Convolution Neural Network Approach to Diagnosing Diabetic Retinopathy

José Solomon *

Abstract

This document discusses the fundamental concepts and results of my project on diabetic retinopathy diagnosis using convolution neural networks (ConvNN). It is derived from the Kaggle Diabetic Retinopathy Challenge [1] aimed at creating a robust algorithm to automate the process of classifying the level of diabetic retinopathy found in retinal scans. The fundamental components of ConvNN are introduced and the details of how these networks were trained are reviewed.

Contents

*jose.e.solomon@gmail.com

1 Introduction

There are two key facets to the project: first is the concept of the diabetic retinopathy and its diagnosis; the second is ConvNNs and their general functioning principles as applied to digital imaging processing and categorization. We begin with a cursory description of the former, and then focus the bulk of the theoretical discussion on the latter.

2 Diabetic Retinopathy

Diabetic retinopathy (DR) is a disease that generally afflicts those who have dealt with diabetes for a period of 5 years or more [2]. It is defined specifically as the deterioration of blood vessels found or that lead to the retina of the human eye, as illustrated in Figure 1 below.

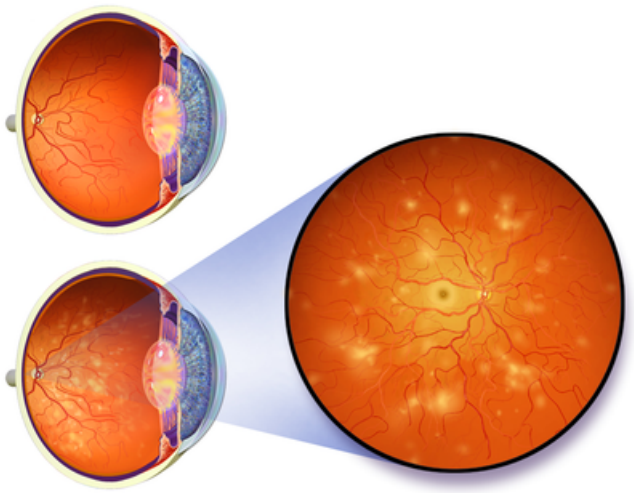
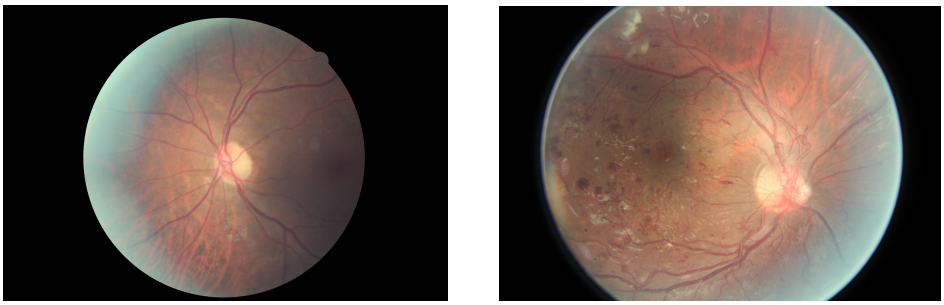


Figure 1: Diabetic retinopathy in comparison to a normal eye [3], (shown at the top left)

The primary cause of DR is a byproduct of the condition of diabetes, where finer blood vessels in the human body tend to form abnormal branching structures and begin to exhibit thinning walls. This leads to deteriorated blood supply to the effected areas and eventually to hemorrhaging. In terms of the retina, this leads to blind spots forming in the person's field of vision.

To diagnose the condition, a retinal scan of the eye is taken and a classification is assigned based on a combination of the following features: the number of abnormal vein branching seen, the general wall thickness of the blood vessels, and the number of blood stains observed due to hemorrhaging. DR is categorized on a scale from 0 to 4, and examples of level 0 and level 4 DR are shown in Figure 7. It should be noted that this method of categorizing DR is not fundamentally rigorous in nature and is more of a qualitative scale.



(a) Level 0 DR (b) Level 4 DR

Figure 2: Examples of level 0 and level 4 DR

2.1 The Kaggle Grand Challenge

Kaggle has presented automated DR diagnosis as an open grand challenge. In doing so, Kaggle provided more than 35 gigabytes of retinal scan data, (over 35,000 images), where each image is roughly 1.5 megabytes in size. To aid in download and processing of the data set, Kaggle divided the image set into a training group and a testing group, and then subsequently divided each group into smaller subsets to facilitate file transfer. All the results presented here and in subsequent discussions are based on the first training set of images.

It is noted that the aim of the current work is to understand ConvNN from a fundamental perspective, and not to actually create a competitive solution for the grand challenge. Due to the size of the reference data set, and the limitations imposed by Python in terms of loading elements into shared memory, it would be very difficult to create a code base that could compete directly with other implementations in CuDa, (a GPU-level system language), which is the de facto favorite language for image processing.

3 Convolution Neural Networks

ConvNN is a powerful machine learning technique that falls under the general premise of deep learning. There are number of flavors of ConvNN, but the specific implementation presented here is one derived from of the first concrete examples of the technique, the LeNet-5 [4], which is especially adept at processing digital imaging.

The LeNet-5 consists of 3 primary component layers: the convolution layer, the pooling layer, and the conventional fully connected layer [5].

3.1 The Convolution Layer

The convolution layer is the work horse of the ConvNN, and it is what makes it such an indispensable tool for image processing. Mathematically, the concept of convolution is somewhat straightforward, and plays a key role in spectral methods/Fourier transform treatments. In terms of digital imaging, convolution can be expressed as [5] the product of the pixel intensity

of a given image with that of a kernel, and is stated as

$$H[m,n] * G[m,n] = \sum_{u=-\infty}^{\infty} \sum_{u=-\infty}^{\infty} H[u,v]G[m-u,n-v] \tag{1}$$

This may seem somewhat complex at first, but consider that a digital image is an array of pixels, each with an intensity that ranges from 0 up to a given bit depth, (e.g. 255 for 8-bit images). We can see that an image can be translated as matrix, where each element is a given pixel's image intensity. The concept is illustrated in Figure 3.

Figure 3: Digital image as a matrix [6]

With this concept in place, a kernel is itself a matrix of a prescribed dimension, smaller than that of the original image, an example of which would be

$$H^{L}_{3\times3} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \tag{2}$$

This matrix is applied, via convolution, across the matrix of the image to create a filtered image which, depending on the *stride*, (i.e. number of times the kernel is applied across the original image), downsizes the original image into smaller matrix which is representative of the dot product between the original image and kernel. The full convolution concept is illustrated in Figure 4.

3.2 The Pooling Layer

In order to reduce the computational expenditure of the ConvNN algorithm, a pooling module is often used to reduce the required number of weights for a subsequent fully connected neural network layer.

Usually the form of the pooling is *max pooling*, where a specific number of elements of the convoluted image are defined as common-pool members, (as illustrated in Figure 5), and only the pool member with the maximum value is carried forward in the network. The size of a pool is usually, but not required to be, the same size as the kernel applied during

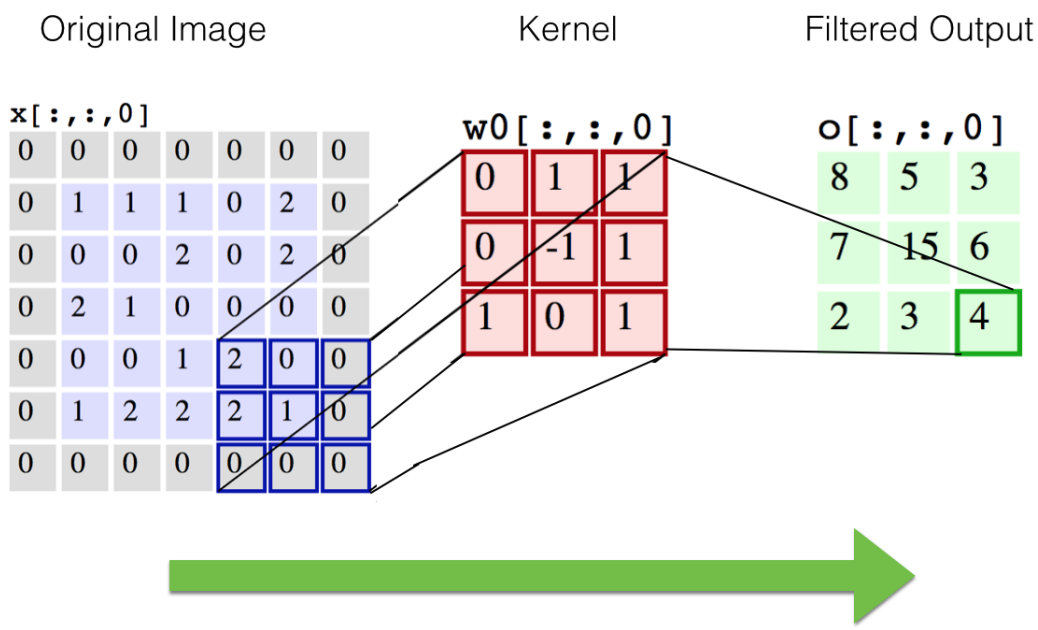


Figure 4: Convolution concept [7]

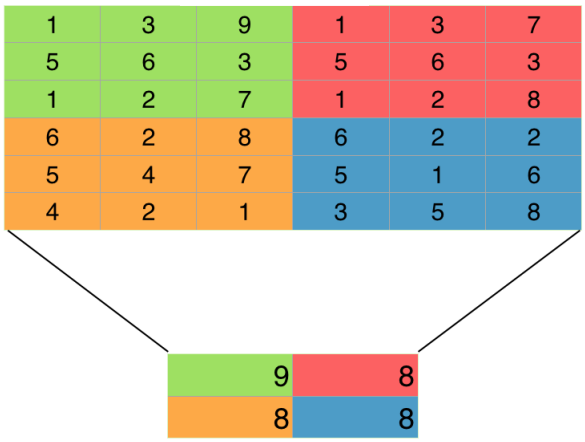


Figure 5: Max pooling is where the largest value of each sub-cell is fed forward

convolution. By focusing on the maximum value of a convolution, the network is becoming more sensitive to those image features which are more dominant in the total feature set.

It is noted that a ConvNN often uses a series of convolution and pooling layers to continuously reduce the computational load of the final layer, or layers, of fully connected NN, which is normally the most computational expensive facet of the ConvNN pipeline.

3.3 The Fully Connected NN Layer

The fully connected NN layer is actually a *conventional* neural network, which is comprised of a series of nodes that are interconnected via a series of weights and bias units. Let's begin with the illustration shown in Figure 6.

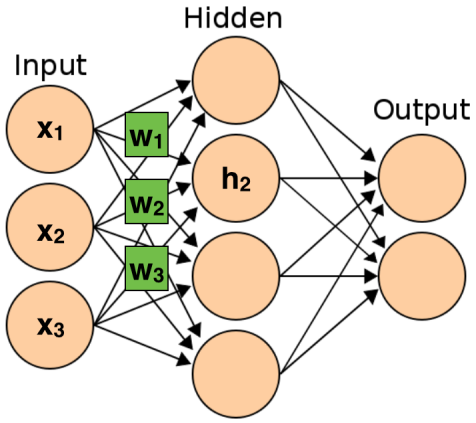


Figure 6: A fully connected neural network [8]

As can be seen from the figure, each node is connected to the series of nodes in the layer directly downstream of it. The nodes downstream are in turn connected to each node in the layer directly upstream, as well as each node in the layer directly downstream. The result is a series of nodes that are *fully* interconnected.

Let's say that for each input connection of a given node, there is an associated weight w_i. Given that there is N number of nodes in the upstream layer, the input to the downstream node can be seen as

$$f(\vec{w}^T\vec{x}) = f(w_0 \cdot x_0 + w_1 \cdot x_1 + \cdots + w_N \cdot x_N) f_{reLu}(a) = \max(0, a) \tag{3}$$

We note that the dot product $\vec{w}^T\vec{x}$ is the input to a function, known as the activation function, which in turn is the input value of the downstream node. There are a variety of activation functions that can and have been used in NN. One of the most popular, and the one which is used here, is the *tanh* activation function, which is defined as

$$f_{\tanh}(a) = \frac{e^a - e^{-a}}{e^a + e^{-a}} \tag{4}$$

which has an activation profile shown in Figure 7(a) As can be seen, the function has asymptotes at 1 and -1 and exhibits positive values in y for positive values of x. Another common

activation function is the sigmoid activation function, which has a similar profile as *tanh* and has the functional form

$$f_{sigmoid}(a) = \frac{1}{1 - e^{-a}} \tag{5}$$

As shown in Figure 7(b), the sigmoid activation function has asymptotes at 1 and 0, and is centered at the origin in x and at 0.5 in y.

Another popular variant, and the one that was ultimately used to provide the final set of results, is the rectifier activation function, (commonly referred to as a *ReLu*), defined as

$$f_{ReLu}(x) = \max(0, x) \tag{6}$$

This is very straightforward activation function, illustrated in Figure 7(c), that tends to have more favorable characteristics from the perspective of backward propagation, which is the *learning* process of the network.

The sigmoid and *tanh* activation function are commonly used, but the *ReLu* activation is gaining more traction in terms of ConvNN due to the favorable back-propagation properties, (i.e. the learning process of the NN). Back-propagation is a key concept in NN and ConvNN, but its description is beyond the scope of the current document.

3.4 Logistic Regression Node

The final layer of the ConvNN is usually a classifier. In the attempted implementation by the author, the logistic regression function was used as the output node. This is a classifier reviewed in class and it's functional form is [5] and sometimes labeled as a *softmax* classifier.

$$y_{prediction} = \text{argmax} P_i(Y = i|\{x, w\}) \tag{7}$$

where the probability is

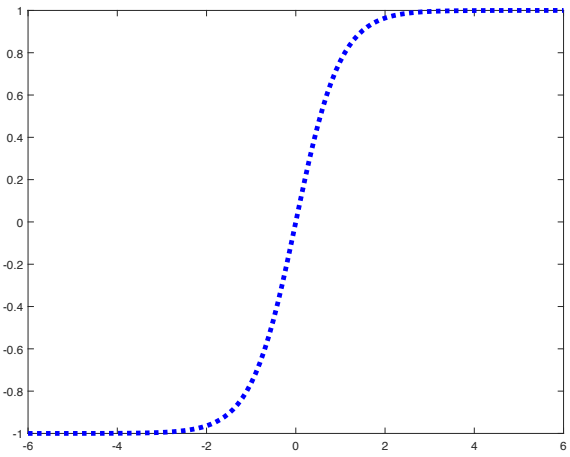
$$P_i = \frac{e^{\vec{w}_i^T \vec{x}_i}}{\sum_j \vec{w}_j^T \vec{x}_j} \tag{8}$$

This simply means that the output of the NN is mapped to one of the label of classes, (in this case the level of DR which ranges from 0 to 4).

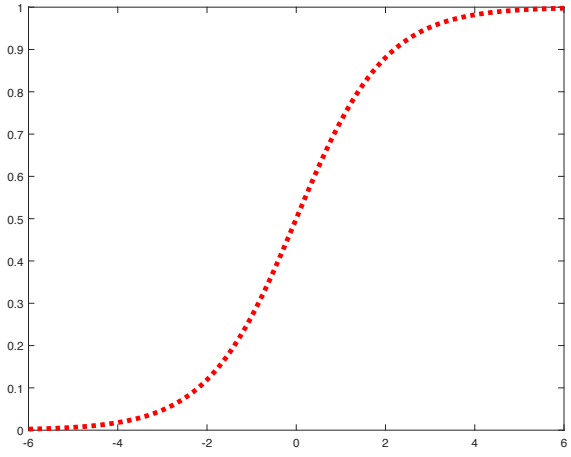
3.5 Summary

So putting all the pieces together, the ConvNN has the general layout seen in Figure 8

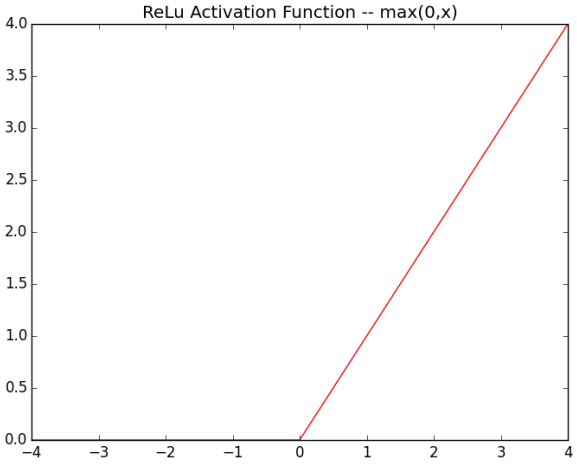
As noted in the figure, the convolution layer and the max pooling layer are often stacked multiple times in practice to reduce the overall computational load of the fully connected neural network module, which is often the most computational expensive member of the ConvNN. In the author's implementation, three convolution/pooling layers where stacked before output was connected to the fully-connected neural network. In the final functional network, which was implemented using NVIDIA's Digits [9], seven convolution layers where used with pooling interspersed throughout the different junctures of the network.



(a) Tanh Function



(b) Sigmoid Function



(c) ReLu Function

Figure 7: NN activation functions

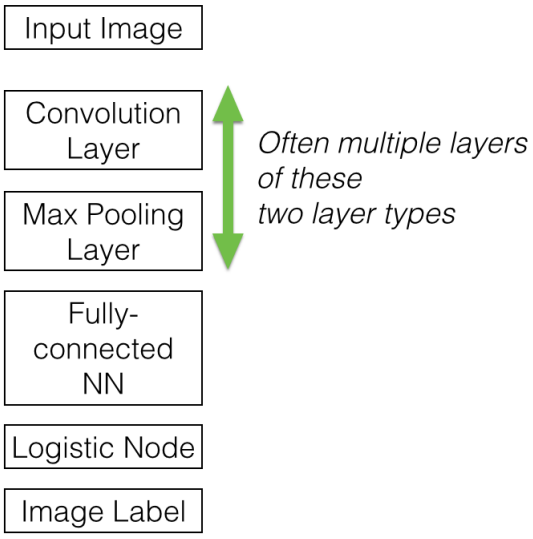


Figure 8: The ConvNN as a whole

4 Actual Implementation

Python is a powerful and flexible language, but it is not entirely ideal to deal with image processing. An initial attempt to code a ConvNN using the Theano library [10] was hampered by excessively long run times that prohibited debugging of the code base or real analysis of the results.

To manage the challenges of handling such large images in conjunction with the issues related to memory management of Python code, a move was made to run code using the Digits framework, which implements ConvNN within the framework of a Graphic Processing Unit, (GPU), architecture. GPUs have significant advantages over Central Processing Units, (CPUs), implementations when it comes to processing digital images for a number of reasons. Mainly, GPUs consist of a high number of cores who are optimized to segregate shared memory instances and run individual processes on subsets of the larger instance. This comes about from the GPU's primary function: to take a large image, break it down into smaller subsections and send it to your display interface. By leveraging GPUs to break up an image matrix into smaller matrices, run convolution/pooling/fully-connected NN on the image sub-samples, and then tying things back together in the final classifier node, a GPU implementation can be order of magnitudes faster than standard, serial CPU implementation.

This section will discuss the architecture of the ConvNN that was used to produce most of the results documented, but first a concise review will be given on image pre-processing which was required before actual ConvNN training could begin.

4.1 Image Preparation

To process images, the *openCV* computer vision library was initially used to load, crop and convert to gray-scale all images in the selected training set. Below is an excerpt from the

code that does some of this processing. Once images are loaded and cropped they look as shown in Figure 9.

```
def processD(self):
        # Load left image
        for i in range(0,len(self.lImageNames)):
            print 'Left images being loaded: ' + str(i)
            name = self.imageDPath+self.lImageNames[i]+'.jpeg'
            image = cv2.imread(name,4)
            # crop the image
            image = image[10:2010,610:3010]
            # convert to gray scale
            imageL = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
            self.lImageList.append(imageL)
```

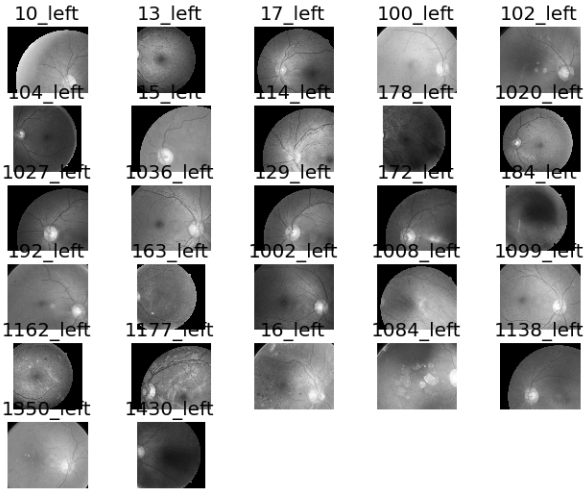


Figure 9: Images loaded, cropped, and converted to gray-scale

Initial testing found that color is actually a very significant aspect of the diagnosing procedure, and so a different method of cropping and reducing the size of image was used using the textitimagemagik toolbox. Resulting images where converted from *jpeg* to *png* file format and went from roughly 4000x2500 pixels to 256x256 pixels, resulting in image file size reductions from 9 Mb to 66 Kb. This conversion made image handling much more manageable for the ConvNN training. An example image is presented in Figure 10.

In addition to considerations of the image file sizes, care had to be given as to the number of images given to the network representing each of the different levels of DR. *Kaggle* has provided training images in five installments. Filtering through the labels for all training images, it is seen from Figure 11 that most of the data is heavily skewed toward lower levels

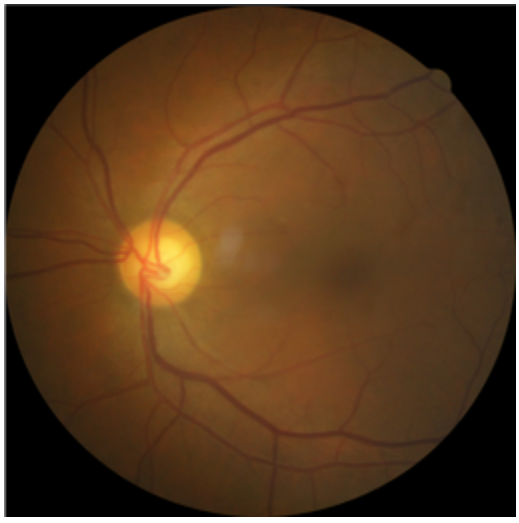


Figure 10: A down-sampled and cropped *PNG* image

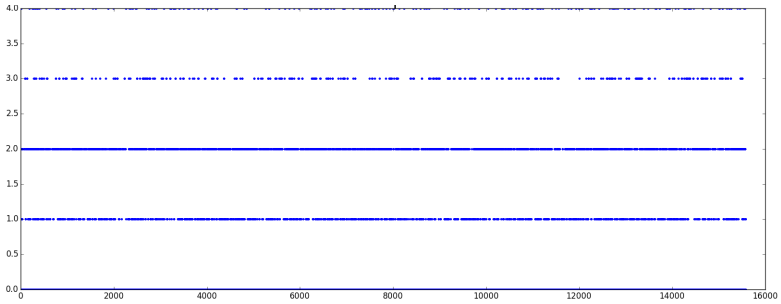


Figure 11: All the labels of the training data

of DR, (level 2 and below). To create a more robust training set, equal number of samples of each DR level is being fed to the ConvNN.

A code was written to filter the images from the training set, giving the distribution show in Figure 12.

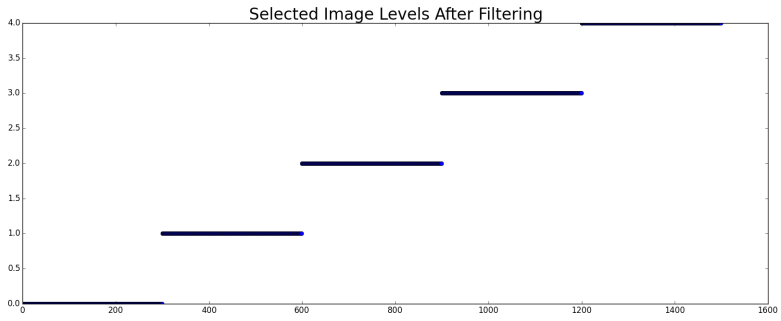


Figure 12: Equal number of DR samples for each category

4.2 Customized *AlexNet*

Using *Digits*, two popular ConvNN documeted in the literature were tested. These were the *GoogLeNet* [11] and the *AlexNet* [12]. By testing data sets with these base implementations, the author was able to fine tune the number of images given to the network to represent each level of DR. Using insight gained from these initial results, a customized *AlexNet*was created that had a larger initial filter size for the starting convolution layers. The aim was to be able to capture with greater accuracy the characteristic blotches representative of blood hemorrhaging in the retina. The customized *AlexNet* used for the final classification results shown in the next section is depicted in Figure 13 below.

5 Results

The results of the work will be presented in two parts: the first will be a series of binary-based results that help fine tuning the balance of images given to train the ConvNNs. The second set will be the final classification network that gave the best results overall.

5.1 Binary Classification Networks

To gain a better understanding of the number of images required to perform a robust training of a ConvNN, (which was evident based on ad-hoc initial attempts of throwing images at the networks and seeing dismal results), a binary study was done in which retinal scans with no DR where compared with retinal scans exhibiting DR, and the network was expected to discern between the two. These results are presented here.

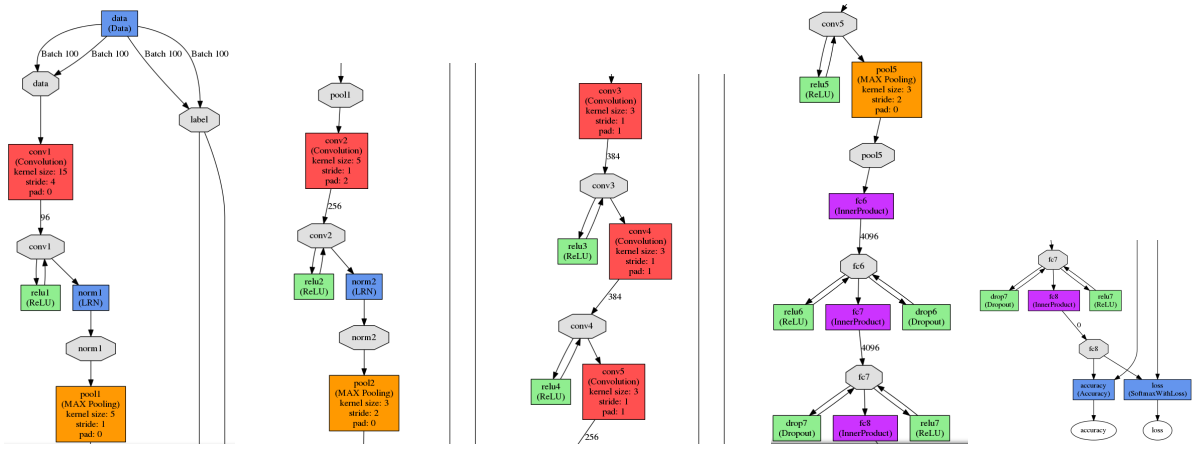


Figure 13: Customized AlexNet with larger intake filter size and pooling layers, (illustration should be read from left to right, top to bottom at each column)

5.1.1 DR 0 vs DR 4

To start the study off, scans with no DR, (i.e. Level 0), where compared to those showing the maximum level of DR at Level 4. The normalized confusion matrix for a data set composed of 200 images for each of the DR levels using a *GoogLenet* ConvNN is shown in Figure 14. The accuracy of the classification was 53.38 %.

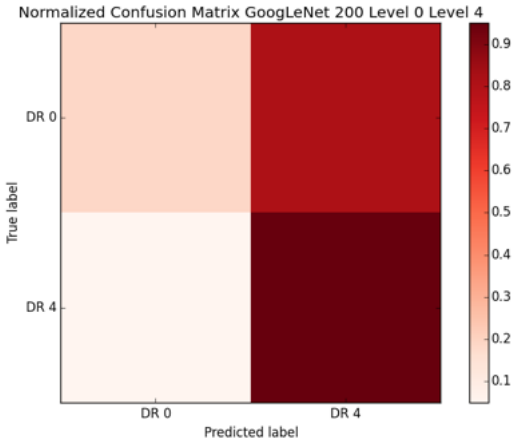


Figure 14: DR 0 vs DR 4 Normalized Confusion Matrix

The minimum certainty of the network for each classification, shown in Figure 15, shows a certain sparsity at the lower confidence regime for DR Level 4 classification. That indicates that the network is less confident when diagnosing Level 4 versus diagnosing Level 0, indicating that number of images used in this training set should be increased.

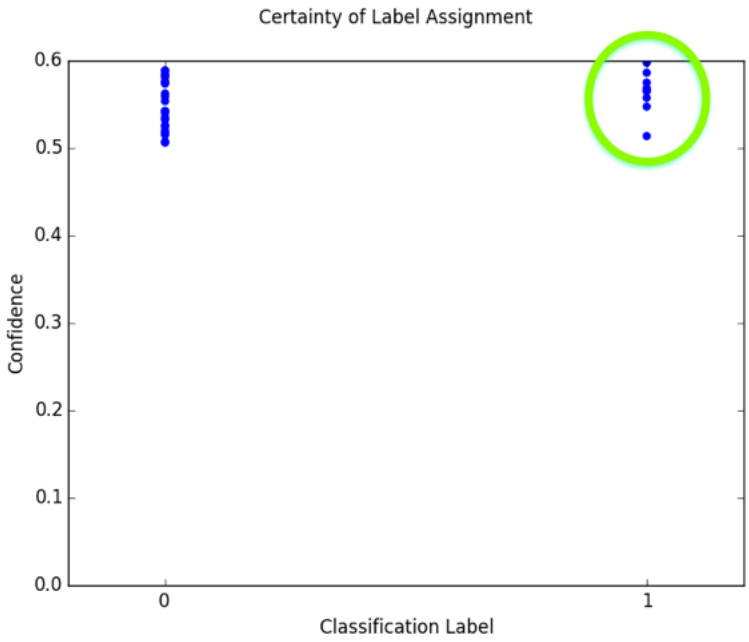


Figure 15: DR 0 vs DR 4 Certainty Comparison

5.1.2 DR 0 vs DR 3 & DR 4

Increasing the number of images to 400 for each level of DR to be classified, and adding DR Level 3 images to the mix, showed an improvement in the accuracy of the network. Using 400 images for each DR level, and combining together levels 3 and 4 to see if the network can use a more detailed feature space of the presence of DR, a *GoogLeNet* was able to achieve 55.15 %. The normalized confusion matrix and certainty plot are shown in Figure 16.

The results indicate some improvement in the certainty distribution of the network, showing less sparsity in the minimum confidence required to issue a classification. The next step was to add an even greater number of images to each level of DR.

5.1.3 DR 0 vs DR 1-4

Using 1000 images for Level DR 0 and roughly 600 images per level of DR for Levels 1 through 4, a binary training was done using a *GoogLeNet* once again. The results are shown in Figure 17. The overall accuracy for the network was 58.70 %.

The results indicate a uniform certainty for each classification, (no DR versus some level of DR), so a move was made to due actual diagnosing of DR level, which is done next.

5.2 Category Classification Network

Studying the results of the binary classification, a customized *AlexNet* was created as discussed in the previous section and illustrated in Figure 13. The primary modification to the base *AlexNet* given in [12] was to increase the size of the first layer convolution's kernel so as

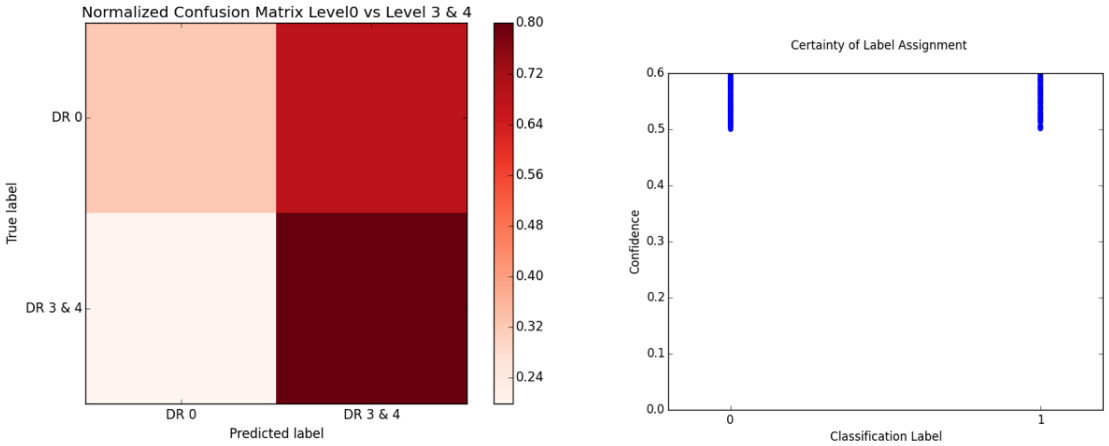


Figure 16: Normalized Confusion Matrix and Certainty Distribution for DR 0 vs DR 3 & 4 Training Set)

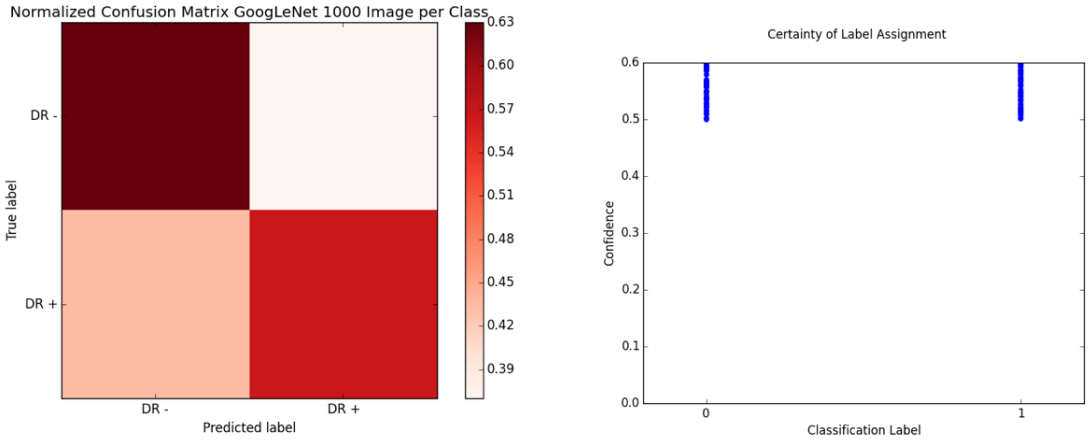


Figure 17: Normalized Confusion Matrix and Certainty Distribution for DR 0 vs DR 1-4)

to try to capture with greater fidelity some of the blood blotches so frequently seen in DR retinal scans. Using this customized network, the normalized confusion matrix and certainty distribution shown in Figure 18 were achieved. The overall accuracy of the trained network was 39.32 %.

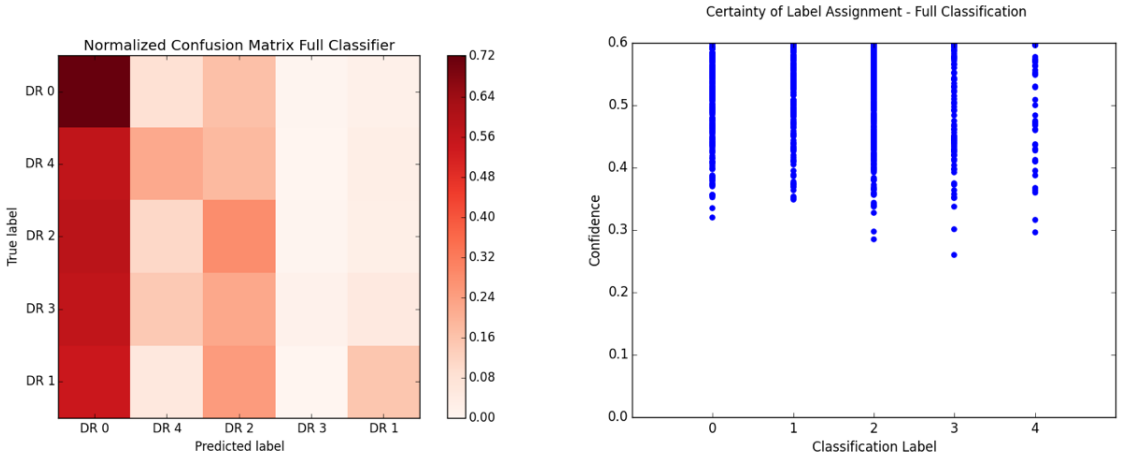


Figure 18: Normalized Confusion Matrix and Certainty Distribution for Customized *AlexNet*)

The overall confidence indicate that there is still substantial room from improvement, but it is interesting to note that the network seems equally certain in issuing diagnosis across the different levels of DR, indicate a decent balance in the image set. Future avenues to explore will be improving the filtering of the pre-processing of images and further customizing of the initial convolution layers of the network. The next series of figures is the 4 images of each level of DR that the network has the **most** confidence in labeling.

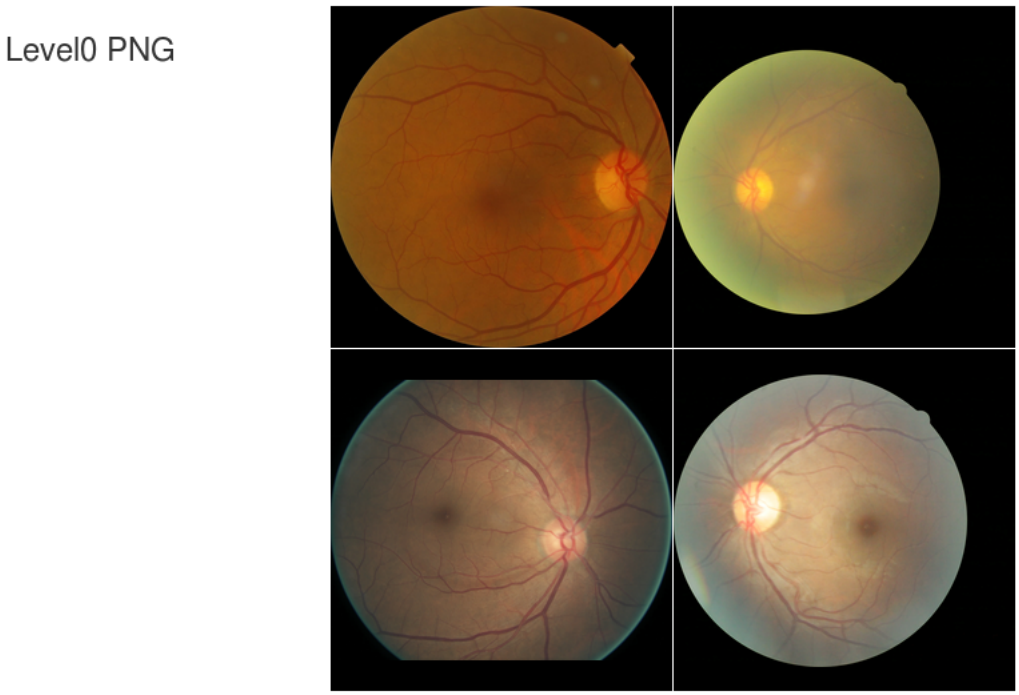


Figure 19: Top Four Confidence Diagnosis of Level 0

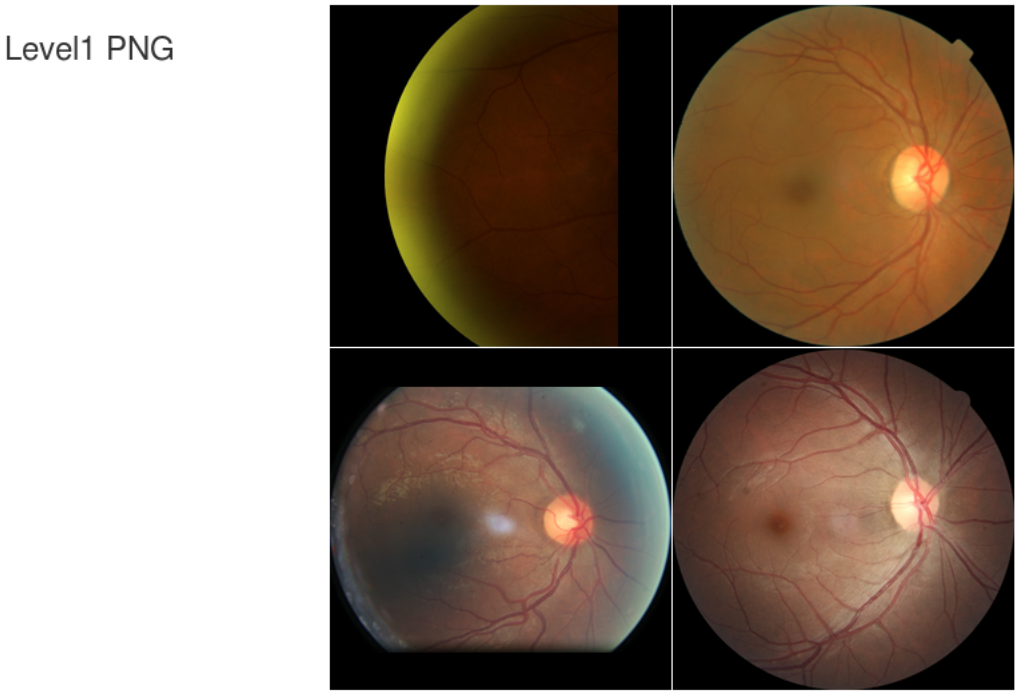


Figure 20: Top Four Confidence Diagnosis of Level 1

Level2 PNG

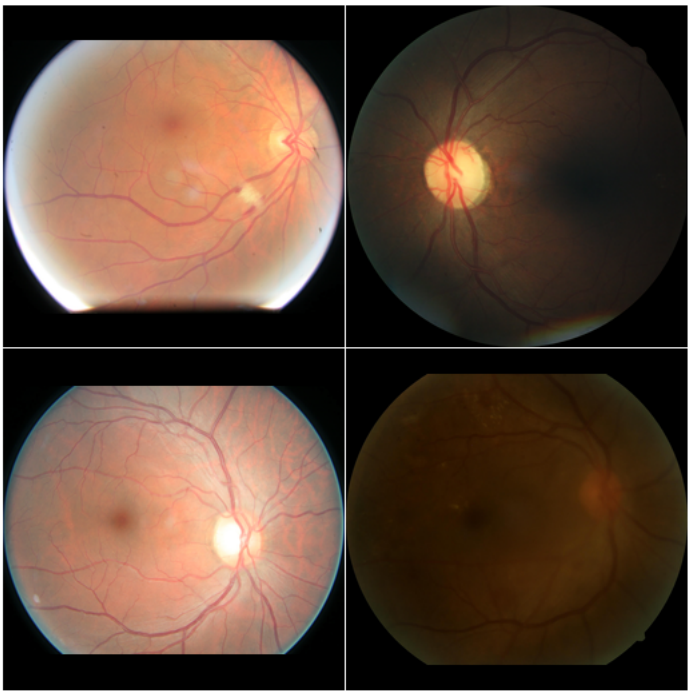


Figure 21: Top Four Confidence Diagnosis of Level 2

Level3 PNG

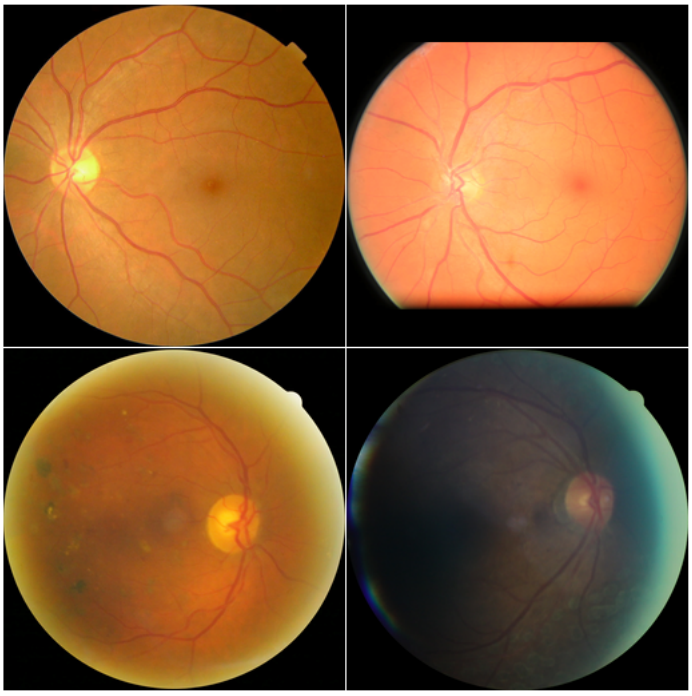


Figure 22: Top Four Confidence Diagnosis of Level 3

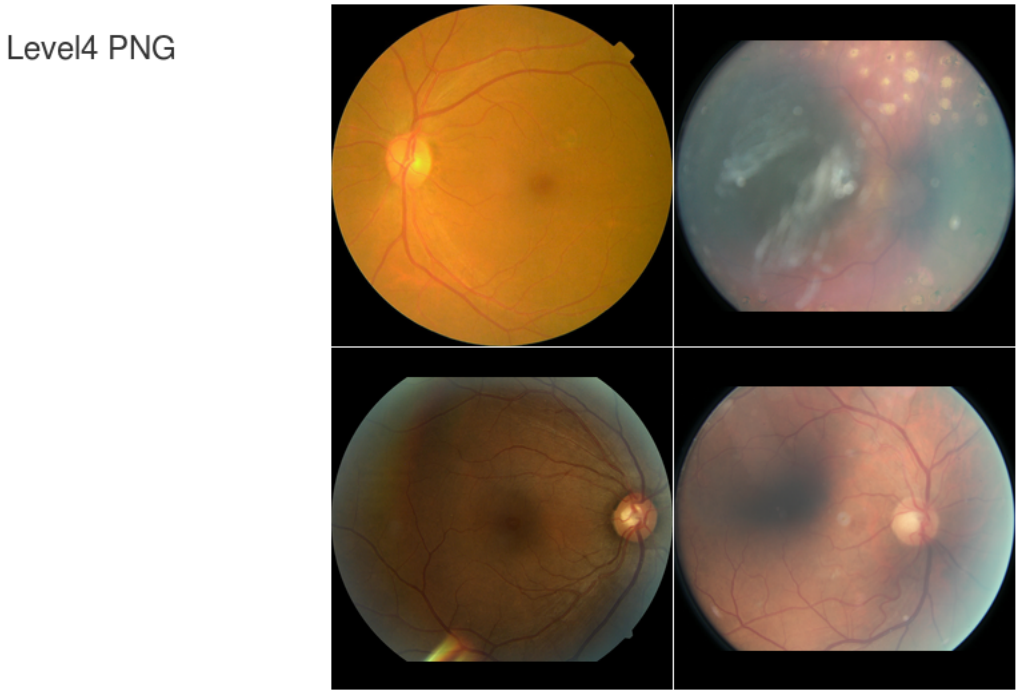


Figure 23: Top Four Confidence Diagnosis of Level 4

6 Conclusion

This was a very interesting and challenging project. Despite that the final results of the trained network being relatively low compared to the leader board posted on *Kaggle*, (which at the time of this report was $\approx$ 85 %), significant insight was gained into ConvNN implementation and application as a result of the work that went into project.

References

[1] Kaggle. https://www.kaggle.com/.

[2] NIH National Eye Institute. https://nei.nih.gov/health/diabetic/retinopathy.

[3] Diabetic Retinopathy Wiki Entry. http://en.wikipedia.org/wiki/Diabetic_retinopathy.

[4] Bottou L. Bengio Y. LeCun, Y. and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 11(86):22782324, 1998.

[5] Theano Convolution Network. http://deeplearning.net/tutorial/lenet.html.

[6] IEEE Computer Society. http://www.computer.org/csdl/trans/tc/2013/04/ttc2013040631-abs.html.

[7] Stanford Computer Science Dept: Class 231 -Convolution Neural Networks. https://cs231n.github.io/.

[8] wikiBooks: Artificial Neural Networks. http://en.wikibooks.org/wiki/Artificial_Neural_Networks/Print_Version.

[9] Digits Deep Learning Architecture. https://developer.nvidia.com/digits.

[10] James Bergstra, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley, and Yoshua Bengio. Theano: a CPU and GPU math expression compiler. In *Proceedings of the Python for Scientific Computing Conference (SciPy)*, June 2010. Oral Presentation.

[11] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014.

[12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C.J.C. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.