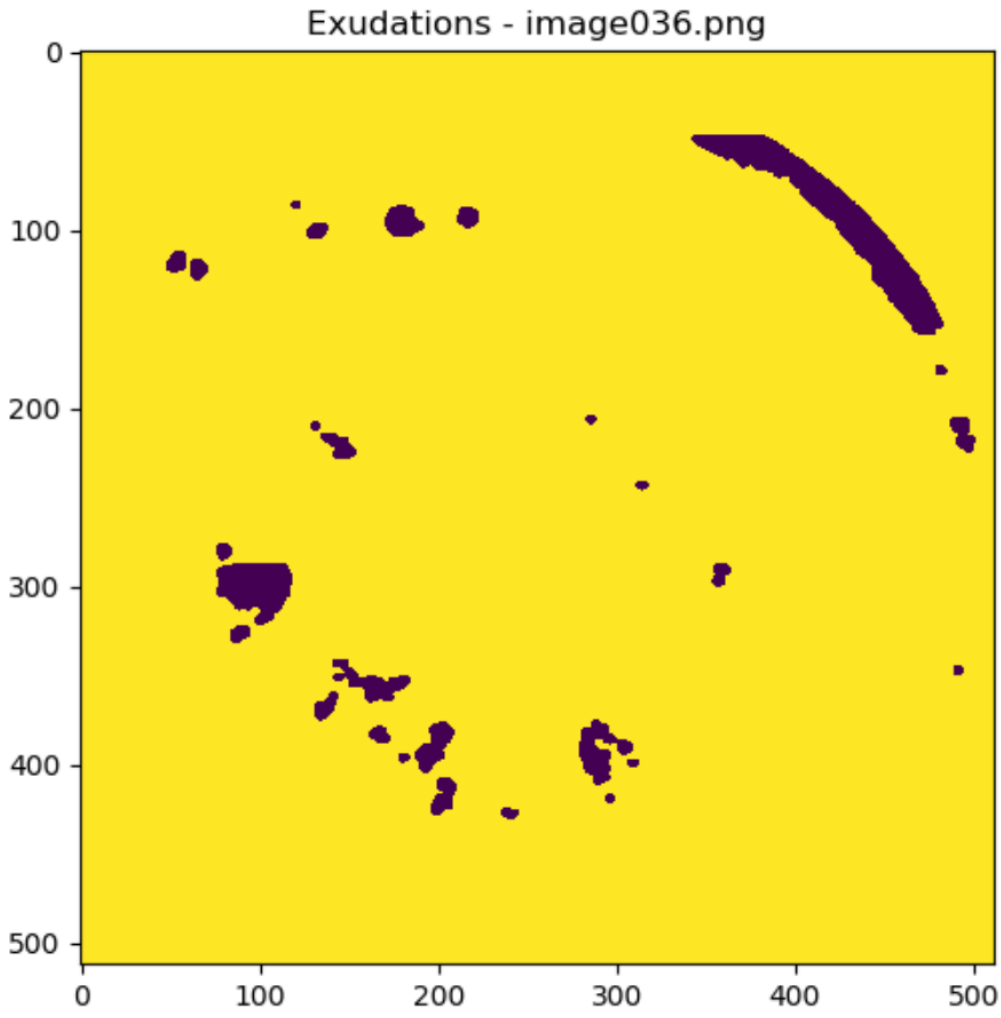

Figure 4: Exudations Extracted

Results

When the model is trained with the following conditions:

1. Epochs = 200
2. Batch size = 32
3. Randomize = True

This gave us a baseline of 80.216%, which is greater than the desired 75% accuracy. If we use a bigger dataset and train it more intensely, the accuracy can climb upwards of 95%. Further we compare our implementation/approach with the execution of others

Comparison: Directly With a CNN[1]

Implementation

In Convolutional Neural Networks for Diabetic Retinopathy, their dataset was of over 80,000 images using Kaggle, for building their Neural Network, the CNN was initially pre-trained on 10,290 images until it reached a significant level. This was needed to achieve a relatively quick classification result without wasting substantial training time. After 120 epochs of training on the initial images, the network was then trained on the full 78,000 training images for a further 20 epochs. Neural networks suffer from severe over-fitting, especially in a dataset such as theirs in which most of the images in the dataset are classified in one class, that shows no signs of retinopathy. To solve this issue, they implemented real-time class weights in the network. For every batch loaded for back-propagation, the class-weights were updated with a ratio respective to how many images in the training batch were classified as having no signs of DR. This reduced the risk of over-fitting to a certain class to be greatly reduced. The network was trained using stochastic gradient descent with Nesterov momentum. A low learning rate of 0.0001 was used for 5 epochs to stabilise the weights. This was then increased to 0.0003 for the substantial 120 epochs of training on the initial 10,290 images, taking the accuracy of the model to over 60%, this took circa 350 hours of training. The network was then trained on the full training set of images with a low learning rate. Within a couple of large epochs of the full dataset the accuracy of the network had increased to over 70%. The learning rate was then lowered by a factor of 10 every time training loss and accuracy saturated.

Results

5,000 images from the dataset were saved for validation purposes. Running the validation images on the network took 188 seconds. For this five-class problem we define specificity as the number of patients correctly identified as not having DR out of the true total amount not having DR and sensitivity as the number of patients correctly identified as having DR out of the true total amount with DR. We define accuracy as the number of patients with a correct classification. The final trained network achieved 95% specificity, 75% accuracy and 30% sensitivity. The classifications in the network were defined numerically as 0 - No DR 1 - Mild DR 2 – Moderate DR 3 - Severe DR 4 - Proliferative DR.

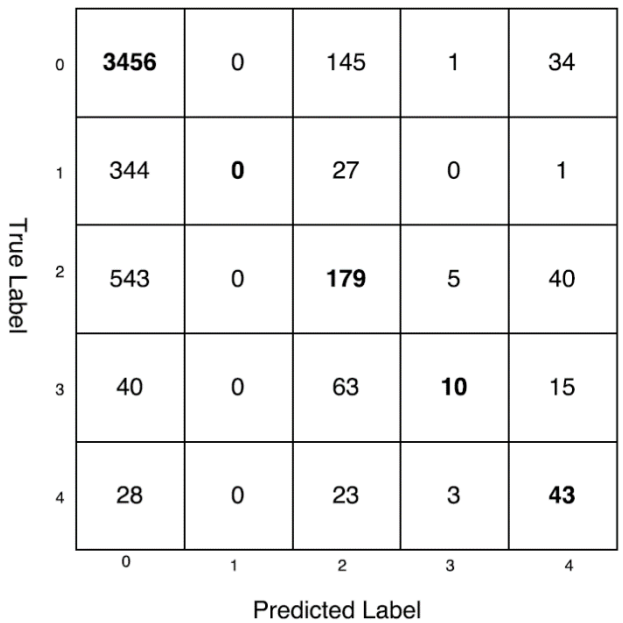

Figure 5: Labels in a 3D CNN

Differences in approach

1. Due to their implementation of a 3D CNN and instead of using inertia as the inherent property, they used the Nesterov Momentum which is much more taxing on the hardware rather than the 2D CNN we used.
2. Their dataset contained 80,000 images, this kind of a dataset and their total training time of 350 hours is the key reason why they have such high final accuracy.

A 3D CNN is a much more difficult approach to the problem that we are trying to solve, but for testing and conceptual purposes, this is an extremely good concept and works at a base level.

Comparison: Convention ML Approach[2]

Implementation

The main concept of this work is to apply methods often used in the field of face recognition to the analysis of retinal fundus images. In face recognition, one usually tries to build a basic set of images to represent the original set, and then, identify the class (i.e., the individual) to which a face image belongs according to its projection on the basis set. A similar methodology was adopted in this work. The basis images were built from a training set of 300 images, namely the first sub-set of each of the 3 main sets of the Messidor database. They thus assure that the 3 different ophthalmologic departments contribute equally to the training set and that both images with and without pupil dilation were included. The popular approaches in face recognition tasks were tested, namely the PCA which will be briefly described.

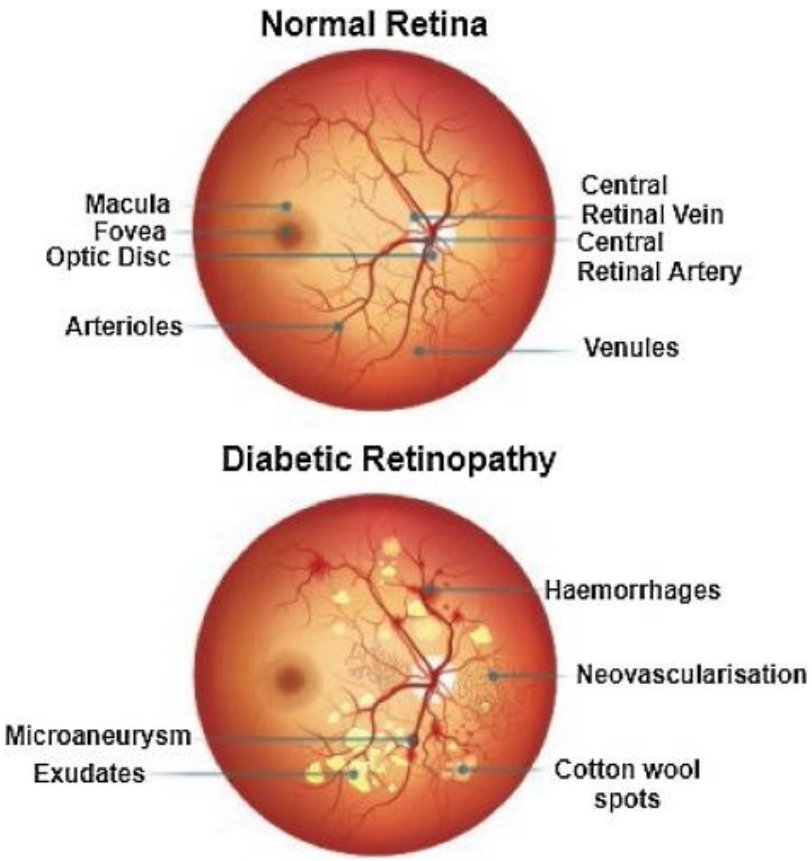

Figure 6: Variables in a Normal and Effected Eye

Principal Component that are related to each other like this, you can perform a mathematical transformation on this data to turn the related variables (length, height, weight) into some unrelated variables (x, y, z). These

unrelated variables don't really have any physical meaning like length, height and weight - they are a mathematical abstraction.

The reason you might want to do this is that once you get your x, y, z variables, you may be able to ignore the z (or the y and the z) variables and use only the x variable if your original data was very related to each other. It allows you to reduce the number of variables. This is helpful when you have massive amounts of data and huge numbers of variables, but the limitation here arises from the fact that you will have to hardcode the initial variables, this is not ideally wanted because then this limits our approach to more variables, as we cannot hardcode all the variables. Here the writers implemented the main variables as Microaneurysms/Exudates, Cotton Wool Spots/Neovascularisation/Haemorrhages were not considered, and their model used this for the basis of PCA

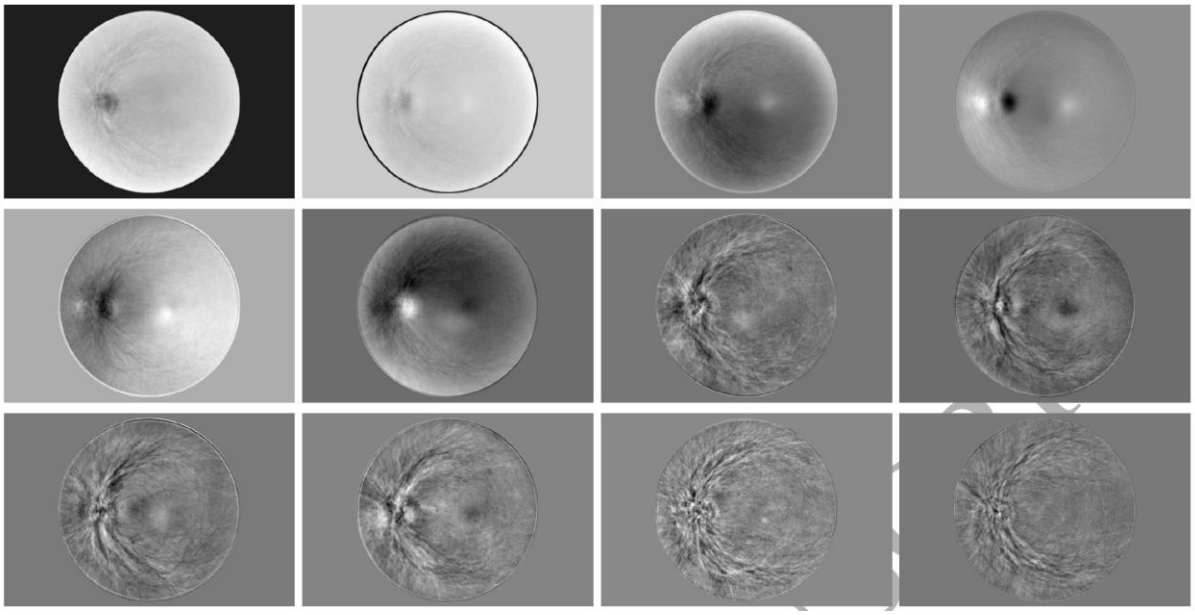

Figure 7: Hardcoded Features

Results

The analysis was then extended to incorporate local retinal features — exudates and micro-aneurysms — using a modular approach developed for face-recognition, 810 which improved considerably the performance of our system. Moreover, both holistic texture and local features from micro-aneurysms and exudates were selected for their relevant diagnostic information, which validates the modular model herein introduced. They achieved an accuracy of ~55%, this is mostly due to adapting PCA for Diabetic Retinopathy.

Differences in approach

Using an algorithm that is used for face recognition, and modifying it and successfully implementing it to produce an accuracy is definitely an unique yet conventional approach that is no match for our CNN, although their model did have hard-coded variables that were more accurate than what our CNN produced, but we hade more

variables and more features were taken into account, this lead to a better accuracy than what a PCA can ever achieve.

Comparison: Using an AlexNet CNN[3]

Implementation

In the present research, we have employed Alexnet architecture because it has better computational ability to address complexities than other architectures. CNN works with three-dimensional (length, width, and depth) values. For a normal colour image, the dimensions of the image are denoted by a, b, and c, where 'a' stands for the length of the colour image, 'b' represents the breadth of the colour image, and 'c' denotes the number of colour channels present in the input. The primary layer receives the input image, and after processing the final layer of the architecture provides the prediction. Generally, Alexnet architecture has eight layers, in which the first five layers are convolutional and maximum pooling layers, followed by three layers fully connected to the neural network.

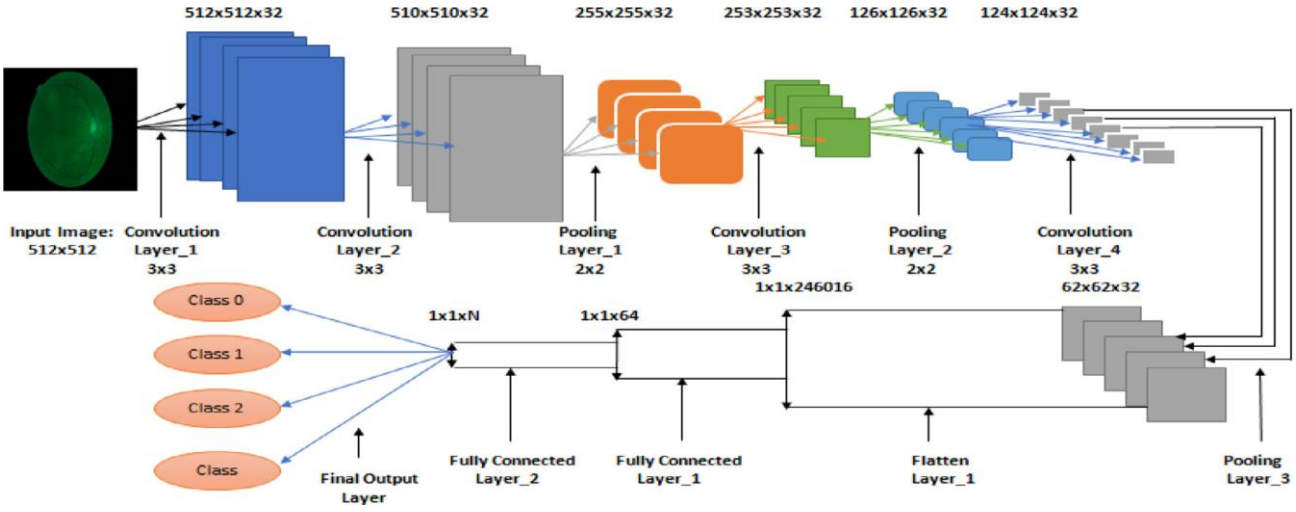

Figure 8: AlexNet Pipeline

Results

Training the architecture has been carried out with 710 fundus images from the given dataset. The performance of the proposed architecture has been evaluated by testing it on 303 fundus images. It is found that among the 303 images of fundus tested with the application of the proposed AlexNet architecture, 107 images are images of a healthy retina, 58 images belong to DR stage 1, 86 images pertain to DR stage 2, and 53 fundus images fall in the category of DR stage 3. This resulted in an accuracy of 96%.

Differences in Approach

The modified AlexNet that was employed here is one of the best ways to carry of a CNN in the detection of Diabetic Retinopathy, our model can probably reach the same level of accuracy, but this would require much more processing power and training

cycles as compared to the AlexNet CNN. The AlexNet competed in the ImageNet Large Scale Visual Recognition Challenge on September 30, 2012. The network achieved a top-5 error of 15.3%, more than 10.8 percentage points lower than that of the runner up. There is no way we can achieve that level of speed and accuracy until we don't hardcode and supervise the data set our self.

Conclusions

Through this project, we learned that feeding processed images into a CNN for it to train itself with a large dataset yields us a very good baseline to work upon. The images we processed for direct input into the Keras Backend and using self-supervised CNN model a very accurate (80%+) model was created which can easily be improved upon.

This implementation is new, but not unique, comparing it to other implementations in the same problems shows us that in the field of Deep Learning, this is the baseline accuracy, from here it is only up. Due to this, there is a lot of room for improvements in our model, (1). like hardcoding the variables would yield a much higher accuracy, (2) using a 3D CNN, although more difficult to implement and understand, gives us more data and layers to work with, this leads to a more accurate model, and finally, (3) using a better CNN, example being the AlexNet we compared it too.

Performance Matrix

Validation Split	Training Accuracy	Training Loss
0.2	80.769%	1.702
0.4	70.234%	3.755
0.6	56.153%	7.488
0.8	40.799%	9.718

Acknowledgements

This project was a learning experience both in the field of Digital Image Processing for our pre-processing pipeline, as well as the basis of how neural network works. We got a good understanding of TensorFlow too, and the basis of what deep learning requires.

A true learning experience that we did not expect to learn was the art of writing papers, including this review paper and the literature review before this, conveying your learning/implementation in words is truly an art.

References

1. Pratt, Harry, et al. "Convolutional neural networks for diabetic retinopathy." *Procedia Computer Science* 90 (2016): 200-205.
2. Frazao, Luis Bastos, Nipon Theera-Umpon, and Sansanee Auephanwiriyakul. "Diagnosis of diabetic retinopathy based on holistic texture and local retinal features." *Information Sciences* 475 (2019): 44-66.
3. Shanthi, T., and R. S. Sabeenian. "Modified Alexnet architecture for classification of diabetic retinopathy images." *Computers & Electrical Engineering* 76 (2019): 56-64.