Classification Of Diabetic Retinopathy Stages Using Deep Learning

DISSERTATION SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

Master of Technology
in
Computer Science

by

Munendra Singh
[Roll No: CS-1615]

under the guidance of

Dr. Sushmita Mitra
Professor
Machine Intelligence Unit

Indian Statistical Institute
Kolkata-700108, India

July 2018

Declaration of Authorship

I, Munendra Singh, declare that this thesis titled, **'Classification Of Diabetic Retinopathy Stages Using Deep Learning'** and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: _____

Date: _____

Certification

This is to certify that this thesis titled **"Classification Of Diabetic Retinopathy Stages Using Deep Learning"** submitted by **Munendra Singh**, embodies the work done under my supervision.

Prof. Sushmita Mitra,
Machine Intelligence Unit,
ISI Kolkata

Abstract

Diabetic Retinopathy (DR) is the leading cause of blindness in the working-age population of the developed world and is estimated to affect over 93 million people.Detecting DR is a time-consuming and manual process that requires a trained clinician to examine and evaluate digital color fundus photographs of the retina.In this report, we have proposed three different methods for classifying DR Images. The first method uses Convolutional Neural Network. The Second method uses a pre-trained 2D VGG16 ConvNet model for feature extraction. The third method uses Capsule Network. We discuss merits and demerits of each method.

Acknowledgements

I would like to express my heartfelt gratitude to my supervisor **Prof. Sushmita Mitra** for encouraging me to pursue research in neural networks and for her constant guidance and support.

I would like to thank **Mr. Subhashis Banerjee** for his valuable suggestions and discussions. I would also like to thank my family members who supported me morally, financially and physically.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Diabetic Retinopathy

People with diabetes can have an eye disease called diabetic retinopathy. This is when high blood sugar levels cause damage to blood vessels in the retina. These blood vessels can swell and leak. Or they can close, stopping blood from passing through. Sometimes abnormal new blood vessels grow on the retina. All of these changes can steal your vision.

1.1.1 Stages of Diabetic Eye Disease

There are two main stages of diabetic eye disease.
1. NPDR (non-proliferative diabetic retinopathy)
2. PDR (proliferative diabetic retinopathy)

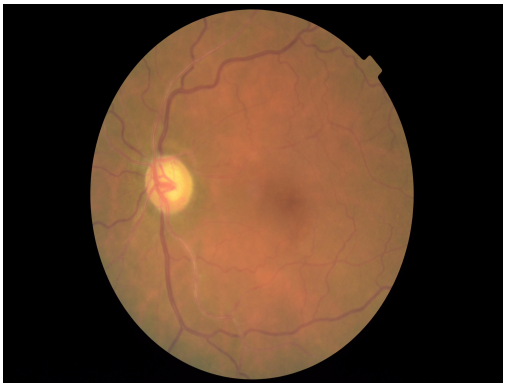
1.2 The Dataset

We are provided with a 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).

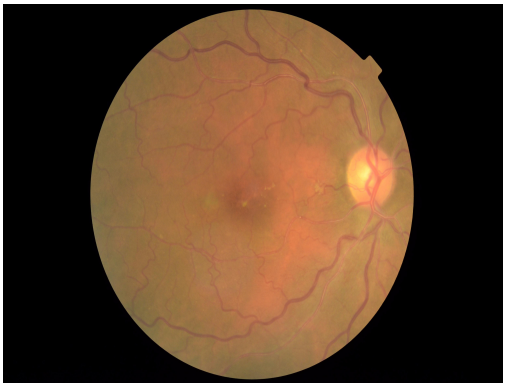
A clinician has rated the presence of diabetic retinopathy in each image on a scale of 0 to 4, according to the following scale:

1. No DR

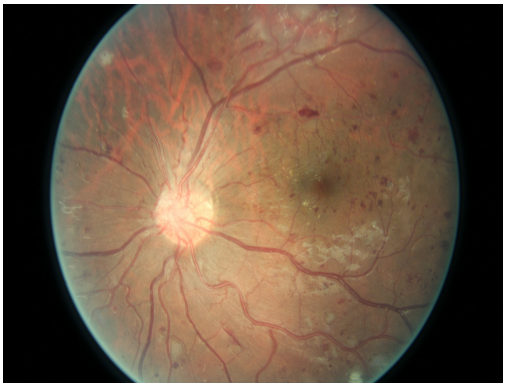
2. Mild
3. Moderate
4. Severe
5. Proliferative DR



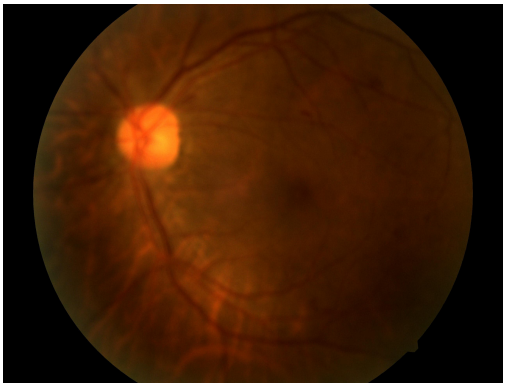
(A) No DR



(B) Mild



(C) Moderate



(D) Severe

(E) Proliferative DR

FIGURE 1.1: Images of Different Classes in the Dataset

1.3 Challenges

The images in the dataset come from different models and types of cameras, which can affect the visual appearance of left vs. right. Some images are shown as one would see the retina anatomically (macula on the left, optic nerve on the right for the right eye). Others are shown as one would see through a microscope condensing lens (i.e. inverted, as one sees in a typical live eye exam). There are generally two ways to tell if an image is inverted:

- It is inverted if the macula (the small dark central area) is slightly higher than the midline through the optic nerve. If the macula is lower than the midline of the optic nerve, it's not inverted.
- If there is a notch on the side of the image (square, triangle, or circle) then it's not inverted. If there is no notch, it's inverted.

Chapter 2

Related Work

2.1 Automatic detection and classification of diabetic retinopathy stages using CNN

Many deep learning based DR classifiers has been published in the last few years. In [1], a deep learning classifier has been published for the prediction of the different disease grades. They used the Kaggle dataset provided by EYEPACS. They achieve around 85% accuracy for the five class classification and 95% accuracy for the two class classification(DR or no DR). They used 512 * 512 images for the training purpose. For augmentation purpose they used rotation of images by 90 and 180 degrees. The following table shows the results they obtained by using their proposed method:

Class Label	Precision	Recall	f1-score
class0	0.88	0.95	0.91
class1	0.40	0.39	0.39
class2	0.70	0.42	0.52
class3	0.36	0.56	0.43
class4	0.62	0.49	0.54

Table 2.1: Automatic detection and classification of diabetic retinopathy stages using CNN Result

2.2 Application of Higher Order Spectra for the Identification of Diabetes Retinopathy Stages

In[2], they have created an automated method for identifying the five classes. Features, which are extracted from the raw data using a higher order spectra method, are fed into the SVM classifier and capture the variation in the shapes and contours in the images. This SVM method reported an average accuracy of 82%, sensitivity of 82%, and specificity of 88%.

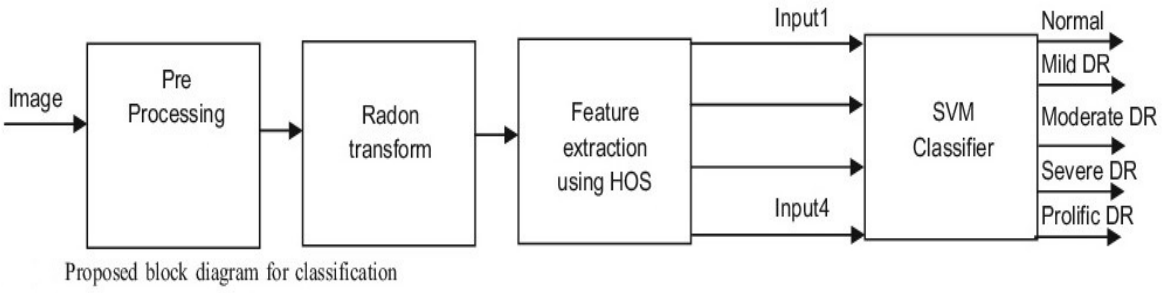


FIGURE 2.1: Proposed Block Diagram for classification

In this work, they used 300 retinal photographs of mild NPDR, moderate NPDR, severe NPDR, PDR, and also normal cases. These data were provided by the National University Hospital, Singapore.Images, taken by Ziess Visucam lite fundus camera inter- faced to a computer, were stored in 24-bit Joint Pho- tographic Experts Group format with an image size of 256 * 256 pixels.

Chapter 3

Proposed Method

3.1 Data Preprocessing

We have used Kaggle dataset, which contains 35,126 images, for diabetic Retinopathy. The provided dataset has images of different dimensions. So we have used various techniques to preprocess the dataset. The following techniques used to preprocess the dataset:

1. Cropping
2. Reshaping
3. Contrast Improvement

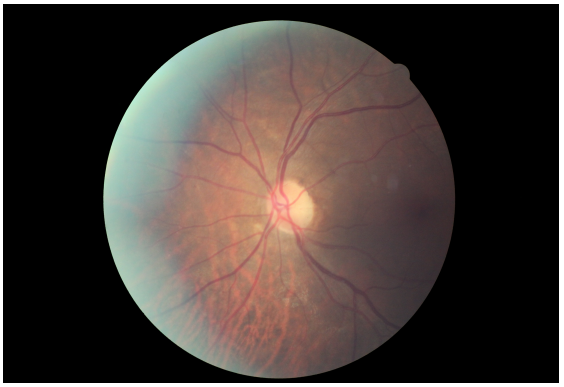


FIGURE 3.1: Original Image

3.1.1 Cropping

Images in the dataset are having black portion around the actual image of eye. Black portion affect the performance of the model because it contains no information. So we need to crop this black portion from the image.

3.1.2 Reshaping

Images in dataset are of different size. So to make the images of same size reshape[3] the images. For different models we used different size of images. We used images of size 192*192 for Capsule Network, 256*256 for VGG16 and 512*512 for Convolutional Network.

3.1.3 Contrast Improvement

For contrast Improvement we use CLAHE(Contrast Limited Adaptive Histogram Equalization)[4]. Ordinary AHE tends to overamplify the contrast in near-constant regions of the image, since the histogram in such regions is highly concentrated. As a result, AHE may cause noise to be amplified in near-constant regions. Contrast Limited AHE (CLAHE) is a variant of adaptive histogram equalization in which the contrast amplification is limited, so as to reduce this problem of noise amplification.

(A) Cropping and Reshaping (B) After CLAHE

FIGURE 3.2: Images after Pre-Processing

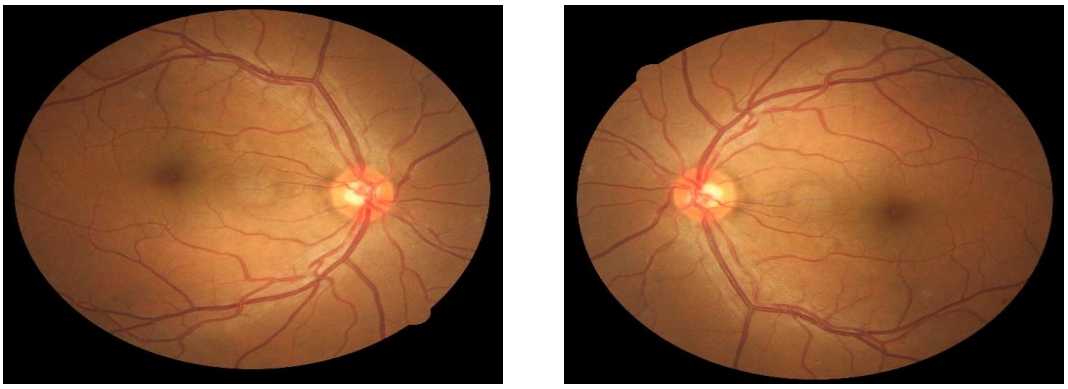
3.2 Data Augmentation

The data-set provided by Kaggle is imbalance. So to make it balance we need data augmentation. We augment only those classes which are having less images. So after augmentation all the classes have more or less same number of images.

We used the following techniques for data augmentation:

1. Flipping Horizontally
2. Flipping Vertically
3. Rotation

By using these techniques our model is more robust for different orientations.



(A) Flipping Horizontally (B) Flipping Vertically

FIGURE 3.3: Images After Horizontal and Vertical Flipping

(A) Rotation 90 degree (B) Rotation 180 degree

FIGURE 3.4: Images After Rotation

We have used images of size 192*192, 256*256, 512*512 for three proposed methods.

3.3 Convolutional Network for Diabetic Retinopathy

They are made up of neurons that have learn-able weights and biases. Each neuron receives some inputs, performs a dot product and optionally follows it with a non-linearity. The whole network still expresses a single differentiable score function: from

the raw image pixels on one end to class scores at the other.And they have a loss function on the last (fully-connected) layer.

3.3.1 Discussion

In this architecture initially we use a kernel of (7*7) because it will extract simple features from the image and we use stride of 2 for first convolutional layer[5]. Conventionally this is better to use small kernel size so that it can extract more information from the image. But initial convolutional layer extract very simple features from the image so we use kernel of size (7*7) with a stride of 2. For rest of network for convolutional layer we use kernel size of (3*3) with a stride of one so that we can extract more information and more complex features of the image. For pooling we use max pooling of kernel size of (3*3) with a stride of 2 so that we can reduce the size of the output of previous layer so that we can reduce the number of parameters by extracting important information by using maximum value around a pixel. To control the overfitting we use different techniques like batch normalization[6], dropout[7] etc. we initialize the kernels as default which is using glorot_uniform method. The kernels initialization is not important because we use the batch normalization between Conv2D layer and activation layer because training Deep Neural Networks is complicated by the fact that the distribution of each layer's inputs changes during training, as the parameters of the previous layers change. This slows down the training by requiring lower learning rates and careful parameter initialization, and makes it notoriously hard to train models with saturating nonlinearities. We refer to this phenomenon as internal co-variate shift, and address the problem by normalizing layer inputs.

Initially we use less number of kernels because initial layers will extract simple features, so by increasing depth of network we increase the number of kernels. So the layers which are at the end of the network will extract more complex features. And at last we use three fully connected layers.

We use activation function as LeakyReLU[8] because ReLU is active during back-propagation only when the units are positive and zero otherwise. This leads to two problems:

1. Dead Neurons
2. Bias Shift

3.3.1.1 Dead Neurons

If the units are not activated initially, then they are always in the off-state as zero gradients flow through them (Dead Neurons). This can be solved by enforcing a small negative gradient flow through the network (Leaky ReLU).

3.3.1.2 Bias Shift

From ReLU, there is a positive bias in the network for subsequent layers, as the mean activation is larger than zero. Though they are less computationally expensive compared to sigmoid and tanh because of simpler computations, the positive mean shift in the next layers slows down learning. This is corrected by either using batch normalization or using activations functions like ELU, SeLU or parametric exponential unit to shift mean towards zero and reduce bias in the activations.

3.3.2 Architecture

We use the following architecture:

ConvNet Architecture		
Layer (type)	Output Shape	Number of Parameter
InputLayer	(None, 512, 512, 3)	0
Gaussian Noise	(None, 512, 512, 3)	0
Conv2D	(None, 253, 253, 32)	4736
Batch Normalization	(None, 253, 253, 32)	128
LeakyReLU	(None, 253, 253, 32)	0
MaxPooling2D	(None, 126, 126, 32)	0
Conv2D	(None, 126, 126, 32)	9248
Batch Normalization	(None, 126, 126, 32)	128
LeakyReLU	(None, 126, 126, 32)	0
Conv2D	(None, 126, 126, 32)	9248
LeakyReLU	(None, 126, 126, 32)	0
MaxPooling2D	(None, 62, 62, 32)	0
Conv2D	(None, 62, 62, 64)	18496
BatchNormalization	(None, 62, 62, 64)	256
LeakyReLU	(None, 62, 62, 64)	0
Conv2D	(None, 62, 62, 64)	36928
BatchNormalization	(None, 62, 62, 64)	256
LeakyReLU	(None, 62, 62, 64)	0

MaxPooling2D	(None, 30, 30, 64)	0
Conv2D	(None, 30, 30, 128)	73856
BatchNormalization	(None, 30, 30, 128)	512
LeakyReLU	(None, 30, 30, 128)	0
Conv2D	(None, 30, 30, 128)	147584
BatchNormalization	(None, 30, 30, 128)	512
LeakyReLU	(None, 30, 30, 128)	0
Conv2D	(None, 30, 30, 128)	147584
BatchNormalization	(None, 30, 30, 128)	512
LeakyReLU	(None, 30, 30, 128)	0
Conv2D	(None, 30, 30, 128)	147584
BatchNormalization	(None, 30, 30, 128)	512
LeakyReLU	(None, 30, 30, 128)	0
MaxPooling2D	(None, 14, 14, 128)	0
Conv2D	(None, 14, 14, 256)	295168
BatchNormalization	(None, 14, 14, 256)	1024
LeakyReLU	(None, 14, 14, 256)	0
Conv2D	(None, 14, 14, 256)	590080
BatchNormalization	(None, 14, 14, 256)	1024
LeakyReLU	(None, 14, 14, 256)	0
Conv2D	(None, 14, 14, 256)	590080
BatchNormalization	(None, 14, 14, 256)	1024
LeakyReLU	(None, 14, 14, 256)	0
Conv2D	(None, 14, 14, 256)	590080
BatchNormalization	(None, 14, 14, 256)	1024
LeakyReLU	(None, 14, 14, 256)	0
MaxPooling2D	(None, 6, 6, 256)	0
Flatten	(None, 9216)	0
Dropout	(None, 9216)	0
Dense	(None, 1024)	9438208
BatchNormalization	(None, 1024)	4096
LeakyReLU	(None, 1024)	0
Dense	(None, 512)	524800
BatchNormalization	(None, 512)	2048
LeakyReLU	(None, 512)	0
Dense	(None, 10)	5130
BatchNormalization	(None, 10)	40
LeakyReLU	(None, 10)	0

Dense	(None, 5)	55

Table 3.1: ConvNet Architecture.

3.3.3 Convolutional Layer

CONV layer will compute the output of neurons that are connected to local regions in the input, each computing a dot product between their weights and a small region they are connected to in the input volume.

3.3.4 Activation Layer

LeakyReLU layer will apply an elementwise activation function. This leaves the size of the volume unchanged.LeakyReLU allow a small, non-zero gradient when the unit is not active.

3.3.5 Batch Normalization Layer

Normalize the activations of the previous layer at each batch, i.e. applies a transformation that maintains the mean activation close to 0 and the activation standard deviation close to 1.

3.3.6 Max Pooling Layer

POOL layer will perform a down-sampling operation along the spatial dimensions (width, height).

3.3.7 Fully Connected Layer

FC layer will compute the class scores, resulting in volume of size [1x1x5], where each of the 5 numbers correspond to a class score. Each neuron in this layer will be connected to all the numbers in the previous layer.

3.3.8 Over-fitting

One of the main problem for ConvNets is Over-fitting. when the network performs better on training data than the validation / test data then model said to be Over-fit. So to control the Over-fitting we used following techniques:

1. Gaussian Noise
2. Batch Normalization
3. Dropout
4. Regularization

3.4 Transfer Learning using VGG16

It usually refers to a deep convolutional network for object recognition developed and trained by Oxford's renowned Visual Geometry Group (VGG)[9], which achieved very good performance on the ImageNet dataset. This model of the 16-layer network used by the VGG team in the ILSVRC-2014 competition.It was the runner up of the ImageNet classification challenge with 7.3 percent error rate.

3.4.1 Discussion

We use the VGG16[9] model upto block5. Then we insert six layers for our data-set. For this model we use image size of (256*256). After getting the output of block5 of VGG16 model we insert a Flatten layer then we insert a Dropout[7] layer to control the over-fitting and and to reduce the number of parameters so that model can be more robust to test dataset. Then we insert five more blocks. Each block consists Dense layer, followed by Batch Normalization layer, which is followed by LeakyReLU layer. We insert a batch normalization so that the mean activation close to 0 and the activation standard deviation close to 1 because of this the training of the model will be fast. We use LeakyReLU activation function because of dying ReLU problem in neural network.

3.4.2 Architecture

The original VGG16 network architecture contains 5 groups of convolutional layers that in total include 13 convolutional layers, each with a kernel size of (3,3), 5 max-pooling layers, each with a pooling size of (2,2). The network accepts 3-channel image of resolution 224*224.

VGG16 Architecture		
Layer (type)	Output Shape	Number of Parameter

InputLayer	(None, None, None, 3)	0
Conv2D(block1)	(None, None, None, 64)	1792
Conv2D(block1)	(None, None, None, 64)	36928
MaxPooling2D(block1)	(None, None, None, 64)	0
Conv2D(block2)	(None, None, None, 128)	73856
Conv2D(block2)	(None, None, None, 128)	147584
MaxPooling2D(block2)	(None, None, None, 128)	0
Conv2D(block3)	(None, None, None, 256)	295168
Conv2D(block3)	(None, None, None, 256)	590080
Conv2D(block3)	(None, None, None, 256)	590080
MaxPooling2D(block3)	(None, None, None, 256)	0
Conv2D(block4)	(None, None, None, 512)	1180160
Conv2D(block4)	(None, None, None, 512)	2359808
Conv2D(block4)	(None, None, None, 512)	2359808
MaxPooling2D(block4)	(None, None, None, 512)	0
Conv2D(block5)	(None, None, None, 512)	2359808
Conv2D(block5)	(None, None, None, 512)	2359808
Conv2D(block5)	(None, None, None, 512)	2359808
MaxPooling2D(block5)	(None, None, None, 512)	0

Table 3.2: VGG16 Architecture upto Block5.

We change the original input size from (224*224) to input size of (256*256). Then after

loading pre-trained weights, fully connected layers are removed from the network up to last densely connected layer of size 4096 hidden units. And we used five fully connected layer in place of last two fully connected layers. To control Over-fitting we use Batch Normalization, Dropout and regularization. The table 3.2 describes the architecture of ConvNet model.

VGG16 Architecture		
Layer (type)	Output Shape	Number of Parameter
InputLayer	(None, 256, 256, 3)	0
VGG16 (model)	multiple	14714688
Flatten	(None, 32768)	0
Dropout	(None, 32768)	0
Dense	(None, 4096)	134221824
BatchNormalization	(None, 4096)	16384
LeakyReLU	(None, 4096)	0
Dropout	(None, 4096)	0
Dense	(None, 2048)	8390656
BatchNormalization	(None, 2048)	8192
LeakyReLU	(None, 2048)	0
Dense	(None, 1024)	2098176
BatchNormalization	(None, 1024)	4096
LeakyReLU	(None, 1024)	0
Dense	(None, 512)	524800
BatchNormalization	(None, 512)	2048
LeakyReLU	(None, 512)	0
Dense	(None, 10)	5130
BatchNormalization	(None, 10)	40
LeakyReLU	(None, 10)	0
Dense(Predictions)	(None, 5)	55

Table 3.3: Modified VGG16 Model.

3.5 Capsule Network

The Capsule Network[10] which performs well on MNIST dataset. CapsNet also required less number of epochs during training but due to large number of kernels at the first and second layer the number of parameters are very high so it increase the time complexity of the model. So We did not made any changes in the CapsNet just use it for Diabetic Retinopathy Images and describe the functioning of the CapsNet.

3.5.1 Discussion

The first part of CapsNet is a traditional convolutional layer. The goal is to extract basic features from the input images, like edges and curves. For this layer we use 256 filters and kernel size of 9*9 and stride of 1. Then we apply a non-linearity function LeakyReLU.

3.5.1.1 PrimaryCaps Layer

The PrimaryCaps layer start of as a traditional convolution layer, but this time we are using a stack of 256 outputs which we are getting from the previous layer. So this time we are using 9*9*256 kernels, instead of 9*9*3 kernels. In the previous layer we were looking for simple features like edges, curves etc., but in this layer we are looking for slightly more complex features, which are combination of the previous layer features. For this layer we are using a stride of 2. That means previously we were moving one pixel at a time now we are moving two pixel at a time . Using stride of two we can reduce the size of our input more rapidly. We will convolve over the output of previous layer with 256 kernels. So we will end up with a stack of 256 89*89 outputs. In total PrimaryCapsules has [32*89*89] capsule outputs (each output is an 8D vector) and each capsule in the [89*89] grid is sharing their weights with each other. These capsules are our new pixels, with a capsule we can store 8 values per location. Now we have 32 capsule layers and each capsule layer has 7921 capsules. That means we have total of 2,53,472 capsules.

Like a traditional 2D or 3D vector, this vector has an angle and a length. The length describes the probability, and the angle describes the instantiation parameters.

3.5.1.2 Squashing

After we have our capsules, we are going to perform another non-linearity function on it, but this time the equation is a bit more involved. The function scales the values of the vector so that only the length of the vector changes, not the angle. This way we can make the vector between 0 and 1, so it's an actual probability.

3.5.1.3 Routing by Agreement

With the help of Routing by Agreement Algorithm we decide what information need to send to the next level. In ConvNet, we usually do "max pooling" after convolutional

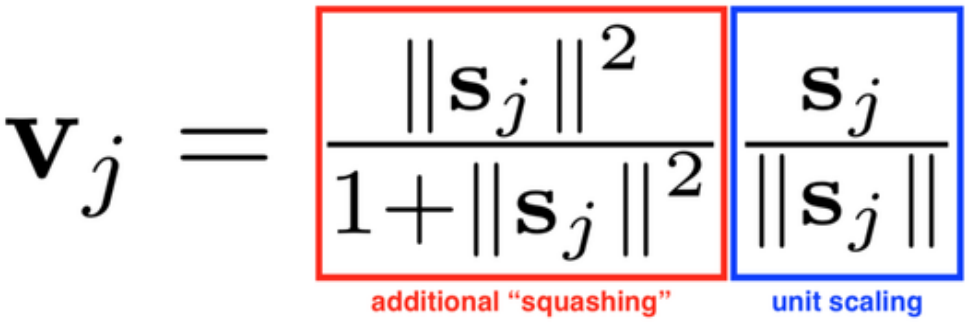


FIGURE 3.5: Squashing function

layer. Max Pooling is used to reduce the size of the image by only passing the highest activated pixel in particular region to the next level.

Now in the CapsNet with the help of Routing by Agreement Algorithm, we only pass the useful information and throw the data that would just add noise to the results. Now using this technique we are reducing the size of the image and also keeping the important information in the image.

The capsule's predictions for each class are made by multiplying its vector by a matrix[16*8] of weights for each class that we are trying to predict. So our prediction is a 16 degree vector.

3.5.1.4 DigitCaps Layer

After Dynamic Routing Agreement we are getting five dimensional vectors i.e one vector for each class. Now this matrix represents the final prediction of the CapsNet model. The length of the vector is the confidence of the correct class prediction. Longer length of the vector represent better prediction. This vector can also be used to regenerate the input image.

3.5.1.5 Reconstruction

This part consists a few fully connected layers. With the help of this reconstruction part we try to generate the original image and then try to minimize the loss between this generated image and the original image. In this way it act like regularizer which help to reduce the over-fitting in the model.

3.5.2 Architecture

CapsNet Architecture		
Layer (type)	Output Shape	Number of Parameter
InputLayer	(None, 192, 192, 3)	0
Conv2D	(None, 185, 185, 256)	49408
LeakyReLU	(None, 185, 185, 256)	0
primarycap_Conv2D	(None, 89, 89, 256)	5308672
primarycap_Reshape	(None, 253472, 8)	0
primarycap_squash	(None, 253472, 8)	0
digitcaps	(None, 5, 16)	163489440
InputLayer	(None, 5)	0
Mask	(None, 16)	0
Dense	(None, 512)	8704
LeakyReLU	(None, 512)	0
Dense	(None, 1024)	525312
LeakyReLU	(None, 1024)	0
Dense	(None, 110592)	113356800
output	(None, 5)	0
out_recon	(None, 192, 192, 3)	0

Table 3.4: CapsNet Model.

Chapter 4

Results and Future Work

In this chapter, we will be showing classification accuracy of some of the benchmark algorithms. We will also compare our results with the deep learning method proposed by Automatic detection and classification of diabetic retinopathy stages using CNN[1] which present the result in terms of accuracy, precision, recall and f1-score. The model has been implemented in python with tensorflow as the backend. We proposed three methods out of which for one model we use the following system configuration:

GPU CONFIGURATION				
Memory	Processor	Graphics	OS type	Disk
125.8 GiB	Intel Xeon(R) CPU E5-2620 V3 @ 2.40 * 24	Quadro k6000/PCIe/SSE2	64-bit	7.6 TB

Table 4.1: GPU Configuration.

For other two models we use Intel AI DevCloud which gives access to a cluster comprised of Intel Xeon Gold 6128 processors.

4.1 Evaluation Criterion

We use accuracy, precision, recall and f1-score to measure the proposed models for Diabetic Retinopathy dataset. We use 5000 images in the test dataset.

4.2 Results

We compare our results with the deep learning method proposed by Automatic detection and classification of diabetic retinopathy stages using CNN[1]. We are getting better results for class1, class2 and class4 using transfer learning from Automatic detection and classification of diabetic retinopathy stages using CNN[1]. For class3 we are getting better precision than Automatic detection and classification of diabetic retinopathy stages using CNN[1] i.e our model gives correct classification for class3 58 percent of the time while their[1] proposed model gives correct classification 36 percent of the time for class3. while we are getting less recall than their[1] recall i.e our model out of total class3 samples in test set predict 34 percent of the time as class3 while their[1] model predict 56 percent of the time as class3 out of total sample for class3.

Modified VGG16 Result			
Class Label	Precision	Recall	f1-score
class0	0.82	0.93	0.87
class1	0.61	0.49	0.55
class2	0.70	0.75	0.72
class3	0.58	0.34	0.43
class4	0.80	0.61	0.69
Average	0.72	0.73	0.72

Table 4.2: Modified VGG16 Result(proposed method2)

Class Label	Precision	Recall	f1-score
class0	0.88	0.95	0.91
class1	0.40	0.39	0.39
class2	0.70	0.42	0.52
class3	0.36	0.56	0.43
class4	0.62	0.49	0.54

Table 4.3: Automatic detection and classification of diabetic retinopathy stages using CNN Result

Class Label	Precision	Recall	f1-score
class0	0.78	0.86	0.82
class1	0.50	0.31	0.39
class2	0.61	0.73	0.66
class3	0.50	0.17	0.25
class4	0.48	0.50	0.49

TABLE 4.4: CNN Result(proposed method1)

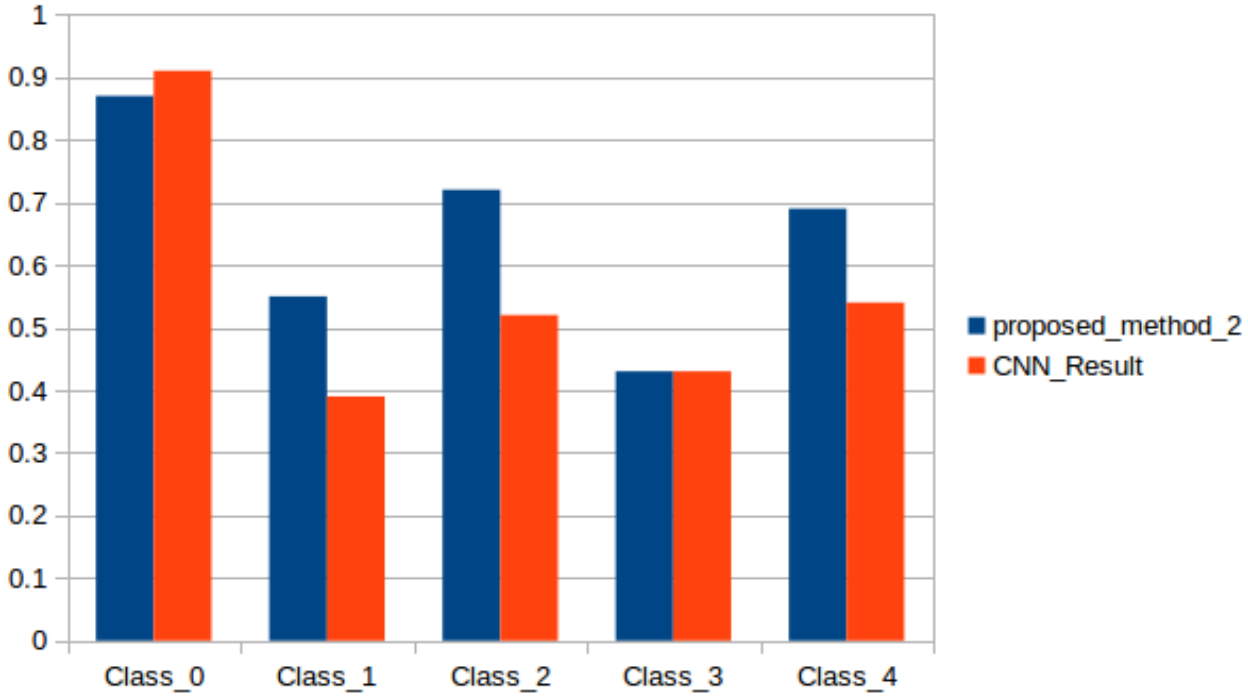


FIGURE 4.1: f1-score Comparison1

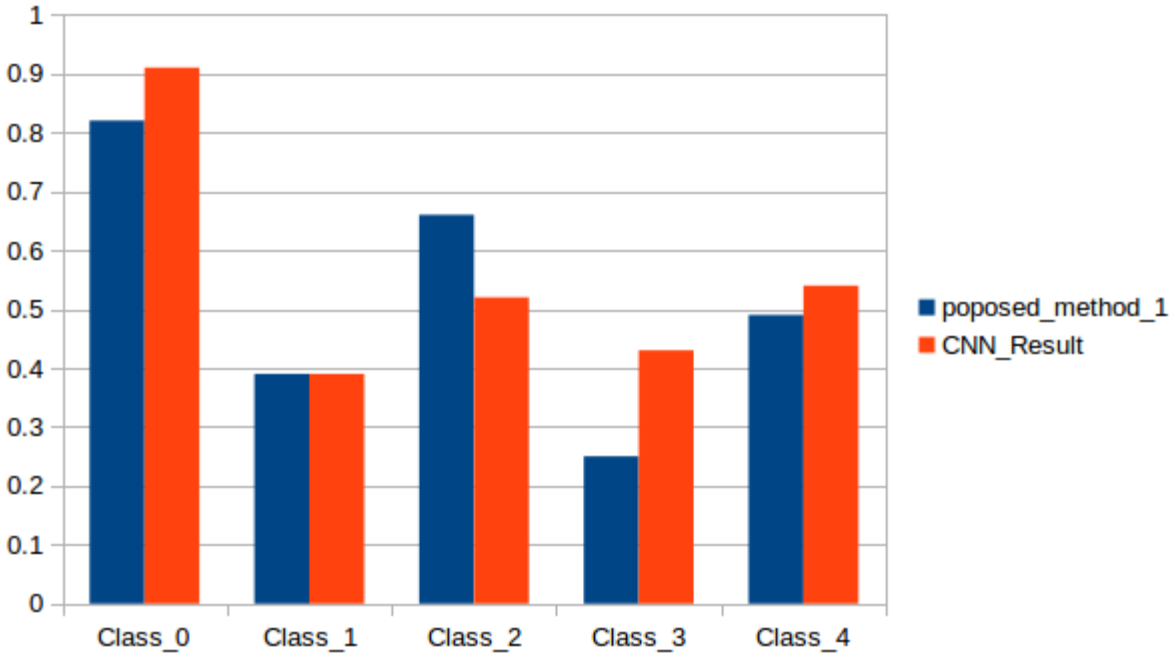


FIGURE 4.2: f1-score Comparison2

4.3 Conclusion

Using transfer learning we are getting better results, except class0, f1-score for all the classes is greater than or equal to their[1] f1-score. We are getting these results using image size of (256 * 256). But using CapsNet[10] we get accuracy of 64.20 percent on test dataset. We use image size of (192*192). Due to large number of parameters we are not able to run the code for image size more than (192*192), so the CapsNet[10] may give better results for image size more than (192*192). For method1 i.e using CNN we are not getting good results. Using method1 we get 64.93 percent accuracy. For method1 we use image size of (512 * 512).

4.4 Future Work

For CapsNet[10] we use image size of (192 * 192). We increase the image size i.e more than (192 * 192) than we get resource exhausted error. So due to limited resources we did not check the CapsNet for image size more than (192 * 192). So by increasing the image size we can get better results.

Bibliography

[1] R. Ghosh, K. Ghosh, and S. Maitra. Automatic detection and classification of diabetic retinopathy stages using cnn. In *2017 4th International Conference on Signal Processing and Integrated Networks (SPIN)*, pages 550–554, Feb 2017. doi: 10.1109/SPIN.2017.8050011.

[2] Rajendra Acharya U, Chua Kuang Chua, E. Y. Ng, Wenwei Yu, and Caroline Chee. Application of higher order spectra for the identification of diabetes retinopathy stages. *J. Med. Syst.*, 32(6):481–488, December 2008. ISSN 0148-5598. doi: 10.1007/s10916-008-9154-8. URL http://dx.doi.org/10.1007/s10916-008-9154-8.

[3] Hadley Wickham. Reshaping data with the reshape package. *Journal of Statistical Software*, 21(12), 2007. URL http://www.jstatsoft.org/v21/i12/paper.

[4] Karel Zuiderveld. Graphics gems iv. chapter Contrast Limited Adaptive Histogram Equalization, pages 474–485. Academic Press Professional, Inc., San Diego, CA, USA, 1994. ISBN 0-12-336155-9. URL http://dl.acm.org/citation.cfm?id=180895.180940.

[5] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, Nov 1998. ISSN 0018-9219. doi: 10.1109/5.726791.

[6] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, pages 448–456. JMLR.org, 2015. URL http://dl.acm.org/citation.cfm?id=3045118.3045167.

[7] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014. URL http://jmlr.org/papers/v15/srivastava14a.html.

[8] Andrew L. Maas, Awni Y. Hannun, and Andrew Y. Ng. Rectifier nonlinearities improve neural network acoustic models. In *in ICML Workshop on Deep Learning for Audio, Speech and Language Processing*, 2013.

[9] K. Simonyan and A. Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. *ArXiv e-prints*, September 2014.

[10] S. Sabour, N. Frosst, and G. E Hinton. Dynamic Routing Between Capsules. *ArXiv e-prints*, October 2017.

[11] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, pages 807–814, USA, 2010. Omnipress. ISBN 978-1-60558-907-7. URL http://dl.acm.org/citation.cfm?id=3104322.3104425.

[12] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger. Densely Connected Convolutional Networks. *ArXiv e-prints*, August 2016.

[13] Shie Mannor, Dori Peleg, and Reuven Rubinstein. The cross entropy method for classification. In *Proceedings of the 22Nd International Conference on Machine Learning*, ICML '05, pages 561–568, New York, NY, USA, 2005. ACM. ISBN 1-59593-180-5. doi: 10.1145/1102351.1102422. URL http://doi.acm.org/10.1145/1102351.1102422.

[14] D. P. Kingma and J. Ba. Adam: A Method for Stochastic Optimization. *ArXiv e-prints*, December 2014.

[15] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[16] François Chollet et al. Keras. https://keras.io, 2015.

[17] Guido Rossum. Python reference manual. Technical report, Amsterdam, The Netherlands, The Netherlands, 1995.

[10] [5] [7] [11] [12] [13] [14] [8] [1] [9] [6] [4] [3] [15] [16] [2] [17]