Mobile Assisted Diabetic Retinopathy Detection using Deep Neural Network

Shorav Suriyal, Christopher Druzgalski, Kumar Gautam
Department of Electrical Engineering, California State University Long Beach, CA, USA
Email: shoravsuriyal@yahoo.com; Ph. (562) 338-1961

Abstract — **Diabetic retinopathy (DR) is a major microvascular complication resulting from diabetes and continues to have a serious impact on global health systems. Globally about 95 million people suffer from DR. This paper focuses on detection aspects of a mobile application developed to perform DR screening in real time. The application is powered by a tensorflow deep neural network architecture that is trained and tested on 16,798 fundus images. These images are preprocessed to remove noise and prepare them to be fed into neural network. Preprocessing steps involve averaging all the images using a 5x5 filter to improve the quality of images and then these images are resized to 256x256 pixels. After preprocessing the input dataset is fed into the neural network. The convolutional neural network model used in this project is MobileNets, which is used for mobile devices. The neural network has 28 convolutional layers and after each layer there is batchnorm and ReLU nonlinear function except at the final layer. The output from last layer is a class label either DR or no DR. The final accuracy of the model is 73.3%. This model is optimized to work on mobile devices and does not require Internet connection to run.**

Keywords — **Diabetic Retinopathy, convolutional neural network, deep learning, tensorflow.**

I. Introduction

Diabetic retinopathy is a serious eye disease that can lead to vision problems and ultimately lead to blindness if not treated in time. However, early diagnosis is difficult many persons because they are not aware of DR especially in the early stages. Presently, the diagnosis of DR is a slow and cumbersome process. Additionally, the diagnosis requires specialized clinical visit. To overcome these challenges a mobile application was developed using a deep neural network that is trained on 16,798 fundus images of both the left and the right eye. The image dataset is using Kaggle's database [1]. The application was made to do the binary class categorization on the input images that are fed to the application via a mobile camera in real time. The classification category includes: DR and no DR [2]. The image dataset includes 26% of the DR and 74% of no DR images. Once the model is trained and tested it can be deployed for use on the android platform. Training and testing of the model is done in a Linux based system while Android development is using a windows operating system. The application is optimized to work on mobile devices by normalizing the weights and freezing some variables. This process of optimization makes the graph file 4 times lighter as compared to a traditional neural network architecture such as Inception or VGGNet [3].

II. Methodology

Two steps implementation focuses on this method's practical applicability. Specifically, the first step is to collect the required dataset. The second step is to sort the images from the dataset because all the images are zipped in one file and they need to be organized in separate folders with correct class labels. This organization of image data helps in training the neural network. Once the data is organized it is ready for preprocessing which involves two steps:

1) *Averaging*: The images in the dataset contain noise such as: low contrast, color variation, and uneven light reflections. To make images more consistent and smooth, a convolution filter is used. The size of the filter is 5x5. Due to smaller variance image is smoother with kernel of size 5x5 as compare to other size kernel. The technique used in averaging is known as Box blur.

2) *Resize:* The images in the dataset are of size 4928x3264 pixels. Due to such big size of the image it is not suitable to feed into the neural network as it can increase the computation time. So all the image in the dataset are resized to 256x256 pixels. Resizing is accomplished by using bilinear interpolation.

All preprocessing and post-processing task on images are done in Python 2.7.12 with necessary dependencies installed. After preprocessing the data, it is then ready to be fed into a neural network. The neural network model used in this project is the MobileNets, which has 28 convolution layers. The details of architecture are discussed in section III of this paper. Once the data is inputted to network it will try to update its parameters such as weights and biases. The technique used in this project for making the neural network learn the input data is referred to as transfer learning. In this technique, a pre-trained network is utilized which was trained with millions of images from ImageNet dataset. This model is stored in the form of a graph file and one can make use of this graph file to generate a new graph file with updated weights and biases. This process is much faster and more efficient than creating a neural network from scratch and try

ISBN: 978-1-5386-5475-0
IEEE CATALOG NUMBER: CFP1818G-ART

to fit such a large dataset to it can take several days of training and requires high GPU power.

Implementing convolutional neural network has become a popular method in the biomedical field. Furthermore, the neural network can be used in detecting brain tumors [4] and analyzing x-ray images. A convolution neural network if it is trained on a large dataset can be proved to work better than humans as described in Harvard Medical School and MIT joint study on classifying metastic breast cancer [5].

The developed Android application was tested in real time on test dataset images. Since the test dataset contains images of both categories of DR and no DR, so it is used as source for real time image analysis as one would be capturing image of an actual subject. Therefore, it's a process of acquiring image from the test database to show that this method will work for any subject. Once this process is done for all the test images, the accuracy, sensitivity and specificity of the neural network model are calculated using the formula given below:

$$accuracy = \frac{TP+TN}{TP+TN+FP+FN} \tag{1}$$

$$sensitivity = \frac{TP}{TP+FN} \tag{2}$$

$$specificity = \frac{TN}{TN+FP} \tag{3}$$

Here TP, TN, FP and FN denotes the true positive, true negative, false positive and false negative samples in the test dataset.

II. Preprocessing- Image Dataset

The images in the dataset contain noise such as varying lighting condition, low contrast, and different sizes. To make the image smooth one can use a 2d convolution filter. The one used in this project is a 5×5 filter:

$$F = \frac{1}{25}\begin{bmatrix}1&1&1&1&1\\1&1&1&1&1\\1&1&1&1&1\\1&1&1&1&1\\1&1&1&1&1\end{bmatrix} \tag{4}$$

The above filter adds all the pixels that come under it while running over the image and then takes the average of it. After that, it replaces the center pixel intensity with the new average value. This operation is applied to all the images in the training dataset. Fig. 1 shows a glimpse of dataset using Kaggle's database [2].

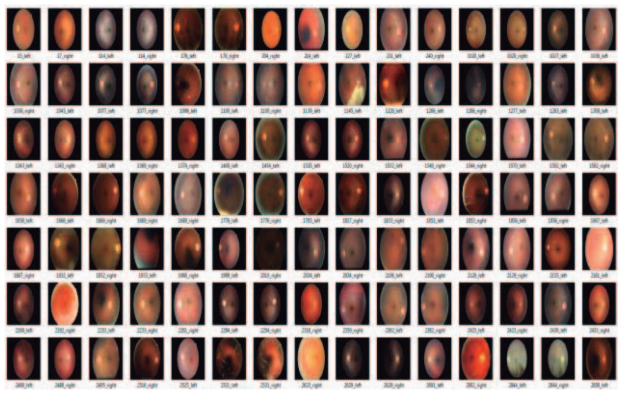
Fig. 1. A glimpse of fundus image dataset.

Once the filtering process is done, all the images are resized to 256×256 pixels. This process contributes to the neural network taking less time to train and makes the data consistent.

III. Neural Network Architecture

The utilized neural network architecture is based on MobileNets. This network is built on depthwise convolution layers which are further divided into depthwise and pointwise convolution, except for the first layer which is a fully connected layer. Depthwise convolution is used for applying a single filter on every input channel while pointwise convolution is used to form a linear combination of the output from the depthwise layer. There are two non-linearity used: batchnorm and ReLU after each layer [3]. The depthwise layer is represented as:

$$G_{k,l,m} = \sum_{i,j} K_{i,j,m} \cdot F_{k+i-1,l+j-1,m} \tag{5}$$

Where k is a depthwise kernel with a size of $D_K \times D_K \times M$.
Here m_{th} filter of k is applied to the m_{th} channel in F to give m_{th} channel of the feature map G.
Computation cost of depth convolution:

$$D_k \cdot D_k \cdot M. D_F \cdot D_F \tag{6}$$

Depthwise convolution is a way to filter the channels but it does not combine them in order to generate new features. So for combining we use 1×1 pointwise convolution. Fig. 2 shows the MobileNets architecture generated from tensorboard that is an inbuilt feature of tensorflow library [6] in Python.

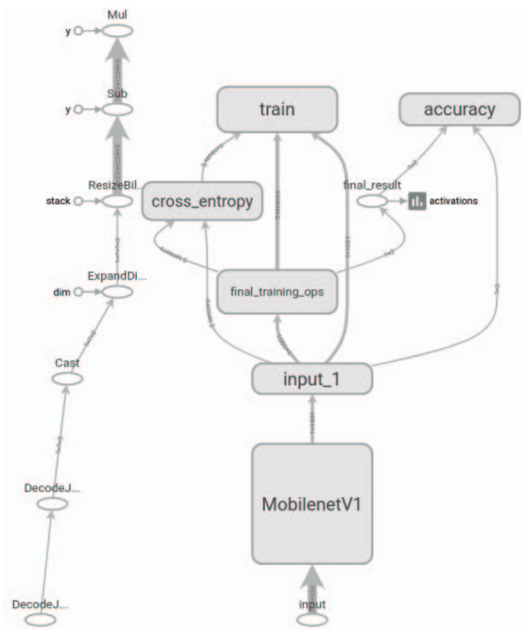

Fig. 2. MobileNets control flow model

The MobileNets training is done in Tensorflow [7] with the help of RMSprop [8] and asynchronous gradient descent. The layer architecture of the MobileNets model is given in Table I. In comparison to other models such as Inception, the MobileNets uses less regularization and data augmentation. In fact, the size of the input to the network is also small. The output of the neural network is class label as DR or no DR. The architecture is trained and tested using Python language with Tensorflow CPU library installed.

TABLE I
MOBILENETS ARCHITECTURE

Type/Stride	Filter Shape	Input Size
Conv/s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw/ s1	$3 \times 3 \times 3 \times 32$ *dw*	$112 \times 112 \times 32$
Conv/s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw/s2	$3 \times 3 \times 64$ *dw*	$112 \times 112 \times 64$
Conv/s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw/s1	$3 \times 3 \times 128$ *dw*	$56 \times 56 \times 128$
Conv/s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw/s2	$3 \times 3 \times 128$ *dw*	$56 \times 56 \times 128$
Conv/s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw/s1	$3 \times 3 \times 256$ *dw*	$28 \times 28 \times 256$
Conv/s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw/s2	$3 \times 3 \times 256$ *dw*	$28 \times 28 \times 256$
Conv/s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
5 × Conv dw/s1	$3 \times 3 \times 512$ *dw*	$14 \times 14 \times 512$
5 × Conv/s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw/s2	$3 \times 3 \times 512$ *dw*	$14 \times 14 \times 512$
Conv/s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw/s2	$3 \times 3 \times 1024$ *dw*	$7 \times 7 \times 1024$
Conv/s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool/s1	Pool 7×7	$7 \times 7 \times 1024$
FC/s1	1024×1000	$1 \times 1 \times 1024$
Softmax/s1	Classifier	$1 \times 1 \times 2$

IV. PREPROCESSING ANDROID APPLICATION

After training the neural network model one gets two files as output: graph file and class labels. Graph file contains all the nodes and operations that are performed during the training of the network. Since the final built application is to be run on a mobile device that has limited capability to perform operation one needs to optimize the graph file before building the application. The whole neural network is built using tensorflow library that has a built-in tool for removing all the nodes that are not needed for a given set of inputs and outputs. With the help of an optimized model, the number of calculations is reduced by merging the explicit batch normalization [9]. Images are in the form of a matrix with some numbers in it. If one wants to compress an image, the number of colors in the image needs to be reduced. Similarly, the graph file of the neural network can be reduced by quantizing the weights of the graph.

V. RESULTS

The accuracy of the model comes out to be about 73.3 %. Fig. 3 shows the accuracy graph that is generated using tensorboard where x and y axis representing the number of training examples and accuracy score [6]. The accuracy can be further increased by applying more preprocessing techniques to the image datasets. The possible image processing techniques may include thresholding, histogram equalization and, data augmentation. All these techniques can improve the accuracy of the model. For testing of the model 1000 random fundus images are classified. Images for testing purposes are not taken from the training dataset. The sensitivity of the model is 74.5 % and specificity is 63%.

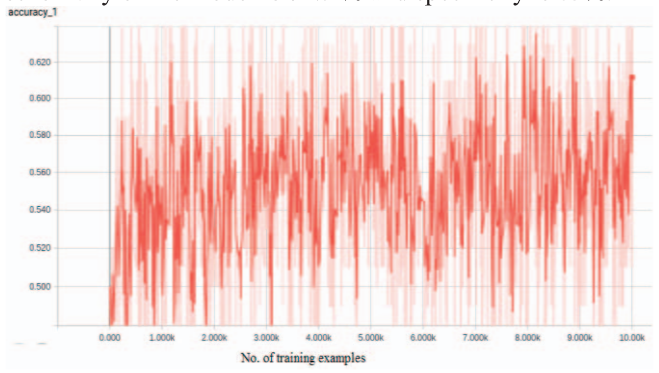

Fig. 3. Accuracy of MobileNets model

The cross entropy of the model also tells how well the model is performing. Fig. 4 shows the cross entropy graph generated using tensorboard where x and y axis representing the number of training examples and cross entropy score [6]. Since the MobileNets model outputs the probability of a particular class that varies between 0 and 1, so cross entropy

ISBN: 978-1-5386-5475-0
IEEE CATALOG NUMBER: CFP1818G-ART

increases if the model does not perform well. Cross entropy can be measured using log loss function as given below [10]:

$$L = -\sum_j y^{(j)} log\sigma(o)^{(j)} \quad (7)$$

Here y, o, σ are the true label, output of last layer in network and probability estimate.

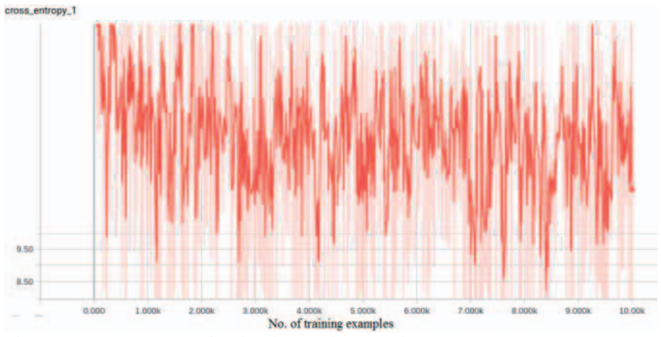

Fig. 4. Cross entropy of trained neural network.

For deployment of the application on Android one can use Android Studio or any other integrated development environment. The Android application built for this project is built on Android Studio 3.0.1 and the mobile device used for this project was LG Leon with Android version 4.0.1. The application was made to run in real time on test dataset images and images were acquired using the built-in camera of the mobile device. Once an image is captured it is fed into the neural network, which is then displays the output label as one of the two classes: DR or no DR. The output is then shown in the form of probability. Kashyap et el. [11] work on Diabetic Retinopathy detection utilizes mobile phone camera for capturing retinal images and then evaluated using artificial neural network but, it is a slow process as images needs to be transferred and analyzed separately on a computer rather than mobile. Bui et al. [12] used neural network for Diabetic Retinopathy detection with an accuracy of 85.54 %, but this method is performed on a computer system that has a greater computing power rather than a mobile device. Fig. 5 shows the screenshot of the Android application running on a mobile device, being a part of this project presented here.

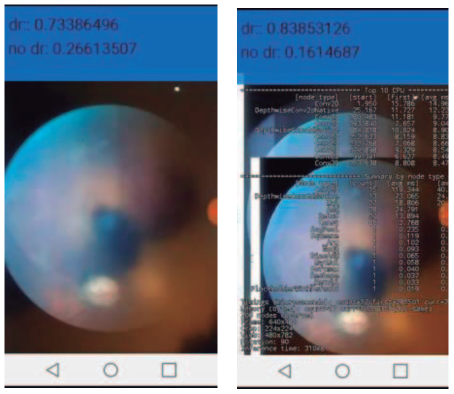

Fig. 5. Two screenshots of application in real-time.

VI. Conclusion

The described neural network model based Android application works well for classification of Diabetic Retinopathy. The application makes use of the robust deep neural network architecture MobileNets that is trained on millions of ImageNets images. With the help of transfer learning, one can retrain such a model with its own image dataset. The application was currently developed to work for Android devices but one can use this model in a windows or Linux operating system with the help Python programming language. This developed application can serve as a useful screening tool for Diabetic Retinopathy.

References

[1] Kaggle. (2015). Diabetic Retinopathy Detection.[Online]. Available:https://www.kaggle.com/c/diabetic-retinopathy-detection/data

[2] Deep CNNs for diabetic retinopathy detection. (2016). Stanford cs229.[Online].Available:http://cs229.stanford.edu/proj2016/report/tamkinusirifufu-Deep%20CNNs%20for%20diabetic%20retinopathy-report.pdf.

[3] Howard, A., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., . . . Adam, H. (2017). MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. arXiv:1704.04861 [cs.CV]

[4] Detection of brain tumor using neural network Amsaveni V., Singh N.A.(2013) *2013 4th International Conference on Computing, Communications and Networking Technologies, ICCCNT 2013*, , art. no. 6726524

[5] Wang, D., Khosla, A., Gargeya, R., Irshad, H., & Beck, A. (2016). Deep Learning for Identifying Metastatic Breast Cancer. arXiv:1606.05718 [q-bio.QM]

[6] Abadi, Agarwal, Barham, Brevdo, Chen, Citro, . . . Zheng. (2016). TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. arXiv:1603.04467 [cs.DC]

[7] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen,C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, et al.Tensorflow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow. org, 1,2015. arXiv:1603.0446 7 [cs.DC]

[8] T. Tieleman and G. Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA*: Neural Networks for Machine Learning, 4(2),2012.

[9] Tensorflow-for-poets-2. (2017) [Online].https://codelabs.developers.google.com/codelabs/tensorflow-for-poets-2/#3

[10] Janocha, K., & Czarnecki, W. (2016). On Loss Functions for Deep Neural Networks in Classification. *Schedae Informaticae, 25*, 49-59. arXiv:1702.05659 [cs.LG].

[11] Kashyap, Nikita ; Singh, Dharmendra Kumar ; Singh, Girish Kumar 2017 4th IEEE Uttar Pradesh Section International Conference on Electrical, Computer and Electronics, Oct. 2017, pp.463-467.

[12] Bui, T., Maneerat, N., & Watchareeruetai, U. (2017). Detection of cotton wool for diabetic retinopathy analysis using neural network. *Computational Intelligence and Applications (IWCIA), 2017 IEEE 10th International Workshop on,* 203-206.

ISBN: 978-1-5386-5475-0
IEEE CATALOG NUMBER: CFP1818G-ART