Diabetic Retinopathy Auto-Detection

I. Introduction：

Diabetic Retinopathy(DR) is one of the complications caused by diabetes. It usually caused by high glucose concentration in the blood vessels which providing retina nutrients, and it may resulted in many other complications as well. In 2015, WHO estimated that there are 415 million people worldwide have diabetes. To detect DR, patients need to take photos of their retina by fundus photography in hospitals. But, traditional fundus photography requires eyes to be dilated, which is very inconvenient. Fortunately, there are more advanced technologies recently that enable less demand of dilation and other preparations. However, even though with high resolution retinal fundus photos, diagnosis of them still require highly-trained medicinal experts with their eyes and labors. If there exists an auto-detection system that can classify photos with respect to their DR severity, we believe that it will be enormous assistance to those medical professionals. Therefore, with the opportunity and resources provided by our instructor, we chose Convolutional Neural Network(CNN) to make attempts to build this DR auto-detection system.

II. Algorithm：

We utilized open-source code from the team who won the first place in Kaggle competition of Diabetic Retinopathy Auto-Detection. This team classified retina photographs with Spatially-sparse Convolutional Neural Network (SparseConvNet). Then, using the trained CNN models to ensemble into a Random Forest classifier, they generated the result of classification.

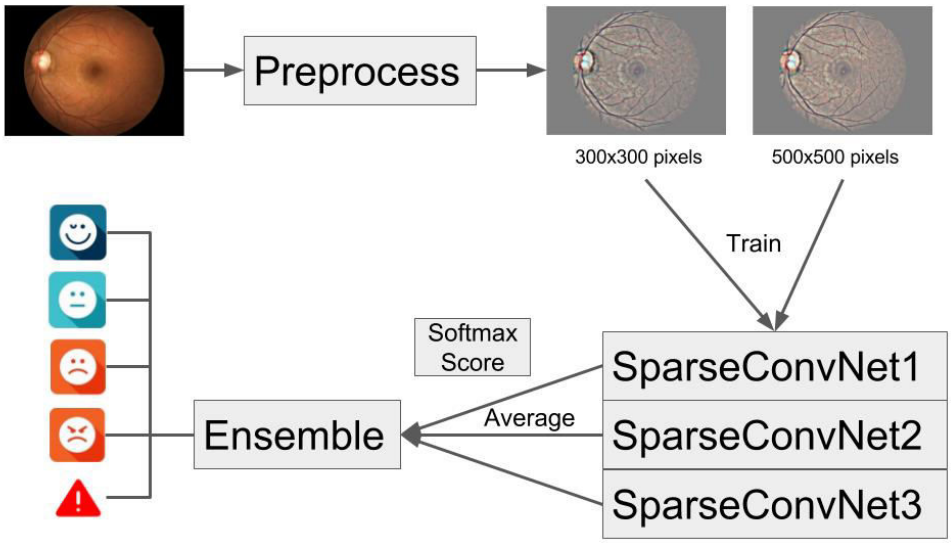
Algorithm Features:

1. Spatially-sparse Convolutional Neural Network (SparseConvNet): It is a kind of CNN which takes advantage of the sparsity of input images to increase its training efficiency and its capability to more layers without sacrificing training time. Therefore, the input images need to be preprocessed into spatially-sparse matrix for feeding into the SparseConvNet.
For more formal definition: see Spatially-sparse convolutional neural networks by Benjamin Graham(arXiv:1409.6070v1).

2. Fractional Max-Pooling (FMP): It is a kind of pooling technique which generate a sequence in ones and twos randomly or pseudo-randomly. And based on this sequence, distorts pooling window size and increments pooling window stride. The FMP Shrink parameter in the source code represents the setting of the fractional max-pooling layer. The parameter ranges from (1, 2), and it also determine how the random or pseudo-random sequence being generated. (For example, if FMP Shrink = 1.8, then $N_{in}/N_{out} = 1.8$, we let $N_{in}=18$, $N_{out}=10$, then the sequence generated can be “2212212222”. Its sum will equal to 18, and it will consist of 10 numbers) In this CNN configuration, it utilizes pseudo-random and overlapping FMP. For more formal definition: see Fractional Max-Pooling by Benjamin Graham(arXiv:1412.6071v4).

Overview:

The flowchart of the CNN we utilized. With fundus photos as input and five different severities as output.



Data Input：

Color retinal fundus photographs. (Training and Testing data are provided by EyePACS) (4000x3000 px / image)

Data Preprocessing：

1. Rescale input images to 300x300 or 500x500 pixels.
2. Subtract the local average color, and the local average gets mapped to 50% gray scale.
3. Clipped the images to 90% size to remove the “boundary effects”.

Data Augmentation：

This step is to increase the variability of training data for enhancing performance.

1. Randomly scale the size of images by ±10%
2. Randomly rotated by between 0° and 360° degrees。
3. Randomly skew the images by ±0.2 .
4. For testing, the images are just rotated randomly.

Network Configuration：

We have codes about the top three CNN configurations. Two of them(**kaggleDiabetes1** and 2) utilized images of 300x300 pixels as training data, and the other(**kaggleDiabetes3**) utilized that of 500x500 pixels. The differences among them are the layer configurations. But Each layer in each CNN all uses Vleaky RELU as activation function. The detailed configurations are the following:

kaggleDiabetes1：

Input images will first go through 7 sets of layers. Each set of layers consists of various features and filter size. All of the sets use fractional max pooling layer with same shrink proportion. Detail configuration is the following:

Features(Number of filters)	Filter Size	Filter Stride	Pool Size	FMP Shrink
32	5	1	3	1.8
64→96→128→160→192→224	3	1	3	1.8

Next, the resulting feature map will go through two more sets of layers with dropout:

Features	Filter Size	Filter Stride	Pool Size	FMP Shrink	Dropout
256	3	1	3	1.8	32.0 / 256
288	3	1	2	1.5	32.0 / 288

Then it will go through two sets of layers with max pooling layer(with dropout)：

Features	Filter Size	Filter Stride	Pool Size	Pool Stride	Dropout
320	2	1	1	1	64.0 / 320
356	1	1	1	1	64.0 / 356

Last, the feature map is outputted by softmax layer.

kaggleDiabetes2：

Input images will first go through 10 sets of layers. Each set of layers consists of various features and filter size. All of the sets use fractional max pooling layer with same shrink proportion. Detail configuration is the following:

Features	Filter Size	Filter Stride	Pool Size	FMP Shrink
32	5	1	3	1.5
64→96→128→ 160→192→224→256→288→320	3	1	3	1.5

Next, the resulting feature map will go through two more sets of layers with dropout:

Features	Filter Size	Filter Stride	Pool Size	FMP Shrink	Dropout
352	3	1	3	1.6	32.0 / 352
384	3	1	2	1.5	32.0 / 384

Then it will go through two sets of layers with max pooling layer(with dropout)：

Features	Filter Size	Filter Stride	Pool Size	Pool Stride	Dropout
416	2	1	1	1	64.0 / 416
448	1	1	1	1	64.0 / 448

Last, the feature map is outputted by softmax layer.

kaggleDiabetes3：(This CNN model is trained by images of 500x500 pixels)

Input images will first go through 14 sets of layers. Each set of layers consists of various features, filter size, filter stride, pooling size and stride, and dropout. All of the sets use max pooling layer. Detail configuration is the following: (i = 1~7)

Features	Filter Size	Filter Stride	Pool Size	Pool Stride	Dropout
32*i	3	1	1	1	0.0
32*i	3	1	3	2	0.0

It will go through three sets of layers with max pooling layer:

Features	Filter Size	Filter Stride	Pool Size	Pool Stride
288→288	2	1	1	1
320	1	1	1	1

Last, the feature map is outputted by softmax layer.

These three CNN models output softmax scores with respect to DR severity(0~4).(eg. If output scores are 0.292, 0.378, 0.250, 0.040, 0.040. Then the score of 0(No DR) is 0.292, and the score of 1(Mild)is 0.378, etc.)

Network Ensemble:

We simply average the Softmax scores from these three CNN models. Although the original team use trained CNN models to train random forest classifier, the accuracy is acceptable even without the utilization of random forest.

Output:

Averaged softmax scores of each severity(0~4). The result of severity is the maximum of the softmax score in each image.

III. Architecture：

We utilized the cluster and NVIDIA® Tesla™ K40C graphic cards in the cluster provided by our instructor to implement our CNN training, testing and classification. The cluster enable us to work simultaneously and independently. Moreover, the GPU in the graphic card reduce our computational time significantly.

Furthermore, the source code we have implemented requires cuDNN package which require GPU to compute our CNN model training. Therefore, without the assistance from the instructor, it would be impossible to utilize the CNN models in such short time.

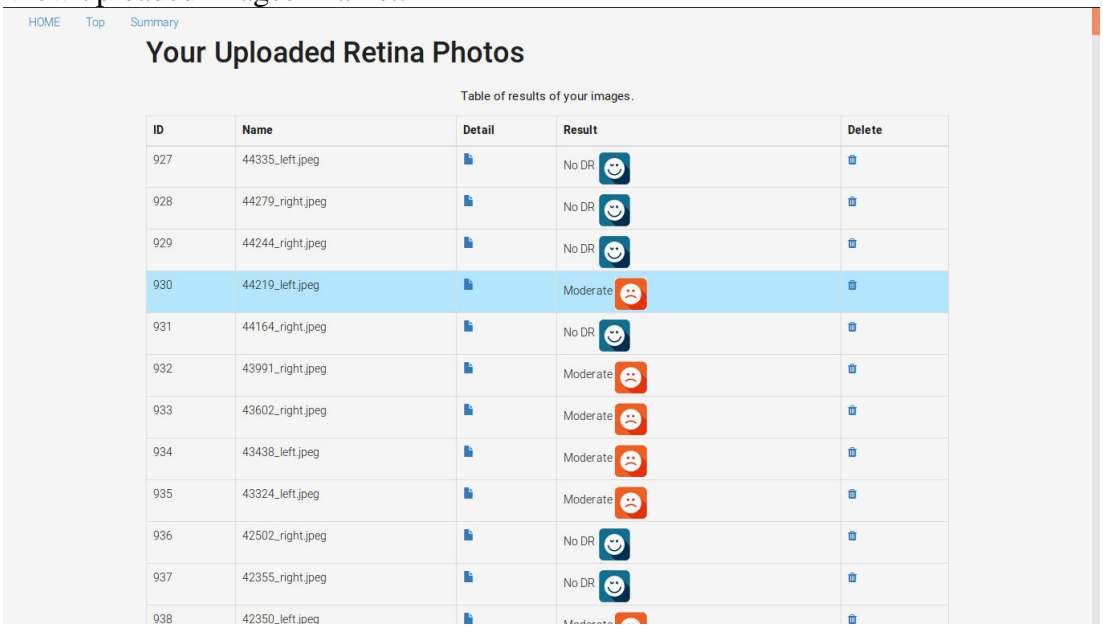
IV. Application:

We build a website, called “THE DEEP EYES”, using django framework. This website aims to enable users to upload their own retinal fundus photographs. After uploading the images, the server can provide five kinds of severity based on the classification results of our CNN models. It can also provide some basic functionalities, such as viewing uploaded images in table form, inspecting each uploaded images, and simple summary of user-uploaded images. While

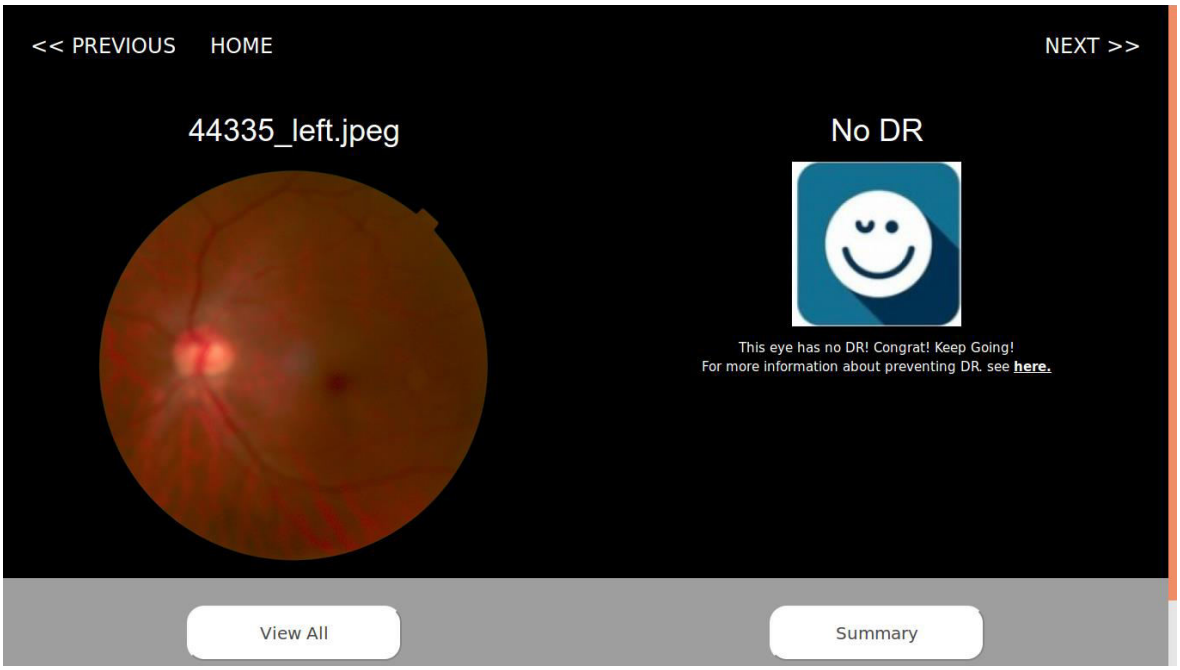
providing these basic functionalities, we also rendered intuitive user interface for anyone who visits this website.

Our primary thought is that when users upload their images, the images are sent to our website server and the cluster where our CNN models stored. Then after classification, the result file will be sent to populate the database of our website server. Finally, website will display the classification results and original images. However, this real-time classification is yet to be done. Therefore, the following views were accomplished by pre-populating the database with pre-classification results. There are still many difficulties to overcome.

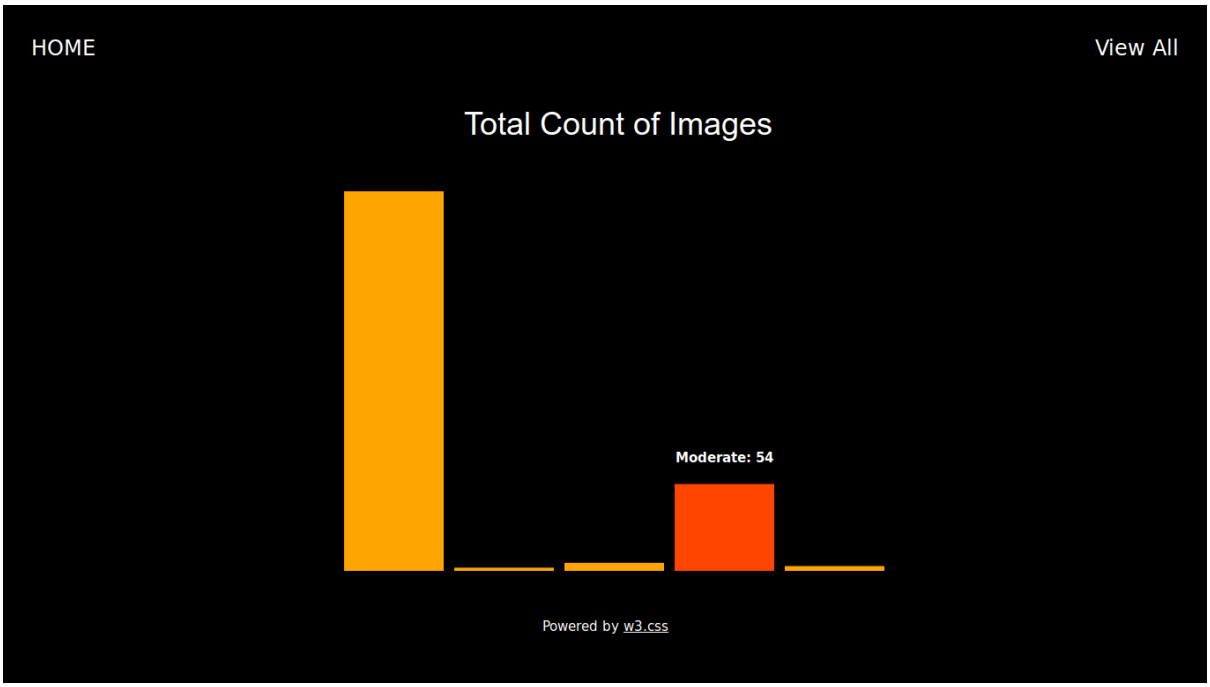
View uploaded images in a list:



View each uploaded image (Suggestions based on severity also displayed):



Total severities counts in barchart:



V. Future Works:

Our website, THE DEEP EYES, is far from complete. There are still various aspects to enhance it. For example, the classification by CNN models and uploads to server simultaneously

have yet to complete. Furthermore, this CNN model can achieve approximately 90% accuracy with testing data provided by Kaggle. But sampling small set of images gives less accurate results. The auto-detection system still has room to improve its performance in order to further assist professional clinicians. The whole website codes and some documentation were uploaded to github repository(https://github.com/jr55662003/Diabetic-Retinopathy-website) for anyone interested in this website.

In addition to improving algorithms or completing THE DEEP EYES website, there still exists possibilities of utilizing this CNN configurations to other kinds of images. Is it possible to train this CNN model to classify X-ray images? Or is it possible to utilize this CNN configurations to various kind of medicinal images? or any other images? Due to our limited resources and time, we did not explore these possibilities, but we think it is interesting to point out for those who want to dive into it.