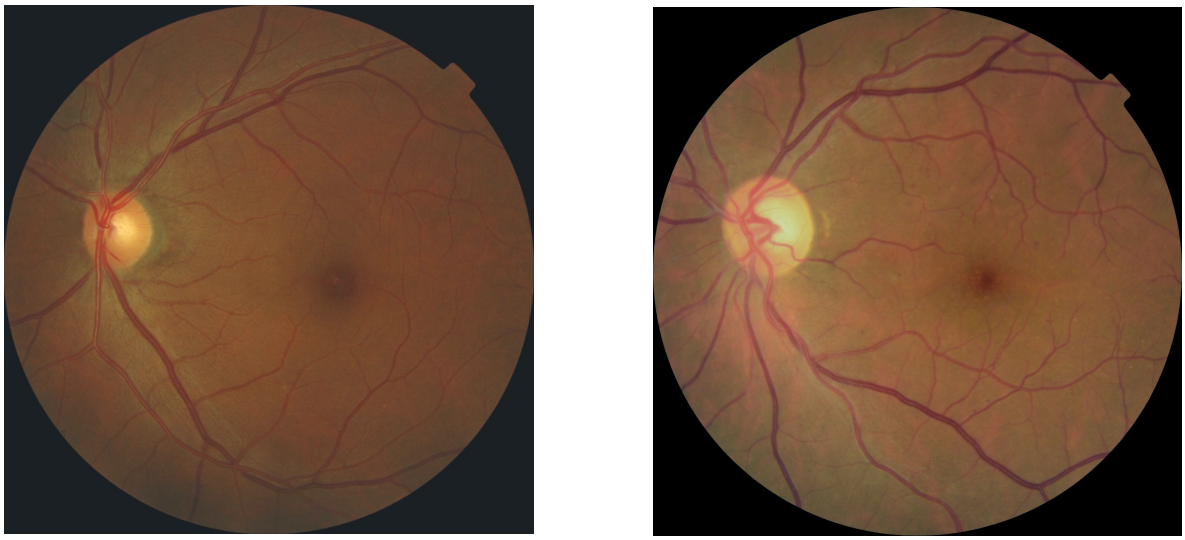
Convolutional Neural Networks to Detect Severity of Diabetic Retinopathy

Chirag Maheshwari
Jaspinder Singh

Introduction

Diabetic retinopathy, an ocular manifestation of diabetes, is the leading cause of blindness in the working-age population of the developed world and affects over 93 million people worldwide. It affects up to 80 percent of patients who have had diabetes for more than 20 years. Left untreated, it can eventually lead to blindness, but it is difficult to detect as it shows few symptoms until it is too late for effective treatment.

Current methods of detecting diabetic retinopathy are time-consuming and require a trained clinician that can perform a variety of tests, one which is fundus photography. Fundus photography involves taking a photograph of the back of the eye so structures such as the central and peripheral retina, optic disc, and macula are captured. Signs to look out for are leaking blood vessels, retinal swelling, fatty deposits on the retina, damaged nerve tissue, or any other changes to the blood vessels. However, as we can see from the following example, it is difficult to detect the difference between a healthy eye and one with diabetic retinopathy.



The image on the left shows the fundus photograph of a completely healthy eye, while the image on the right shows severe signs of diabetic retinopathy.

Due to the resource intensive process and expertise needed to accurately identify diabetic retinopathy, not all populations throughout the world can get the disease diagnosed in time. The World Health Organization predicts that the number of cases of people with diabetes will increase from about 8% of the global population to 10% by the year 2030 [1]. 80% of those people live in low- and middle-income countries which don't have the proper facilities or expertise to correctly diagnose a large number of people. Therefore, it is important to try to get systems that can either assist or autonomously detect diabetic retinopathy from images to help prevent blindness in millions of people who don't have access to adequate healthcare.

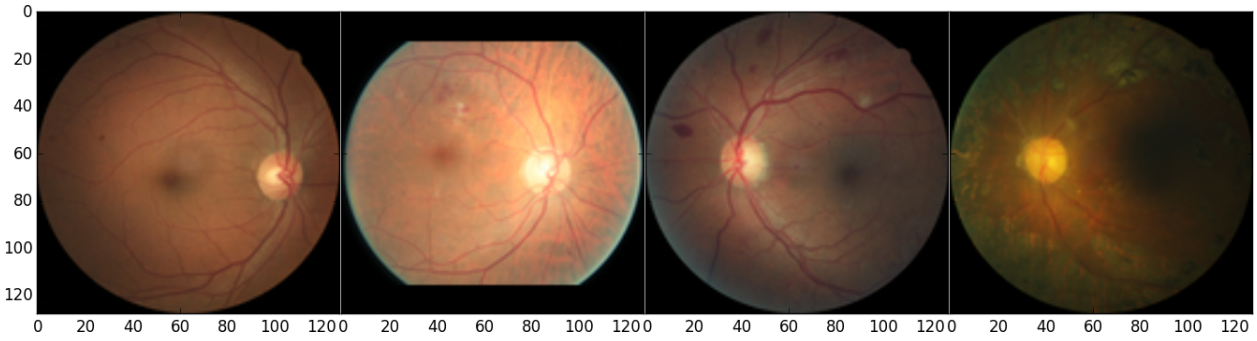
Previous work done to detect diabetic retinopathy used algorithms such as feature extraction [2]. Such methods were used to detect the size of the optic disc in the image by measuring the contours using a Hough transform, try to detect fat deposits, extracting the area of the blood vessels, and trying to detect the texture features of the image. These studies only

used about 400 images which were homogenous and achieved a sensitivity of 82% and a specificity of about 86%. Considering that fundus photographs won't always have the same brightness, contrast, or other features, this was not an acceptable classifier to detect diabetic retinopathy. Nor did this classifier detect the differences between various severities of diabetic retinopathy since it used feature detection to only check for size of optic disc or blood vessels.

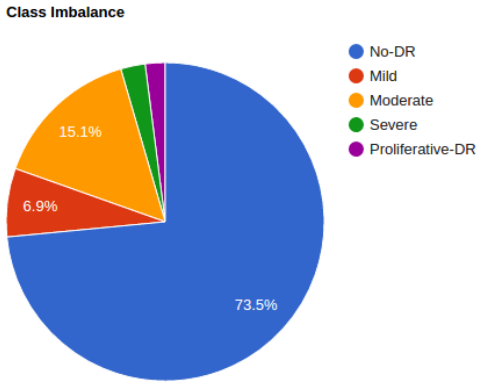
Overview

Dataset

We decided to use a convolutional neural network to detect diabetic retinopathy in a set of images. A total of 35,126 high resolution images were provided by EyePACS and they were separated into five different classes. The five states of severity are: 0 – No diabetic retinopathy, 1 – mild diabetic retinopathy, which usually has cases of microaneurysms, 2 – moderate diabetic retinopathy, which has microaneurysms and fuzzy light splotches, 3 – Severe diabetic retinopathy, which has shunt vessels, venous bleeding, intra-retinal hemorrhages but no new blood vessels growing, and 4 – Proliferative diabetic retinopathy, which has neovascularization, new blood vessels growing, and vitreous/pre-retinal hemorrhage [3]. The data set has highly imbalanced class labels which creates difficulty when building a classifier and needs to be addressed.



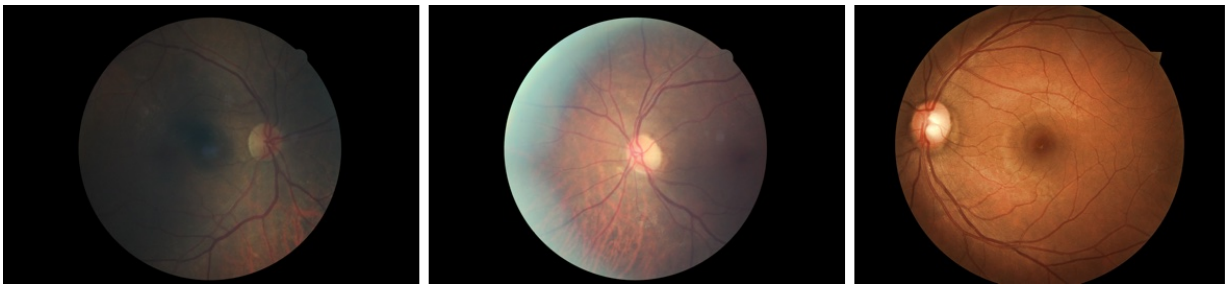
Examples of the various severities of diabetic retinopathy of classes 1,2,3,4 – left to right



Class imbalance of the data set
0 - No DR – 25,180 images
1 – Mild NDPR – 2,443 images
2 – Moderate NDPR – 5,292 images
3 – Severe NPRDR – 873 images
4 – PDR – 708 images

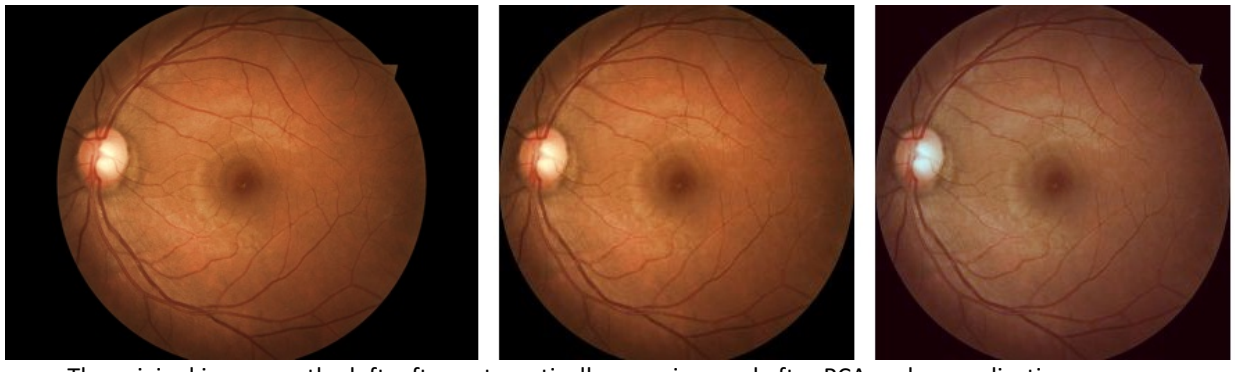
Pre-processing

The majority of the images had lots of noise, variations in the brightness, under or over exposed, came in various sizes and aspect ratios (about 80% of the images had a 3:2 aspect ratio, the rest a 4:3), and often had excess background black space. The images are often large too, usually contain millions of pixels and are larger than one megabyte in size.



Most images have large areas of black background that needed to be cropped out. The leftmost image is an example of an image darker than many of the others. Middle image shows an example of over-exposure. Right image is an ideal fundus photograph that has proper exposure and sharp contrast.

We wanted to remove as much of the black background as possible while avoiding cropping any part of the eye in the image. Since the images were of different sizes, we couldn't not just batch crop them so had to devise a different technique. The technique we used was to first convert the image to grayscale and keep any part of the image above a certain threshold, in our case we used 10 since the black background would have a value of 0. After cropping the image, we then performed principal component analysis using the entire data set and getting the eigenvalue for each RGB space. This allows us the catch a property of natural images that object identity is invariant to changes in the intensity and color of the illumination [4]. After that we than performed the normalization on all the images using mean and variance. Finally, we resized the images so training them on a neural network would take a reasonable amount of time. We created two different sets, one of size 128x128 and another of size 256x256. This also allows us to test how much of a difference in accuracy we get if we use images of different sizes.



The original image on the left, after automatically cropping, and after PCA and normalization.

Pipeline

After pre-processing the images and creating new sets, we jittered each batch while running the training program to help prevent overfitting [5]. The different methods applied were color cast where there is a probability of adding/subtracting a constant to each channel, saturation jitter, brightness jitter, rotation, horizontal flip, vertical flip, and translation. Each method had a probability of 0.5 of being applied. Convolutional neural networks have a built-in invariance to small translations and rotations so if the dataset does not originally contain them, adding them in gives a more robust training set and can avoid overfitting.

We also tried random resampling due to the imbalance of images in different classes but we noticed that it vastly increased the running time of the program and were not able finish running the training program in time.

Network Architecture

We tried a variety of different architectures for our convolutional neural network with some that take in image inputs of size 128x128 pixels and others size 256x256 pixels. Our first model was a deep architecture that took in images of 256x256 pixels and modeled after the architecture of DeepSense.io [6] but we also added batch normalization after every convolution. Batch normalization helps with not having to worry about properly initializing the hyper-parameters of a neural network and eliminates the need for dropout [7]. Input was 3x256x256 with either 3x3 or 2x2 strides, the architecture had 9 convolution layers, 5 pooling layers, used Rectified Linear Unit (ReLU) as the non-linearity and dropout at the end with probability of 0.5 which should help prevent overfitting by introducing stochastic behavior in the forward pass of the neural network [8]. The ReLU layer is an activation function of

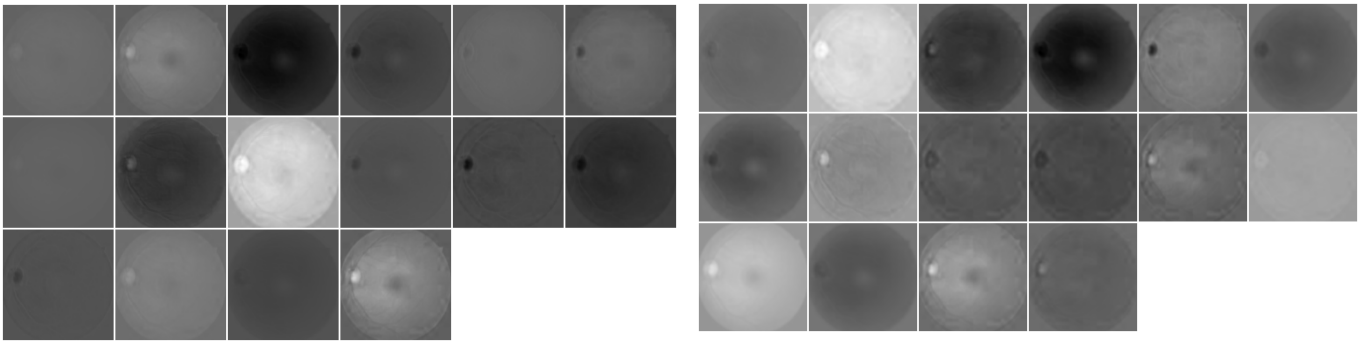
$$f(x) = \max(0, x)$$

where x is the input to the neuron. A smooth approximation is the analytic function

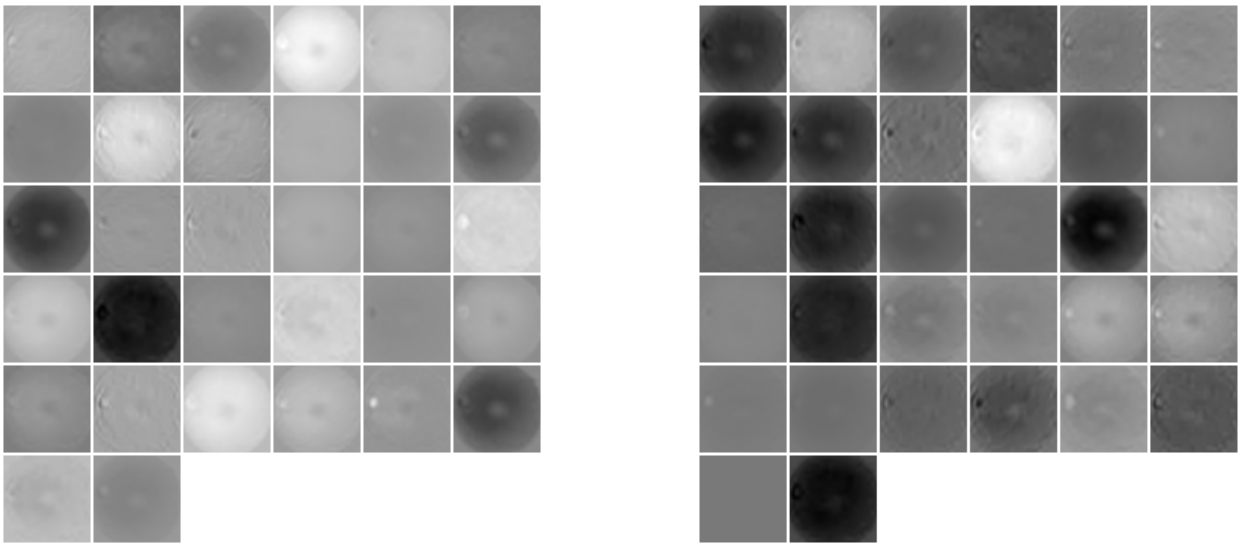
$$f(x) = \ln(1 + e^x)$$

Other architectures we tried out used 3x128x128 sized images to help speed up the training time. We tried out more simple architectures with 4, 5, 6, 7, and 8 layers of convolution. Some of these models did not have batch normalization or dropout added into the model architecture. We tried out these simple models to speed up the time it takes to train the models and to see how much accuracy improves as we add a convolution layer in our model. The simple architectures used 3x3 strides for all convolution layers and used either TanH or leaky-ReLU with a parameter of 0.1 for non-linearity. Leaky ReLU were used to help prevent the "dying ReLU" problem so instead of the function being 0, it will have a negative slope [9].

When running the training program, we used the same hyper-parameters. We used a consistent batch size of 128 between all models, ran it for 50 epochs, a momentum of 0.9, learning rate of 0.1, and a weight decay of 1e-4 and a learning rate decay of 1e-4. Using a consistent set of hyper-parameters allowed us to better compare the accuracy of the different models.



First layer (left) and second layer (right) of weights from the 8-layer architecture



Third (left) and fourth (right) layers of weights from the 8-layer architecture.

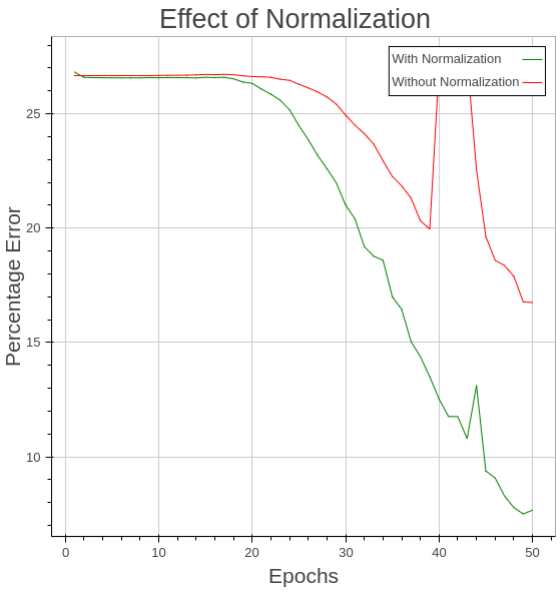
Analysis

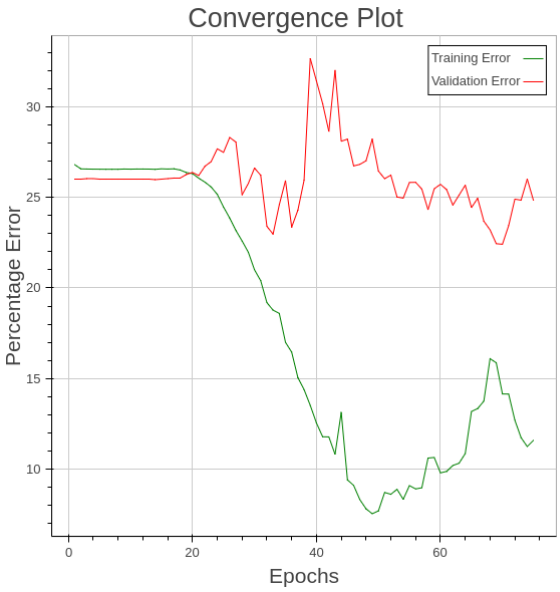
Looking at weights from the 8-layer architecture, we notice that once it starts going into the third and fourth layer, some of the different feature maps are starting to look similar to one another which may make it difficult to differentiate one from another. For example, the third layer has many images where it looks uniform gray. This is as opposed to the first two layers which have feature maps that are different form one another. One possible cause for this is not letting the training model run for enough epochs. Experimenting with different hyper-parameters such as changing the learning rate may helped converge faster and given better accuracy. This architecture is similar to the other simple layered architecture and those were also run with the same parameters which is why the images from them looked similar to the 8-layered convolutional neural network.

Looking at the effect of training with and without normalization, we can see that adding in batch normalization makes about a 5% difference in the training and reduces the amount of randomness (such as the size of spikes) and noise. After about 50 epochs, the percentage error with normalization in our models is about 8% while without normalization it is at 16% with spikes. Since the data is not normalized, that means the ranges of the feature values are

different for each feature and therefore the learning rate causes corrections in each dimension that differs from one another. Therefore, it might be overcompensating a correction in one weight dimension while undercompensating in another.

Looking at the convergence plot for the 8-layer architecture, we see that the training error decreases over time with an increase after epoch 60. However, the validation error seems to hover around 25%. This may be possible because the learning rate and/or momentum is too high. Therefore, it may be stuck in some local minima since there percent error is no longer decreasing.





Looking at the training and test errors after 75 epochs for all the models created, we notice that the validation error decreases as more layers are used in a neural network architecture. Compared to the benchmark, the validation error does not perform better until the neural network has at least 6 layers, and Deepsense, which had 9 layers performs the best. One major issue that may affect the error rates is the class imbalance in the data set. As noted before, the majority of the images are considered to be healthy and only a small portion have a case of severe diabetic retinopathy. An imbalanced training set can have a severely negative impact on the performance of a cnn, and a balanced training set would yield much better results [10].

Network	**Training Error**	**Validation Error**
Benchmark	-	26.52
4 Layer	6.6428	36.4465
5 Layer	7.9271	37.2722
6 Layer	27.1787	25.1993
7 Layer	5.6274	25.6177
8 Layer	11.5648	24.8442
DeepSense	5.6375	20.3924

Conclusion

We have shown that it is possible to classify the severity of diabetic retinopathy from fundus photographs of the eye. Improvements to the accuracy can be made by building the proper architecture for the neural network and properly processing the images. Most importantly, especially for classifying medical data, is to make sure to use random resampling in during the training process. Medical data will have a healthy classification for the majority of the classes since a specific disease may not be as common. A properly trained CNN can be of benefit since it can train thousands of images in real-time and can be of assistance to areas where facilities to properly diagnose diabetic retinopathy is not available.

Bibliography

[1] S. Akter, M. Rahman, S. Abe and P. Sultana, Prevalence of diabetes and prediabetes and their risk factors among Bangladeshi adults: a nationwide survey, World Health Organization, 2014.

[2] T. Walter, J.-C. Klein, P. Massin and A. Erginay, A Contribution of Image Processing to the Diagnosis of Diabetic Retinopathy—Detection of Exudates in Color Fundus Images of the Human Retina, IEEE TRANSACTIONS ON MEDICAL IMAGING, 2002.

[3] B. Klein, M. Davis, P. Segal, J. Long and A. Harris, Diabetic Retinopathy: Assessment of Severity and Progression, Opthalmology, 1984.

[4] A. Krizhevsky, I. Sutskever and G. Hinton, ImageNet Classification with Deep Convolutional Neural Networks, NIPS, 2012.

[5] P. Sermanet and Y. LeCun, Traffic Sign Recognition with Multi-Scale Convolutional Networks, IJCNN, 2011.

[6] R. Bogucki, "Diagnosing diabetic retinopathy with deep learning," DeepSense.io, 3 September 2015. [Online]. Available: https://deepsense.io/diagnosing-diabetic-retinopathy-with-deep-learning/. [Accessed 23 December 2016].

[7] S. Ioffe and C. Szegedy, Batch Normalization: Accelerating Deep Network Training b y Reducing Internal Covariate Shift, Google, 2015.

[8] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever and R. Salakhutdinov, Dropout: A Simple Way to Prevent Neural Networks from Overfitting, Journal of Machine Learning Research, 2014.

[9] D.-A. Clevert, T. Unterthiner and S. Hochreiter, Fast And Accurate Deep Network Learning By Exponential Linear Units, ICLR, 2015.

[10] P. Hensman and D. Masko, The Impact of Imbalanced Training Data for Convolutional Neural Networks, KTH School of Computer Science and Communication, 2015.