Convolution Neural Network Approach to Diagnosing Diabetic Retinopathy

José Solomon *

Abstract

This is the first draft of the report for 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 purpose of this initial draft is primarily to describe how ConvNN work, and the inter-dependencies of its various functional elements.

Contents

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 functional principles as applied to

*jose.e.solomon@gmail.com

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 that feed the retina of the human eye, as illustrated in Figure 1 below.

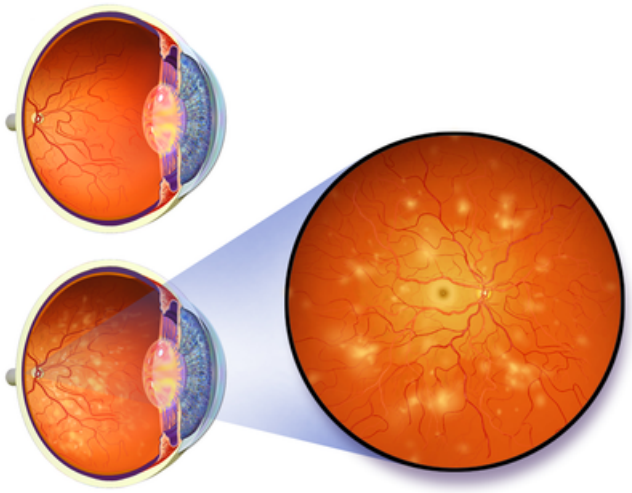


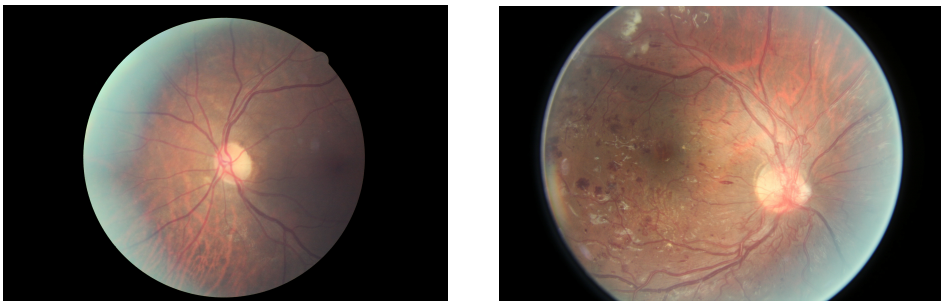
Figure 1: Diabetic retinopathy in comparison to a normal eye [3], (shown at the top left)

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

To diagnose the condition, a retinal scan of the eye is taken and a classification is assigned depending on a combination of the number of abnormal vein branching seen, the general wall thickness of the blood vessels and the number of blood stains observed. DR is categorized on a scale from 0 to 4, and examples of level 0 and 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.

2.1 The Kaggle Grand Challenge

Kaggle has presented the 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 facilitate download and processing of the data set, Kaggle divided the image set into a training group, and a testing group, and the



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

Figure 2: Examples of level 0 and level 4 DR

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 de facto favorite language for image processing.

3 Convolution Neural Networks

ConvNN are a powerful machine learning technique that fall under the of 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, 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 and Fourier transform treatments as well. 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 an 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),

Figure 3: Digital image as a matrix [6]

than 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.

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 5.

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 **??**), 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 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.

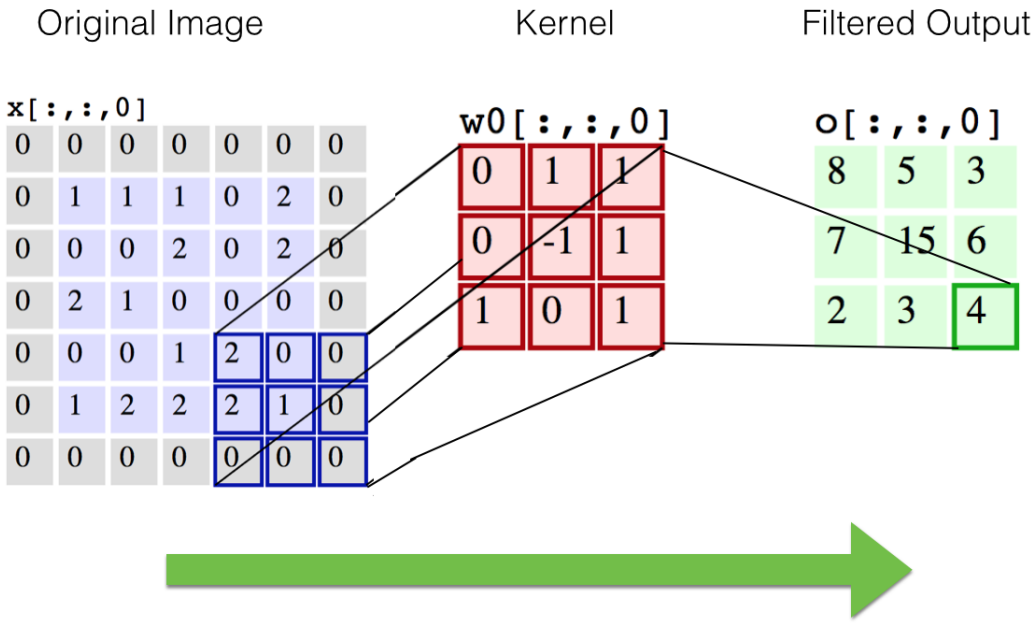


Figure 4: Convolution concept [7]

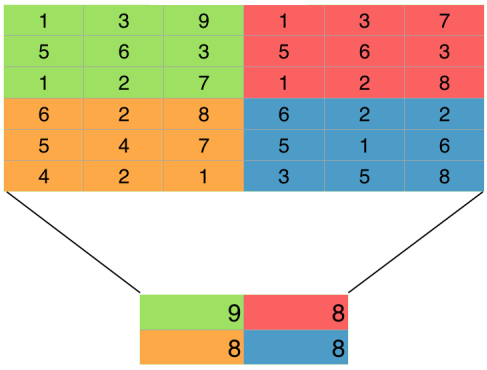


Figure 5: Max-pooling: the largest value of each sub-cell is fed forward

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.

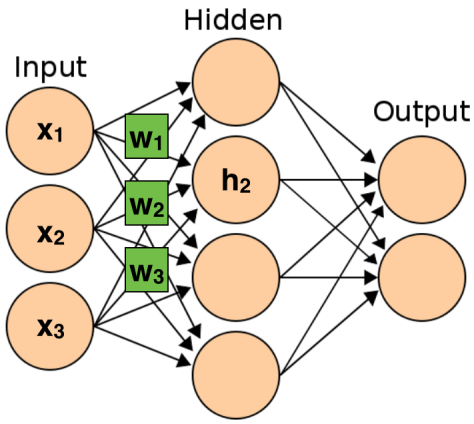


Figure 6: A fully connected neural network [8]

As can bee 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 upstream of it, as well as each node in the layer downstream.

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

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

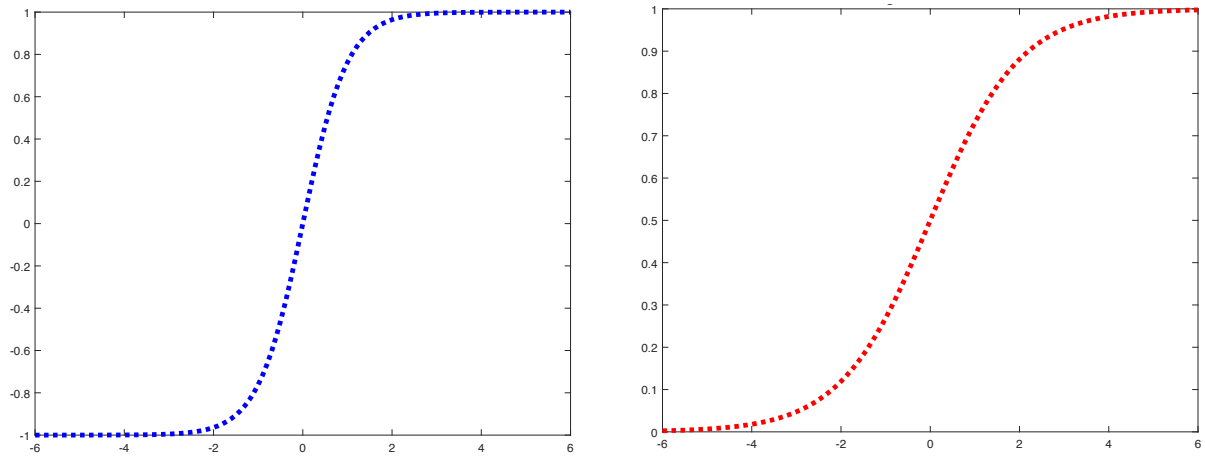
We note that the dot product shown above is the input to a function, known as the activation function, that in turn is the input value of the downstream node. There are a variety of activation functions that can and have been used in NN. One of the most popular, and the one which is used here, is the *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 bee 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.



(a) Tanh Function (b) Sigmoid Function

Figure 7: NN activation functions

These two functions are very similar in profile, but the *tanh* function is more commonly used due to its favorable back-propagation properties, (i.e. the learning process of the NN). Back-propagation is a key concept to NN and ConvNN, but its description is beyond the scope of this first draft and will most likely be discussed in the final draft of the current document.

3.4 Logistic Regression Node

The final layer of the ConvNN is the logistic regression function node, which is a classifier we have reviewed in class. It's functional form is [5]

$$y_{prediction} = \text{argmax} P_i(Y = i|\{x, W\}) \tag{6}$$

where the probability is

$$P_i = \frac{e^{\vec{w}_i^T \vec{x}_i}}{\sum_j \vec{w}_j^T \vec{x}_j} \tag{7}$$

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).

4 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 current work, two repetitive blocks of convolution layers and max pooling layers have been used to process the input image.

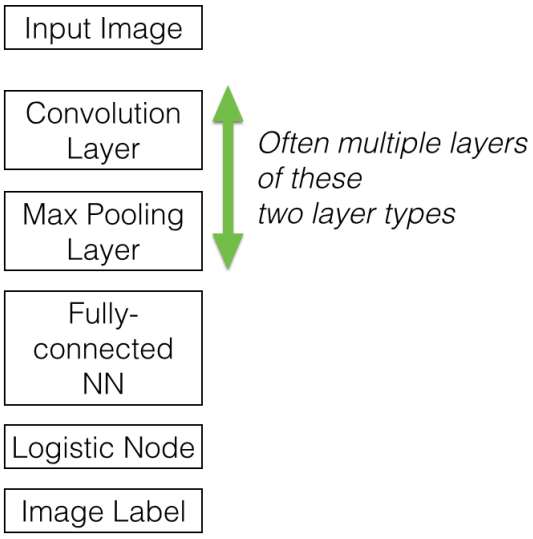


Figure 8: The ConvNN as a whole

5 Current Project Status

Python is a powerful and flexible language, but it is not entirely ideal to deal with image processing. To manage the challenges of such large images in conjunction with the issues related to memory management of Python code, a subsection of the training image data set was used to begin code development. To deal with these issues, an instance on EC2, (*Elastic Compute Cloud*), was created to run the code base and train the ConvNN. This work is still ongoing. The currents status update will consist of two parts. The first part will review the code that has been written up to now and the libraries that are being used. The second section will discuss issues relevant to image preparation and the current data set being used.

5.1 Code Base

Initial attempts at writing an entire ConvNN from scratch proved to be too complex for the allotted period of time. Especially in consideration that a significant amount of the initial code development centered on loading and processing the retinal image scans. To process images, the *openCV* computer vision library was 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)
```

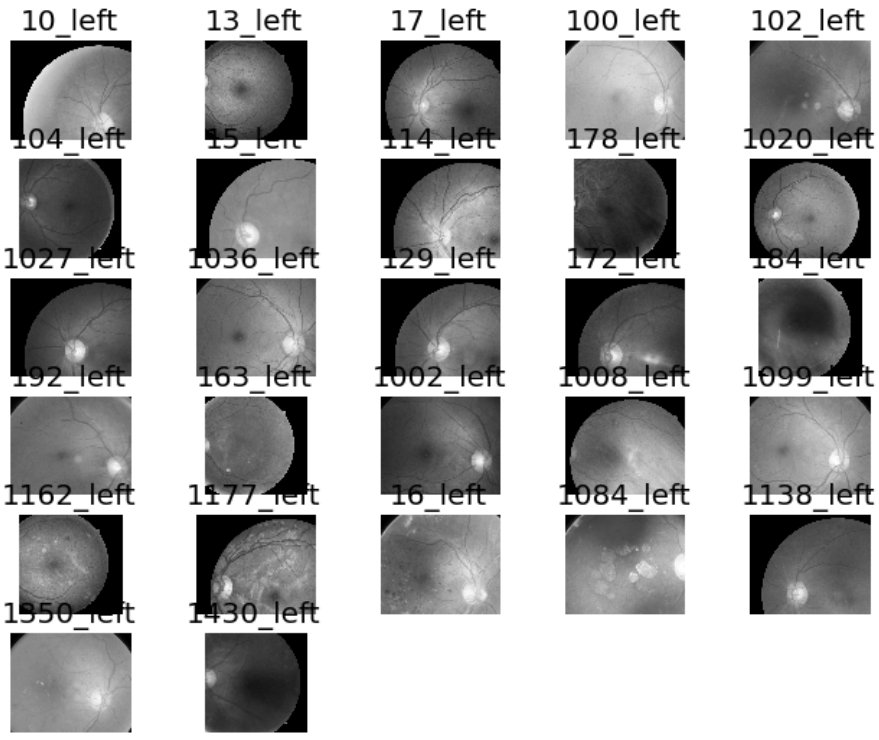


Figure 9: Loaded, cropped, and converted to gray-scale

Manual labels were defined for each of the training, validation and testing data sets. These data sets consists of the image names themselves and the labels that correspond to them. Data sets are prepared by converting 2D image matrices into 1D row vectors, and then segregating these images into the three distinct groups.

```
# Create a database with labels and images
    nPixels = PIXELDIM*PIXELDIM
    dataSetLeft = np.zeros([nImagesSubset,nPixels])
    count = 0
    for i in range(0,nImagesSubset):
        image = np.reshape(iL.lImageList[i],iL.lImageList[i].shape[0]\
                           *iL.lImageList[i].shape[1],1)
```

```
        print i
        dataSetLeft[count,:] = image[0:nPixels]
        count += 1

    # Index arrays
    trainIndex = [0,1,2,3,6,7,8,12,13,16,17,18,19,22,23,24]
    validIndex = [4,9,10,14,20,25]
    testIndex = [5,11,15,21,26]

    # subset data sets
    trainSetL = prepareData(trainIndex, nPixels, dataSetLeft)
    validSetL = prepareData(validIndex, nPixels, dataSetLeft)
    testSetL = prepareData(testIndex, nPixels, dataSetLeft)
```

It is noted that the manual index labeling for the different groups is only for code prototyping, and once a more robust code is finished, there will be more random selection for each of these data sets.

To define and train a ConvNN, the deep learning library *theano* was used to create a model. The initial architecture of the ConvNN is two layers of convolution layers, followed by one fully connected NN, with a final logistic regression node to do actual classification of the data. The pertinent sections of code are shown below:

```
# First convolution layer
 layer0 = LeNetConvPoolLayer(
        rng,
        input=layer0_input,
        image_shape=(batch_size, 1, nPixels, nPixels),
        filter_shape=(nkerns[0], 1, 15, 15),
        poolsize=(12, 12)
    )
# Second convolution layer
    layer1 = LeNetConvPoolLayer(
        rng,
        input=layer0.output,
        image_shape=(batch_size, nkerns[0], 153, 153),
        filter_shape=(nkerns[1], nkerns[0], 8, 8),
        poolsize=(2, 2)
    )
 # Fully connected layer
     layer2 = HiddenLayer(
        rng,
        input=layer2_input,
        n_in=nkerns[1] * 73 * 73,
        n_out=5,
        activation=T.tanh
```

```
    )
    # Classification layer
    layer3 = LogisticRegression(input=layer2.output, n_in=5, n_out=5)

    # Entire network
    params = layer3.params + layer2.params + layer1.params + layer0.params
```

This concludes the discussion on the current code base. The next section will discuss some elements of the data set preparation.

5.2 Image Preparation

Kaggle has provided training images in 5 8 Gig installments. Filtering through the labels for all training images, we see from Figure 10 that most labels are heavily skewed toward lower levels 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.

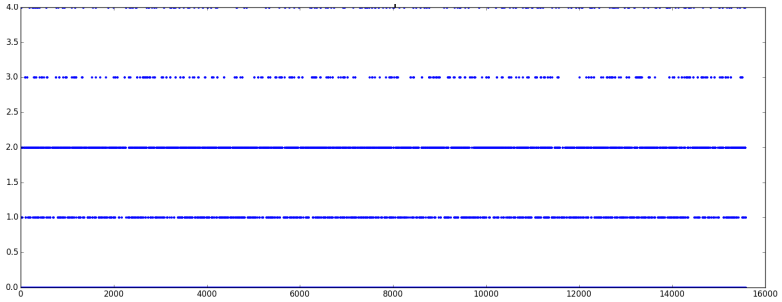


Figure 10: All the labels of the training data

A code was written to filter the images from the training set, giving the distribution show in Figure 11.

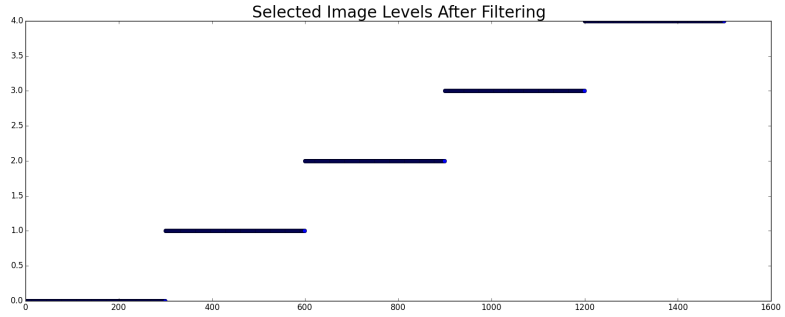


Figure 11: Equal number of DR samples for each category

For code prototyping, a subset of the images in this filtered data set is being used to train the network.

6 Conclusion

Training of the network in ongoing. A variety of challenges still remain, (batch processing on EC2, saving various model files, improving ConvNN architecture, etc.), but hopefully significant strides at bringing the work to reasonable conclusion will be made over the next three weeks.

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`.