

# The Eye Fundus Image Analysis

Using Deep Convolutional Neural Networks

Mohammad Huzefa Shaikh

M.Sc. Data Science and Computational Intelligence  
8012085

**Abstract—** This paper explicates a study on the eye fundus Images, a retinal image analysis; which, as a result, provides segmentation of the retinal blood vessels. The Deep Convolutional Neural Networks were used to perform the extraction and analysis; which, in recent years, has been recognized as one of the most successful Artificial Neural Networks' methods, specifically for the feature extraction and the analysis of images. The CNN model was trained with the patches of the eye fundus images and the blood vessel ground truths (examples made by human-experts) of the same images. Various datasets of the eye fundus images are publically available, nonetheless, the DRIVE dataset was used in this experiment since it was easier to obtain.

## INTRODUCTION

Eye Fundus Image Analysis (EFIA) is the crux of diagnosis of the eye diseases such as diabetic retinopathy, macular degeneration, and glaucoma. Specifically, the diabetic retinopathy induced by diabetes can cause serious vision impairment (blindness). The retinal (i.e. light-sensitive tissue) blood vessels are mutated (affected) by the diabetic retinopathy, which results in a complete blindness [3].

In order to identify the diabetic retinopathy, the contusion (lesion) correlated with vascular deformities due to diabetes, needs to be identified in the blood vessels [3]. The current method requires trained ophthalmologist to assess eye fundus images to extract mutations in the blood vessels, such illustrations are time-consuming and sometimes inconsistent [3]. Until now, optometrists predominantly used such a method. These human assessments were serviceable for a few

diagnoses, however, they failed to deliver in many situations [2].

EFIA can be used to automatically predict the later mutations in the retinal blood vessels; such a prediction might help optometrists to conduct the forthcoming eye conditions. Deep Convolutional Neural Networks (CNNs) can enhance such diagnoses [1]. The advantage of having a 3-dimensional neuron architecture can provide deep segmentation and individual patch analysis [11]. ConvNets has produced medical diagnoses, in many cases, better than manual interpretations of the human experts [1].

## LITERATURE REVIEW

In the recent years, Deep Convolutional Neural Networks have been successful in the field of Computer Vision [3, 1, 2]. Especially, the biomedical image segmentation has become a popular CNNs interpretation [1, 3, 2]. In [1] and [3], the challenges faced by the current diagnostic measurements of the diabetic retinopathy and how the CNN can enhance it, is indicated. CNNs' unique network architecture enables it to make automation of the feature design efficient, the non-linearity transformation of CNN layers' outputs makes the network effective [3].

The CNNs standard requirement is mostly for classification of the whole image (one image belonging to one class), but the current problem requires more than that, every pixel of the image needs to be classified (localization) [2]. [1] and [2] has discussed that the CNNs' initial approach to EFIA, which was essentially patch-based, did deliver to some extent but it was not adequate. It is implemented by assigning a patch around

each pixel for localization. Nevertheless, it had a few disadvantages; network slowness, due to the individual execution of each patch (local region), redundancy, caused by the extending patches, and either localization accuracy or the context usage could be achieved [2].

The fully connected Convolutional Neural Networks architecture, used by [1] and [2], is established to be efficient in EFIA. The architecture converges a base network for the process of blood vessel segmentation [1]. The network is transformed in such a way that it can get the work done only with the small amount of data (a few images) [2], a similar approach is employed for EFIA. The principle approach in [2] is to extend a typical shrinking network by progressive CNN layers and upsampling variables, containing numerous feature channels, are used instead of pooling controllers, which increases output resolution. In [1], Initially, Visual Geometry Group (VGG) network is employed, which is mostly used for large-scale images' classification. The final frames of the network, containing fully connected layers, are removed, leaving the network merely with the CNNs implemented with Rectified Linear Unit (ReLU) activations and four max-pooling layers [1]. The use of max-pooling layers distinguishing multiple serialized frames, containing various Convolutional layers [1]. The class-balancing cross entropy loss function is applied for training the network [1, 2]. However, in [3], rendering of the Regression Activation Maps (RAM) is used, RAM is formed for an input image which confines the distinct regions upon the regression inferences.

## DATASET DESCRIPTION AND FEATURES SELECTION

A publically available eye fundus images' dataset, DRIVE (Digital Retinal Images for Vessel Extraction), is used for Eye Fundus Image Analysis (EFIA) [5]. The dataset's photographs were procured from a screening program of retinopathy in the Netherlands [5]. The diabetic subjects on whom the experiments were conducted, were between twenty-five to ninety years [5]. Around forty images were randomly accumulated, barely seven of

them showed implications of scanty diabetic retinopathy, and thirty-three of them were declared normal [5].

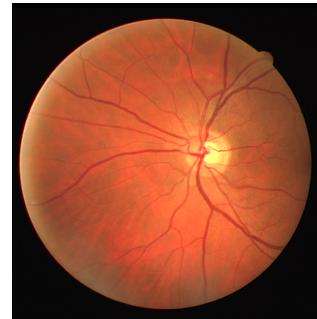


FIGURE 1: EYE FUNDUS IMAGE [5]

The non-mydriatic Canon CR5 3CCD camera, with a 45-degree view field (FOV), was used to obtain the images [5]. 8 bits per color plane at pixels of 768 by 584 was used as the configuration while capturing the images [5]. All the images are spherical, with the 540 pixels' diameter [5]. The segments of the images outside the field of view have been excluded, for the dataset [5]. The masks (spherical area of the images) are included in the dataset to outline the field of view [5].

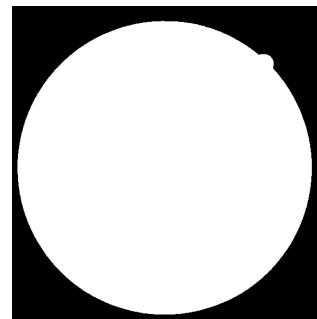


FIGURE 2: IMAGE MASK [5]

The dataset is already divided by [5], into two segments, for the training and testing [5]. Each part consists of 20 images, 20 manual illustrations of the blood vessels (vasculature) in the eye fundus, and 20 masks of each image [5]. The illustrated vasculatures in the testing set were used to compare with the system-generated segmentation of vasculatures [5]. An experienced ophthalmologist trained the human observers who manually performed the segmentation of vasculatures [5]. Many clinical diagnoses were actually conducted using images of the DRIVE dataset [5].

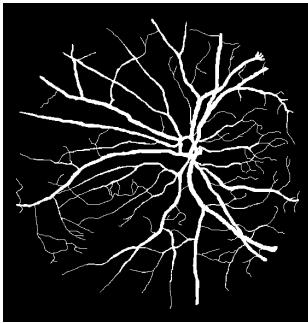


FIGURE 3: HUMAN-MADE SEGMENTATION [5]

The current experiment consists of the feature extraction (patches' acquirement) of the eye fundus images, classification of the acquired patches, and then automatic segmentation of the retinal blood vessels; analyzed on the patches. The purpose of the experiment is to observe if the computer-generated illustrations of the retinal blood vessels are as accurate as the human-made manual segmentation of (vasculature) blood vessels.

## USED METHODS

Convolutional Neural Networks was the appropriate choice for the Eye Fundus Image Analysis (EFIA) since it has been proved that CNN architecture delivers the best results when it comes to image analysis and classification.

ConvNets' network operates exactly as the ANNs'. It is also feed-forward, the information flow is unidirectional, the network takes in the data as the input, process it through the network, and to the output [11]. As the ANNs are inspired by the biological Neural Networks, CNNs also emulates the brain's visual cortex, it comprises variant layers of fluctuating (simple and complex) cells [11].

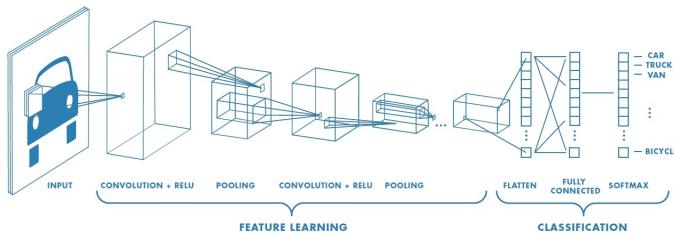


FIGURE 4: CONVNETS ARCHITECTURE [12]

The architecture is considered a feature extractor; thus it learns the representation of the image (input) features [11]. CNNs architecture reduces the computation as it's

feature map's each neuron consist of a receptive field, unlike ANNs, it only connects to the neighboring neurons (a certain group of neurons) of the previous layer through the set of trainable weights, instead of every neuron [11]. Every feature map within the network carries different weights, however, each neuron within every feature map have exactly same weights [11].

$$- \quad Y_k = f(W_k * x)$$

$Y_k$ : feature map of the  $k$ th output

$W_k$ : Convolutional filter of the  $k$ th feature map (pooling and extraction)

$x$ : Input image

$f$ : Non-linear activation function (Rectified Linear Unit in our case)

## EXPERIMENTAL SETUP

### PRE-PROCESSING

The DRIVE dataset's (training) images were pre-processed using the OpenCV methods, prior to the training. OpenCV is an open computer vision library, it comprises useful Computer Vision algorithms. Below are the applied transformations. (*FYI, DRIVE dataset was converted to HDF5 for making it easily accessible, go to Appendix E*)

- *adjustment of Gamma*
- *Standardization*
- *conversion to gray-scale*
- *Contrast-limited adaptive histogram equalization (CLAHE)*

Please observe figure 5 for the histogram equalization of the images and figure 6 to observe the transformed image. (go to Appendix A for the whole code)

```
===== histogram equalization
def histo_equalized(imgs):
    assert (len(imgs.shape)==4) #4D arrays
    assert (imgs.shape[1]==1) #check the channel is 1
    imgs_equalized = np.empty(imgs.shape)
    for i in range(imgs.shape[0]):
        imgs_equalized[i,0] = cv2.equalizeHist(np.array(imgs[i,0], dtype = np.uint8))
    return imgs_equalized
```

FIGURE 5: HISTOGRAM EQUALIZATION

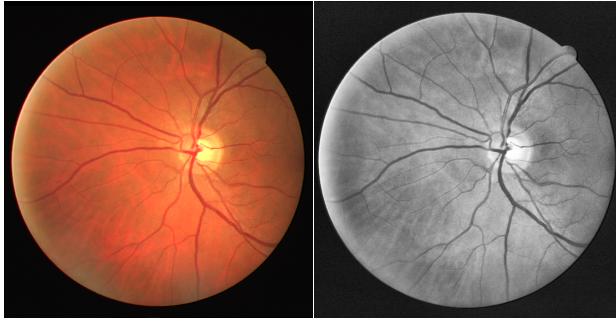


FIGURE 6: TRANSFORMED IMAGE

## FEATURE EXTRACTION

The training was conducted using the extracted patches of images. To show the model the difference between blood vessels and the Field OF View (FOV), patches outside the FOV were selected. Please see *figure 7* for the python code and *figure 8* for observing a sample of extracted patches (image and its segmentation). (go to *Appendix B* for the whole code)

```
#extract patches randomly from the full training images
# -- Inside the full image
def extract_random(full_imgs,full_masks, patch_w,patch_h,N_patches, inside=True):
    if (N_patches*patch_w*patch_h*patch_h*patch_w) == 0:
        print("N_patches: please enter a multiple of 20^4")
        exit()
    assert (len(full_imgs.shape)==4 and len(full_masks.shape)==4) #4D arrays
    assert (full_imgs.shape[1]==1 or full_imgs.shape[1]==3) #check the channel is 1 or 3
    assert (full_masks.shape[1]==1 or full_masks.shape[1]==3) #check the channel is 1 or 3
    assert (full_imgs.shape[2]==patch_w and full_imgs.shape[3]==patch_h)
    assert (full_masks.shape[2]==patch_w and full_masks.shape[3]==patch_h)
    patches = np.empty((N_patches,full_imgs.shape[1],patch_h,patch_w))
    patches_masks = np.empty((N_patches,full_masks.shape[1],patch_h,patch_w))
    img_h = full_imgs.shape[2] #height of the full image
    img_w = full_imgs.shape[3] #width of the full image
    patches_per_img = int(N_patches*patch_w*patch_h*patch_w*patch_h)
    print("patches per full image: "+str(patch_per_img))
    iter_tot = 0 #iter over the total number of patches (N_patches)
    for i in range(full_imgs.shape[0]): #loop over the full images
        k=0
        while k < patch_per_img:
            x_center = random.randint(0,int(patch_w/2),img_w-int(patch_w/2))
            y_center = random.randint(0,int(patch_h/2),img_h-int(patch_h/2))
            # print("x_center "+str(x_center))
            # print("y_center "+str(y_center))
            # print("x_center "+str(y_center))
            #check whether the patch is fully contained in the FOV
            if inside==True:
                if is_patch_inside_FOV(x_center,y_center,img_w,img_h,patch_h)==False:
                    continue
            patch = full_imgs[i,:,y_center-int(patch_w/2):y_center+int(patch_w/2),x_center-int(patch_w/2):x_center+int(patch_w/2)]
            patch_masks = full_masks[i,:,y_center-int(patch_h/2):y_center+int(patch_h/2),x_center-int(patch_w/2):x_center+int(patch_w/2)]
            patches[k] = patch
            patches_masks[k] = patch_masks
            iter_tot += 1
            k+=1 #per full img
    return patches, patches_masks
```

FIGURE 7: EXTRACTING PATCHES

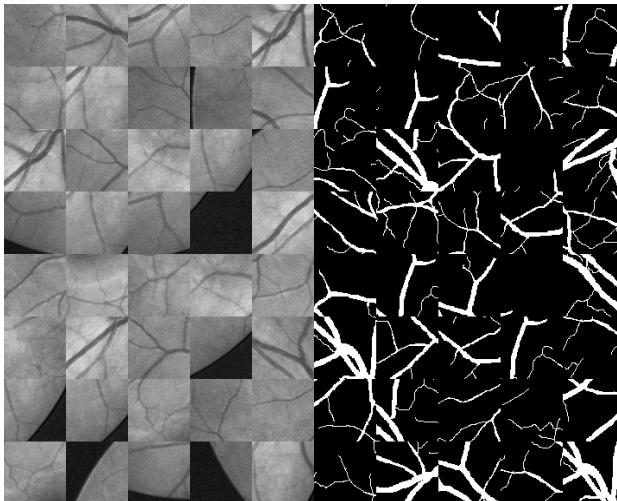


FIGURE 8: IMG AND SEGMENTATION PATCHES

## TRAINING

A set of 3200 patches were considered to be extracted, for our first and second experiment at least, the results of both of them were a disaster, nevertheless, 160 patches of each training images. 90% of the set was used for the training and the rest of it was used for validation.

The cross-entropy was used as a loss function (please see *figure 9*) and the optimizer, which was used in the experiment, is the stochastic gradient descent [1]. Rectifier Linear Unit was employed, as the activation function, after every convolutional layer. 0.2 dropout between the layers was used and the training was conducted for 100 epochs, for our first experiment at least, with the batch (size) of 32 patches.

$$\mathcal{L}(\mathbf{W}) = -\beta \sum_{j \in Y_+} \log P(y_j = 1 | X; \mathbf{W}) - (1 - \beta) \sum_{j \in Y_-} \log P(y_j = 0 | X; \mathbf{W})$$

FIGURE 9 CROSS-ENTROPY (REMOVED SUBSCRIPT  $N$  TO SIMPLIFY) [1]

$W$ : Set of CNN parameters

$\beta$ : Handler of the imbalance between background and foreground pixels' numbers

$Y_+$  &  $Y_-$ : Ground truth's ( $Y$ ) foreground and background sets

$$- \quad \beta = |Y_-| / |Y| \text{ (for the EFIA)}$$

Please go through *figure 10* to observe convolutional neural networks' implementation with *Keras* and go to *Appendix C* to see the whole training code (*Keras* is an open source API for Neural Networks and it is developed in python).

The training in the first experiment exerted around 15 hours to complete, even though it did not produce any usable results (please go to the results section for more details).

```

#define the neural network
def get_unet(n_ch,patch_height,patch_width):
    inputs = Input(shape=(n_ch,patch_height,patch_width))
    conv1 = Conv2D(32, (3, 3), activation='relu', padding='same',data_format='channels_first')(inputs)
    conv1 = Dropout(0.2)(conv1)
    conv1 = Conv2D(32, (3, 3), activation='relu', padding='same',data_format='channels_first')(conv1)
    pool1 = MaxPooling2D((2, 2))(conv1)
    #
    conv2 = Conv2D(64, (3, 3), activation='relu', padding='same',data_format='channels_first')(pool1)
    conv2 = Dropout(0.2)(conv2)
    conv2 = Conv2D(64, (3, 3), activation='relu', padding='same',data_format='channels_first')(conv2)
    pool2 = MaxPooling2D((2, 2))(conv2)
    #
    conv3 = Conv2D(128, (3, 3), activation='relu', padding='same',data_format='channels_first')(pool2)
    conv3 = Dropout(0.2)(conv3)
    conv3 = Conv2D(128, (3, 3), activation='relu', padding='same',data_format='channels_first')(conv3)

    up1 = UpSampling2D(size=(2, 2))(conv3)
    up1 = concatenate([conv2,up1],axis=1)
    conv4 = Conv2D(64, (3, 3), activation='relu', padding='same',data_format='channels_first')(up1)
    conv4 = Dropout(0.2)(conv4)
    conv4 = Conv2D(64, (3, 3), activation='relu', padding='same',data_format='channels_first')(conv4)
    #
    up2 = UpSampling2D(size=(2, 2))(conv4)
    up2 = concatenate([conv1,up2], axis=1)
    conv5 = Conv2D(32, (3, 3), activation='relu', padding='same',data_format='channels_first')(up2)
    conv5 = Dropout(0.2)(conv5)
    conv5 = Conv2D(32, (3, 3), activation='relu', padding='same',data_format='channels_first')(conv5)
    #
    conv6 = Conv2D(2, (1, 1), activation='relu',padding='same',data_format='channels_first')(conv5)
    conv6 = core.Reshape((2,patch_height*patch_width))(conv6)
    conv6 = core.Permute((2,1))(conv6)
    #####
    conv7 = core.Activation('softmax')(conv6)

    model = Model(input=inputs, output=conv7)

    # sgd = SGD(lr=0.01, decay=1e-6, momentum=0.3, nesterov=False)
    model.compile(optimizer='sgd', loss='categorical_crossentropy',metrics=['accuracy'])

    return model

```

FIGURE 10: CONVOLUTIONAL NEURAL NETWORKS IMPLEMENTATION

## TESTING

The testing was conducted on the provided 20 test images in the DRIVE dataset. Used the provided manual blood vessels' segmentation to compare with the machine-generated segmentation. (please see *figure 11* for the testing implementation and go to *Appendix D* to see what else is happening in the testing file)

```

===== Run the prediction of the patches =====
best_last = config.get('testing settings', 'best_last')
#Load the saved model
model = model_from_json(open(path_experiment+name_experiment + '_architecture.json').read())
model.load_weights(path_experiment+name_experiment + '_'+best_last+'_weights.h5')
#Calculate the predictions
predictions = model.predict(patches_imgs_test, batch_size=32, verbose=2)
print("predicted images size :")
print(predictions.shape)

===== Convert the prediction arrays in corresponding images
pred_patches = pred_to_imgs(predictions, patch_height, patch_width, "original")

```

FIGURE 11: TESTING IMPLEMENTATION

In order to identify the field of view, just so the model would only consider the pixels within it, used the provided image masks. In order to advance the performance, averaged the predictions of each pixel to procure the probability of it being a blood vessel. By using the stride of certain pixels, in width and height, extracted serialized overlapping patches from all the test images.

Onwards, by averaging probabilities through every predicted patch incorporating the pixel, acquired the blood vessel probability for all the pixels.

## RESULTS

### FIRST EXPERIMENT

Experiment 1 ran for almost 15 hours, even though the patches' size (*N\_subimgs*) was reduced to 3200 (160 patches from each image), but it did not produce any results, however, it did create a nohup log file and a file containing the successful weights (visit [https://github.com/Huzmorgoth/Aritificial-Neural-Networks/blob/master/ANNCourseWorkExperimentResult/ANNCourseWorkExperimentResult\\_training.nohup](https://github.com/Huzmorgoth/Aritificial-Neural-Networks/blob/master/ANNCourseWorkExperimentResult/ANNCourseWorkExperimentResult_training.nohup) for the produced results of experiment 1). Below mentioned image shows the configuration for the first experiment.

```

[data paths]
path_local = ./DRIVE_datasets_training_testing/
train_imgs_original = DRIVE_dataset_imgs_train.hdf5
train_groundTruth = DRIVE_dataset_groundTruth_train.hdf5
train_border_masks = DRIVE_dataset_borderMasks_train.hdf5
test_imgs_original = DRIVE_dataset_imgs_test.hdf5
test_groundTruth = DRIVE_dataset_groundTruth_test.hdf5
test_border_masks = DRIVE_dataset_borderMasks_test.hdf5

[experiment name]
name = ANNCourseWorkExperimentResult

[data attributes]
#dimensions of the patches extracted from the full images
patch_height = 48
patch_width = 48

[training settings]
#number of total patches:
N_subimgs = 3200
#if patches are extracted only inside the field of view:
inside_FOV = False
#Number of training epochs
N_epochs = 100
batch_size = 32
#if running with nohup
nohup = True

[testing settings]
#choose the model to test: best==epoch with min loss, last==last epoch
best_last = best
#number of images for the test (max 20)
full_images_to_test = 20
#How many original-groundTruth-prediction images are visualized in each image
N_group_visual = 1
#Compute average in the prediction, improve results but require more patches to be predicted
average_mode = True
#Only if average_mode==True. Stride for patch extraction, lower value require more patches to be predicted
stride_height = 46
stride_width = 46
#if running with nohup
nohup = True

```

FIGURE 12: STRUCTURE FILE 1

### SECOND EXPERIMENT

The second experiment ran for 7 hours. The epochs were set to 20, the images (of training and testing) were reduced to 50% of the original set, and the stride\_height and stride\_width were set to 46 (while testing) to reduce the running time of the training and testing.

Please observe the results mentioned below. As you can see, the results are produced but they are quite bad.

```
[data paths]
path_local = '/NEWDRIVE/datasets/training_testing/'
train_imgs_original = NEWDRIVE_dataset_imgs_train.hdf5
train_groundTruth = NEWDRIVE_dataset_gtTruth_train.hdf5
train_border_masks = NEWDRIVE_dataset_borderMasks_train.hdf5
test_imgs = NEWDRIVE_dataset_imgs_test.hdf5
test_groundTruth = NEWDRIVE_dataset_groundTruth_test.hdf5
test_border_masks = NEWDRIVE_dataset_borderMasks_test.hdf5

[experiment name]
name = ANNcourseWorkExperimentResult

[data attributes]
#dimensions of the patches extracted from the full images
patch_height = 48
patch_width = 48

[training settings]
#number of total patches:
N_patches = 30000
##if patches are extracted only inside the field of view:
inside_FOV = False
#Number of training epochs
N_epochs = 20
batch_size = 32
##if running with nohup
nohup = True

[testing settings]
#Choose the model to test: best==epoch with min loss, last==last epoch
best_last = best
#number of full images for the test (max 10)
full_images_to_test = 10
##how many original-groundTruth-prediction images are visualized in each image
N_visualizations = 3
##Compute average in the prediction, improve results but require more patches to be predicted
average_mode = True
##Only if average_mode==True. Stride for patch extraction, lower value require more patches to be predicted
stride_height = 46
stride_width = 46
##if running with nohup
nohup = True
```

FIGURE 13: STRUCTURE FILE 2

```
Area under the ROC curve: 0.7024789612104095
Area under Precision-Recall curve: 0.29859295758516197
Jaccard similarity score: 0.8718904135185923
F1 score (F-measure): 0.0
```

```
Confusion matrix: [[1978417      0]
 [ 290695      0]]
ACCURACY: 0.8718904135185923
SENSITIVITY: 0.0
SPECIFICITY: 1.0
PRECISION: 0
```

FIGURE 14: PERFORMANCE OF THE MODEL

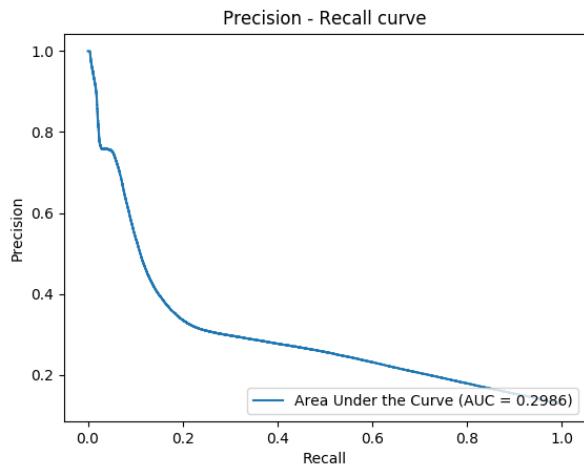


FIGURE 15: PRECISION RECALL

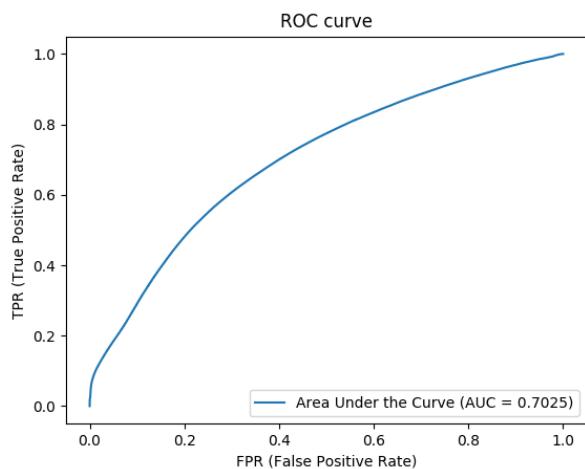


FIGURE 16: ROC CURVE

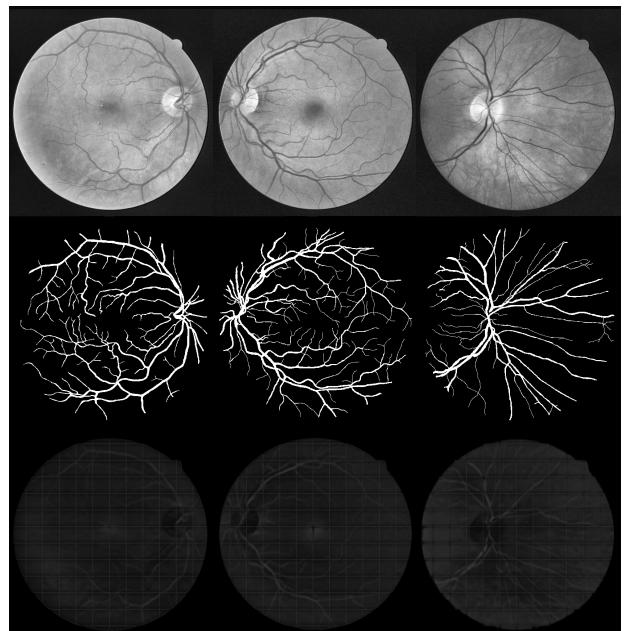


FIGURE 17: PREDICTED SEGMENTATION

Above are 3 of the system-generated blood vessel segmentations. Originals are in the first row, manual segmentations are in the second row, and the system-generated segmentations are in the third row.

## FINAL EXPERIMENT

The final experiment ran for 20 hours (please see figure 18 for configuration of the third experiment). Please observe the results mentioned below.

As you can see, the produced results of the final experiment are better than previous experiments'.

```
[data paths]
path_local = '/DRIVE/datasets_training_testing/
train_imgs_and_gt = DRIVE_dataset_imgs_train.hdf5
train_gt = DRIVE_dataset_groundTruth_train.hdf5
train_border_masks = DRIVE_dataset_borderMasks_train.hdf5
test_imgs = DRIVE_dataset_imgs_test.hdf5
test_gt = DRIVE_dataset_groundTruth_test.hdf5
test_border_masks = DRIVE_dataset_borderMasks_test.hdf5

[experiment name]
name = ANN_NEW_Results

[data attributes]
#Dimensions of the patches extracted from the full images
patch_height = 48
patch_width = 48

[training settings]
#Number of total patches:
N_subings = 190000
#if patches are extracted only inside the field of view:
inside_FOV = False
#Number of training epochs
N_epochs = 150
batch_size = 32
#if running with nohup
nohup = False

[testing settings]
#Choose the model to test: best==epoch with min loss, last==last epoch
best_last = best
#Number of full images for the test (max 20)
full_images_to_use = 20
#How many original-groundTruth-prediction images are visualized in each image
N_group_visual = 1
#Compute average in the prediction, improve results but require more patches to be predicted
average_mode = True
#Only if average mode=True. Stride for patch extraction, lower value require more patches to be predicted
stride_height = 5
stride_width = 5
#if running with nohup
nohup = False
```

FIGURE 18: STRUCTURE FILE 3

```
Area under the ROC curve: 0.97904342123
Area under Precision-Recall curve: 0.91033534256
Jaccard similarity score: 0.955927459326
F1 score (F-measure): 0.816022456728
```

```
Confusion matrix: [[3892134  65128]
 [ 134311  443119]]
ACCURACY: 0.955983977246
SENSITIVITY: 0.767137103303
SPECIFICITY: 0.98352749132
PRECISION: 0.871673209431
```

FIGURE 19: FINAL EXPERIMENT'S PERFORMANCE

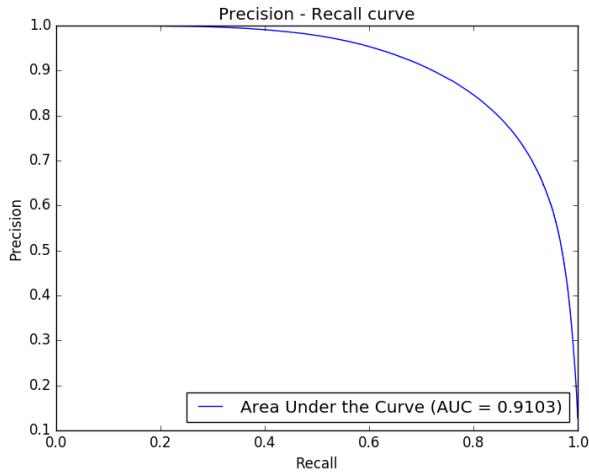


FIGURE 20: FINAL EXPERIMENT'S PRECISION RECALL CURVE

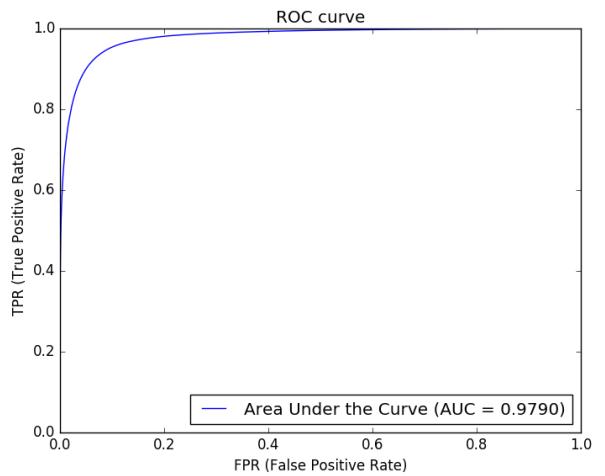


FIGURE 21: FINAL EXPERIMENT'S ROC CURVE

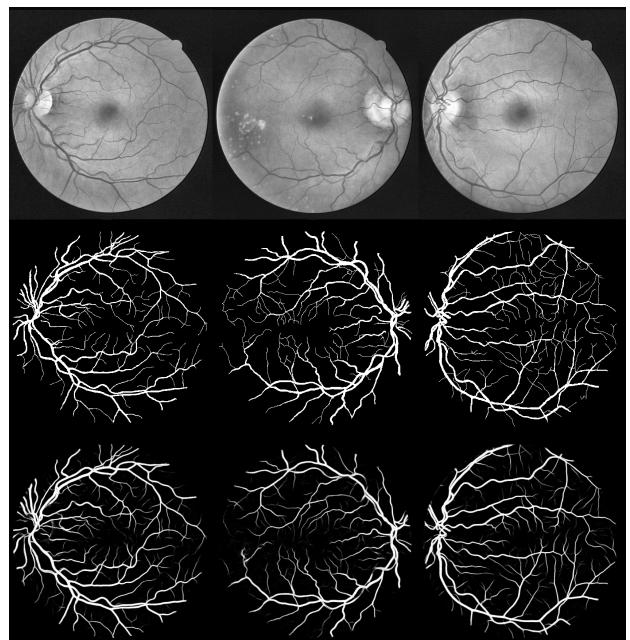


FIGURE 22: FINAL EXPERIMENT'S SYSTEM-GENERATED SEGMENTATION

Above are 3 of the system-generated blood vessel segmentations.

Originals are in the first row, manual segmentations are in the second row, and the system-generated segmentations are in the third row.

## DISCUSSION AND CONCLUSION

In this paper, the Eye Fundus Image Analysis (EFIA) is introduced, an automated approach to the retinal blood vessel segmentation. With the employment of ConvNets, one of the most striking image classification method of Artificial Neural Networks, and its u-net architecture, EFIA is enhanced in such a way that it produces results as good as a human annotator, even better in some situations.

Publically available, DRIVE dataset is determined to be an effective source, it rendered sufficient information for the system to develop an intelligent model.

The experiment merely devoted 20 hours, on the laptop of 1.3 GHz Intel Core i5, to be completed with an outstanding success, which is a lot better than the time a human observer takes to illustrate retinal blood vessel vasculature. The automated medical diagnoses can bring evolution to the medical fields with an immeasurable novelty.

## REFERENCES

- [1] K. Maninis, J. Pont-Tuset, P. Andrés Arbeláez and L. Van Gool, *Deep Retinal Image Understanding*. Ios Andes: MICCAI, 2016.
- [2] O. Ronneberger, P. Fischer and T. Brox, *U-Net: Convolutional Networks for Biomedical Image Segmentation*. Freiburg: Computer Science Department and BIOSS Centre for Biological Signalling Studies, 2015.
- [3] Z. Wang and J. Yang, *Diabetic Retinopathy Detection via Deep Convolutional Networks for Discriminative Localization and Visual Explanation*. San Ramon: GE Global Research, 2017.
- [4] S. Mallick and V. Gupta, "Learn OpenCV ( C++ / Python )", *Learnopencv.com*. [Online]. Available: <https://www.learnopencv.com>. [Accessed: 09- Mar- 2018].
- [5] "DRIVE: Digital Retinal Images for Vessel Extraction", *Isi.uu.nl*. [Online]. Available: <https://www.isi.uu.nl/Research/Databases/DRIVE/>. [Accessed: 09- Mar- 2018].
- [6] "Welcome to OpenCV-Python Tutorials' documentation! — OpenCV-Python Tutorials 1 documentation", *Opencv-python-tutorial.readthedocs.io*, 2018. [Online]. Available: <https://opencv-python-tutorial.readthedocs.io/en/latest/>. [Accessed: 18- Mar- 2018].
- [7] "Convolutional Neural Networks | TensorFlow", *TensorFlow*. [Online]. Available: [https://www.tensorflow.org/tutorials/deep\\_cnn](https://www.tensorflow.org/tutorials/deep_cnn). [Accessed: 24- Mar- 2018].
- [8] "Writing your own Keras layers - Keras Documentation", *Keras.io*. [Online]. Available: <https://keras.io/layers/writing-your-own-keras-layers/>. [Accessed: 28- Mar- 2018].
- [9] "Introduction to HDF5", *Support.hdfgroup.org*. [Online]. Available: <https://support.hdfgroup.org/HDF5/doc/H5.intro.html>. [Accessed: 28- Mar- 2018].
- [10] "API Reference — scikit-learn 0.19.1 documentation", *Scikit-learn.org*. [Online]. Available: <http://scikit-learn.org/stable/modules/classes.html>. [Accessed: 28- Mar- 2018].
- [11] W. Rawat and Z. Wang, *Deep Convolutional Neural Networks for Image Classification: A Comprehensive Review*. Florida: Department of Electrical and Mining Engineering, University of South Africa.
- [12] "Convolutional Neural Network", *Uk.mathworks.com*. [Online]. Available: <https://uk.mathworks.com/discovery/convolutional-neural-network.html>. [Accessed: 04- Apr- 2018].
- [13] "Learning Less to Learn Better — Dropout in (Deep) Machine learning", *Medium*. [Online]. Available: <https://medium.com/@amarbudhiraja/https-medium-com-amarbudhiraja-learning-less-to-learn-better-dropout-in-deep-machine-learning-74334da4bfc5>. [Accessed: 31- Mar- 2018].

## LIST OF FIGURES

FIGURE 1: EYE FUNDUS IMAGE [5].....	2	FIGURE 24: HELP FUNCTION FILE 2.....	10
FIGURE 2: IMAGE MASK [5] .....	2	FIGURE 25: PRE-PROCESSING 1 .....	10
FIGURE 3: HUMAN-MADE SEGMENTATION [5].....	3	FIGURE 26: PRE-PROCESSING 2 .....	10
FIGURE 4: CONVNETS ARCHITECTURE [12] .....	3	FIGURE 27: PATCH-EXTRACTION 1.....	10
FIGURE 5: HISTOGRAM EQUALIZATION .....	3	FIGURE 28: PATCH-EXTRACTION 2.....	10
FIGURE 6: TRANSFORMED IMAGE .....	4	FIGURE 29: PATCH-EXTRACTION 3.....	10
FIGURE 7: EXTRACTING PATCHES .....	4	FIGURE 30: PATCH-EXTRACTION 4.....	11
FIGURE 8: IMG AND SEGMENTATION PATCHES.....	4	FIGURE 31: PATCH-EXTRACTION 5.....	11
FIGURE 9 CROSS-ENTROPY (REMOVED SUBSCRIPT $n$ TO SIMPLIFY) [1] ....	4	FIGURE 32: PATCH-EXTRACTION 6.....	11
FIGURE 10: CONVOLUTIONAL NEURAL NETWORKS IMPLEMENTATION .....	5	FIGURE 33: PATCH-EXTRACTION 7.....	11
FIGURE 11:TESTING IMPLEMENTATION.....	5	FIGURE 34: PATCH-EXTRACTION 8.....	11
FIGURE 12: STRUCTURE FILE 1 .....	5	FIGURE 35: TRAINING 1 .....	11
FIGURE 13: STRUCTURE FILE 2 .....	6	FIGURE 36: TRAINING 2 .....	11
FIGURE 14: PERFORMANCE OF THE MODEL .....	6	FIGURE 37: TRAINING 3 .....	12
FIGURE 15: PRECISION RECALL .....	6	FIGURE 38: TRAINING 4 .....	12
FIGURE 16: ROC CURVE.....	6	FIGURE 39: TRAINING 5 .....	12
FIGURE 17: PRDECTED SEGMENTATON .....	6	FIGURE 40: TEST 1.....	12
FIGURE 18: STRUCTURE FILE 3 .....	7	FIGURE 41: TEST 2.....	12
FIGURE 19: FINAL EXPERIMENT'S PERFORMANCE.....	7	FIGURE 42: TEST 3.....	12
FIGURE 20: FINAL EXPERIMENT'S PRECISION RECALL CURVE .....	7	FIGURE 43: TEST 4.....	12
FIGURE 21: FINAL EXPERIMENT'S ROC CURVE .....	7	FIGURE 44: TEST 5.....	12
FIGURE 22: FINAL EXPERIMENT'S SYSTEM-GENERATED SEGMENTATION... 7		FIGURE 45: CONVERSION TO HDF5-1 .....	13
FIGURE 23: HELP FUNCTION FILE 1.....	10	FIGURE 46: CONVERSION TO HDF5-2.....	13

## APPENDICES

## APPENDIX A

```

1 import h5py
2 import numpy as np
3 from PIL import Image
4 from matplotlib import pyplot as plt
5
6 def load_hdf5(infile):
7     with h5py.File(infile,"r") as f: # "with" close the file after its nested commands
8         return f["image"][:]
9
10 def write_hdf5(arr,outfile):
11     with h5py.File(outfile,"w") as f:
12         f.create_dataset("image", data=arr, dtype=arr.dtype)
13
14 #convert RGB image in black and white
15 def rgb2gray(rgb):
16     assert (len(rgb.shape)==4) #4D arrays
17     assert (rgb.shape[3]==3)
18     bn_imgs = rgb[:,0,:,:]*.299 + rgb[:,1,:,:]*.587 + rgb[:,2,:,:]*.114
19     bn_imgs = np.reshape(bn_imgs,(rgb.shape[0],1,rgb.shape[2],rgb.shape[3]))
20     return bn_imgs
21
22 #group a set of Images row per columns
23 def group_images(data_per_row):
24     assert (len(data_per_row)==2)
25     assert (data_per_row[0].shape[0]==data_per_row[1].shape[0])
26     assert ((data_per_row[0].shape[1]==1) or (data_per_row[1].shape[1]==3))
27     data = np.transpose(data,(0,2,0,1)) #correct format for imshow
28     all_stripe = []
29     for i in range(int(data.shape[0]/per_row)):
30         stripe = data[i*per_row]
31         for k in range(i*per_row+1, i*per_row+per_row):
32             stripe = np.concatenate((stripe,data[k]),axis=1)
33         all_stripe.append(stripe)
34     totimg = all_stripe[0]
35     for i in range(1,int(all_stripe)):
36         totimg = np.concatenate((totimg,all_stripe[i]),axis=0)
37     return totimg
38
39
40 #visualize image (as PIL image, NOT as matplotlib!)
41 def visualize(data,filename):
42     assert (len(data.shape)==3) #height*width*channels
43     img = None
44     if data.shape[2]==1: #in case it is black and white
45         img = np.reshape(data,(data.shape[0],data.shape[1]))
46         img = np.max(img)*np.ones((img.shape[0],img.shape[1]))
47     else:
48         img = Image.fromarray((data*255).astype(np.uint8)) #the image is already 0-255
49     img.save(filename+'.png')
50     return img
51
52
53

```

**FIGURE 23: HELP FUNCTION FILE 1**

```

51
52 #prepare the mask in the right shape for the Unet
53 def masks_2d(masks):
54     assert (len(masks.shape)==4) #4D arrays
55     assert (masks.shape[1]==1) #check the channel is 1
56     im_h = masks.shape[2]
57     im_w = masks.shape[3]
58
59     masks_2d = np.reshape(masks,(masks.shape[0],im_h+im_w))
60     new_masks = np.empty((masks.shape[0],im_h+im_w,2))
61
62     for i in range(masks_2d.shape[0]):
63         for j in range(im_h+im_w):
64             if masks_2d[i,j] == 0:
65                 new_masks[i,j,0]=1
66                 new_masks[i,j,1]=0
67             else:
68                 new_masks[i,j,0]=0
69                 new_masks[i,j,1]=1
70
71     return new_masks
72
73
74 def pred_to_imgs(pred,patch_height,patch_width,mode="original1"):
75     assert (len(pred.shape)==3) #3D array: (Npatches,height*width,2)
76     assert (pred.shape[2]==2) #check the classes are 2
77     pred_images = np.empty((pred.shape[0],pred.shape[1])) #Npatches,height*width)
78     if mode=="original1":
79         for i in range(pred.shape[0]):
80             for pix in range(pred.shape[1]):
81                 pred_images[i,pix]=pred[i,pix,i]
82
83     elif mode=="threshold":
84         for i in range(pred.shape[0]):
85             for pix in range(pred.shape[1]):
86                 if pred[i,pix]>=0.5:
87                     pred_images[i,pix]=1
88                 else:
89                     pred_images[i,pix]=0
90
91     else:
92         print("mode "+str(mode)+" not recognized, it can be 'original' or 'threshold'")
93         exit()
94
95 pred_images = np.reshape(pred_images,(pred_images.shape[0],1,patch_height,patch_width))
96 return pred_images

```

**FIGURE 24: HELP FUNCTION FILE 2**

```

1 #####-----#####
2 #   pre-processing the original images
3 #   -----
4 #####
5 #####
6 #
7
8 import numpy as np
9 from PIL import Image
10 import cv2
11
12 from Help_functions import *
13
14
15 #PRE-processing (used for both training and testing)
16 def my_PrepProcess():
17     dataset = datasets.ImageFolder()
18     assert (dataset.data.shape[1]==3) #Use the original images
19     blackwhite_conversion
20     train_imgs = np.array(dataset.data)
21     assert (train_imgs.shape[1]==3)
22     train_imgs = dataset_normalized(train_imgs)
23     train_imgs = channel_equalized(train_imgs)
24     train_imgs = train_imgs/255. #reduce to 0-1 range
25     return train_imgs
26
27
28 #####
29 #####-----#####
30 #####-----#####
31 #####-----#####
32 #####-----#####
33 ##### histogram equalization
34 def histo_equalized(img):
35     imgs = np.array([img]) #list of images
36     assert (imgs.shape[1]==3) #4D arrays
37     assert (imgs.shape[1]==1) #check the channel is 1
38     imgs_equalized = np.empty(imgs.shape)
39     for i in range(imgs.shape[0]):
40         img = imgs[i]
41         img_equalized = cv2.equalizeHist(np.asarray(imgs[i], dtype = np.uint8))
42         imgs_equalized[i] = img_equalized
43
44     return imgs_equalized
45
46

```

**FIGURE 25: PRE-PROCESSING 1**

```

# CLAHE (Contrast Limited Adaptive Histogram Equalization)
# Adaptive histogram equalization is used. In this, image is divided into small blocks called "tiles" ( tileSize=8x8 by default in OpenCV). Then each of
# a small area, histogram would confine to a small region (unless there is noise). If noise is there, it will be amplified. To avoid this, contrast limit
# specific contrast limit (by default is 40 in OpenCV), those pixels are clipped and distributed uniformly to other bins before applying histogram equalization
# This limits the dynamic range of the output.
def clahe_equalized(imgs):
    assert (len(imgs.shape)==4) #4D arrays
    assert (imgs.shape[1]==3) #check the channel is 3
    #create a CLAHE object (Arguments are optional).
    clahe = cv2.createCLAHE(clipLimit=4.0, tileGridSize=(8,8))
    imgs_equalized = np.empty(imgs.shape)
    for i in range(imgs.shape[0]):
        for j in range(imgs.shape[1]):
            imgs_equalized[i,j] = clahe.apply(np.array(imgs[i,j], dtype = np.uint8))
    return imgs_equalized

# ===== normalize over the dataset
def dataset_normalizing(imgs):
    assert (len(imgs.shape)==4) #4D arrays
    assert (imgs.shape[1]==3) #check the channel is 3
    imgs_normalized = np.empty(imgs.shape)
    imgs_std = np.std(imgs)
    img_mean = np.mean(imgs)
    imgs_normalized = ((imgs-imgs_mean)/(img_mean*imgs_std))
    for i in range(imgs.shape[0]):
        for j in range(imgs.shape[1]):
            imgs_normalized[i,j] = ((imgs_normalized[i,j] - np.min(imgs_normalized[i])) / (np.max(imgs_normalized[i])-np.min(imgs_normalized[i])))**255
    return imgs_normalized

# ===== adjust_gamma
def adjust_gamma(imgs, gamma=0):
    assert (len(imgs.shape)==4) #4D arrays
    assert (imgs.shape[1]==3) #check the channel is 3
    # build a lookup table mapping the pixel values [0, 255] to
    # their adjusted gamma values
    table = np.array([(i ** (1.0 / gamma)) * 255 for i in np.arange(0, 256)]).astype("uint8")
    # apply gamma correction using the lookup table
    new_imgs = np.empty(imgs.shape)
    for i in range(imgs.shape[0]):
        for j in range(imgs.shape[1]):
            new_imgs[i,j] = cv2.LUT(np.array(imgs[i,j], dtype = np.uint8), table)
    return new_imgs

```

**FIGURE 26: PRE-PROCESSING 2**

## APPENDIX E

```

1 import numpy as np
2
3 import configargparse
4
5 from Help_Functions import load_yaml
6 from Help_Functions import visualize
7 from Help_Functions import import_group_images
8
9 from pre_processing import my_PrepPrc
10
11
12 # Select the same images
13 # random.sample()
14
15 # Load the original data and return the extracted patches for training/testing
16 def get_Data_Training(img, train_imgs_original,
17                      patch_size, height, width,
18                      patch_height, patch_width,
19                      N_patches_per_image, N_patches_wanted):
20
21     Train_Img = np.zeros((N_patches_wanted,patch_size,patch_size,3))
22
23     Train_Mask = np.zeros((N_patches_wanted,patch_size,patch_size,1))
24
25     train_imgs = np.zeros((N_patches_wanted,patch_size,patch_size,3))
26
27     train_masks = np.zeros((N_patches_wanted,patch_size,patch_size,1))
28
29     train_imgs = my_PrepPrc(img, train_imgs_original)
30     train_masks = mask_generator(img, train_imgs_original)
31
32     Train_Img[0:N_patches_wanted,:,:,:] = train_imgs[0:N_patches_wanted,:,:,:]
33     Train_Mask[0:N_patches_wanted,:,:,:] = train_masks[0:N_patches_wanted,:,:,:]
34
35     train_imgs = Train_Img[1:N_patches_wanted,:,:,:]
36     train_masks = Train_Mask[1:N_patches_wanted,:,:,:]
37
38     train_imgs = train_imgs[:,train_imgs.shape[1]//2,:,:]
39     train_masks = train_masks[:,train_masks.shape[1]//2,:,:]
40
41     print("Extracting the TRAINING patches from the full images")
42     patches_imgs_train, patches_masks_train = extract_randomTrain_imgs_train_imgs(patch_size,patch_height,patch_width,N_sub_data,N_patches_wanted,patch_size,patch_height,patch_width)
43
44     print("Extracting the TESTING patches from the full images")
45     patches_imgs_test, patches_masks_test = extract_randomTest_imgs_test_imgs(patch_size,patch_height,patch_width,N_sub_data,N_patches_wanted,patch_size,patch_height,patch_width)
46
47     print("patches_imgs TRAIN shape : ", patches_imgs_train.shape)
48     print("patches_masks TRAIN shape : ", patches_masks_train.shape)
49
50     print("patches_imgs TEST shape : ", patches_imgs_test.shape)
51     print("patches_masks TEST shape : ", patches_masks_test.shape)
52
53     print("Extracting the Patches Images range [0,1] : ", " -> " + str(patches_imgs_train.shape) + " -> " + str(patches_masks_train.shape))
54
55     print("Extracting the Patches Images range [0,1] : ", " -> " + str(patches_imgs_test.shape) + " -> " + str(patches_masks_test.shape))
56
57     print("patches_imgs TRAIN shape : ", patches_imgs_train.shape)
58     print("patches_masks TRAIN shape : ", patches_masks_train.shape)
59
60     print("patches_imgs TEST shape : ", patches_imgs_test.shape)
61     print("patches_masks TEST shape : ", patches_masks_test.shape)
62
63     return patches_imgs_train, patches_masks_train, patches_imgs_test, patches_masks_test

```

**FIGURE 27: PATCH-EXTRACTION 1**

```

# load the original image and its mask for testing
img = Image.open('img/original/DRIVE_tvtest_Img_tvtest.png')
mask = Image.open('img/original/DRIVE_tvtest_Mask_tvtest.png')

# test
test_imgs = load_MRFNet(DRIVE_tvtest_Img_tvtest, patch_width, patch_width)
test_masks = load_MRFNet(DRIVE_tvtest_Mask_tvtest)

# test_imgs = my_ProfTest(imgs_original)
# test_masks = my_ProfTest(mask_original)

# assertions
assert len(test_imgs) == 1000, "The number of images to test must be divided exactly by the patches dimensions"
test_imgs = test_imgs[0]
test_masks = test_masks[0]
assert len(test_imgs) == len(test_masks), "The number of images to test must be divided exactly by the patches dimensions"
test_imgs = np.array(test_imgs)
test_masks = np.array(test_masks)
assert test_imgs.shape == test_masks.shape, "The images and masks must have the same shape"
assert np.max(test_imgs) <= 1.0, "The images must be between 0 and 1"
assert np.min(test_imgs) >= 0.0, "The images must be between 0 and 1"
assert np.max(test_masks) <= 1.0, "The masks must be between 0 and 1"
assert np.min(test_masks) >= 0.0, "The masks must be between 0 and 1"

# print metrics
print("Mean images shape : ", np.mean(test_imgs.shape))
print("Mean mask shape : ", np.mean(test_masks.shape))

# print metrics
print("Mean images range from the full range")
print("maxes : ", np.max(test_imgs))
print("mines : ", np.min(test_imgs))
print("patch_size : ", patch_width)
print("patches : ", test_imgs.shape[0] * extract_centered_patches(test_imgs, patch_width, patch_width))
print("patches : ", test_masks.shape[0] * extract_centered_patches(test_masks, patch_width, patch_width))

```

```
shape:"}
```

FIGURE 28: PATCH-EXTRACTION 2

5.2. PATCH EXTRACTION 3

FIGURE 30: PATCH-EXTRACTION 4

**FIGURE 31: PATCH-EXTRACTION 5**

**FIGURE 32: PATCH-EXTRACTION 6**

**FIGURE 33: PATCH-EXTRACTION 7**

**FIGURE 34: PATCH-EXTRACTION 8**

APPENDIX C

```

1 #!/usr/bin/python
2
3 # Script to:
4 # - load the images and extract the patches
5 # - build the neural network
6 # - define the training
7
8
9 import numpy as np
10 import configparser
11
12 from keras.models import Model
13 from keras.layers import Input, Conv2D, MaxPooling2D, UpSampling2D, Reshape, Dense
14 from keras.optimizers import Adam
15 from keras.callbacks import ModelCheckpoint, LearningRateScheduler
16 from keras import backend as K
17 from keras.utils import plot_model
18 from keras.preprocessing import image
19
20 import sys
21 sys.path.append("./src")
22
23 from Data_Functions import *
24 from Data_Functions import *
25
26 #Function to obtain data for training/testing (validation)
27 from extract_patches import get_data_training

```

**FIGURE 35: TRAINING 1**

### FIGURE 36: TRAINING

**FIGURE 37: TRAINING 3**

**FIGURE 38: TRAINING 4**

```

178
179 #oooooooooooooo Tracing oooooooooooooo
180 checkpoint = ModelCheckpoint(filepath,'--name_experiment=/' + name_experiment + '_best_weights.h5', verbose=1, monitor='val_loss', mode='auto', save_best_only=True) #save at each epoch if the validation
181          #loss is minimum
182
183         # d def step_decay(epoch):
184         #     lr = 0.01 * 0.45**epoch # initial learning rate (by default in keras)
185         #     # print("Epoch %d Learning Rate %f" % (epoch, lr))
186         #     # update learning rate
187         #     return lr*0.005
188         # else:
189         #     # update learning rate
190         #     return lr*0.005
191
192         # lr_step = LearningRateScheduler(step_decay)
193
194 pathos_mp3_train = mp3s_(pathos_mp3, mask, train) #reduce memory consumption
195 model.fit(pathos_mp3_train, pathos_mp3_train, nb_epochs=nb_epochs, batch_size=batch_size, verbose=1, validation_split=validation_split, callbacks=[checkpoint]);
196
197 #oooooooooooooo Done and test the last model oooooooooooooo
198 model.load_weights('' + name_experiment + '/last_weights.h5', overwrite=True)
199
200 #print("Model loaded")
201 #evaluateModel(pathos_mp3,mp3_test, mask, pathos_mp3, mask, test, verbose=0)
202
203 # a print("Test accuracy"), accuracy[0]
204 # a print("Test accuracy"), accuracy[1]

```

FIGURE 39: TRAINING 5

APPENDIX D

**FIGURE 40: TEST 1**

**FIGURE 41: TEST 2**

```

best_lat_>= self.config['best_lat']
        if best_lat < 0:
            raise ValueError("The best latency is negative, please check the patches")
        if not os.path.exists(self.config['output_dir']):
            os.makedirs(self.config['output_dir'])

        # Load the model
        print("Load the model from %s" % self.config['model'])
        model = Model()
        model.load_weights(os.path.join(self.config['output_dir'], 'experiment_%s.h5' % experiment))
        model._name = 'model'
        model._weights = 'weights'

        predictions = model.predict(imgs_dataloader, img_size=1024, batch_size=1024, verbose=0)

        print(predictions.shape)

        print("-----")
        print("Convert the prediction arrays in corresponding images")
        pred_d_array = pred_to_impressions(predictions, patch_height, patch_width, "original")

        print("-----")
        print("Load the images and visualize the predicted images")
        org_d_array = None
        org_l_array = None

        org_d_name = None
        org_l_name = None

        if average_node == True:
            org_d_name = 'average_overlap_grouped'
            org_l_name = 'average_grouped'
            new_width, stride_height, stride_width = 1024, 1, 1
        else:
            org_d_name = 'grouped'
            org_l_name = 'grouped'
            new_width, stride_height, stride_width = 1024, 1, 1

        org_d_array = np.zeros((len(predictions), new_width, stride_height, stride_width)) * predictions
        org_l_array = np.zeros((len(predictions), new_width, stride_height, stride_width)) * predictions

        for i in range(len(predictions)):
            org_d_array[i] = predictions[i]
            org_l_array[i] = predictions[i]

        print("-----")
        print("Reconstruct the images")
        org_d_array = reconstruct_images(org_d_array, 1024, 1)
        org_l_array = reconstruct_images(org_l_array, 1024, 1)

        print("-----")
        print("Get the DNN's mask on the predictions set except everything outside the FOV to zero!")
        org_d_array = get_dnn_masks(org_d_array)
        org_l_array = get_dnn_masks(org_l_array)
        print("-----")
        print("Reshape the images to original dimensions")
        org_d_array = org_d_array.reshape(-1, 1024, 1024, 3)
        org_l_array = org_l_array.reshape(-1, 1024, 1024, 3)

        print("-----")
        print("Get the masks")
        group_d_array = get_grouped_masks(org_d_array)
        group_l_array = get_grouped_masks(org_l_array)

        print("-----")
        print("Get the image shapes")
        org_d_array = get_im_shapes(org_d_array)
        org_l_array = get_im_shapes(org_l_array)

        print("-----")
        print("Visualize the grouped images")
        visualize_grouped_images(org_d_array, N=10, show=True)
        visualize_grouped_images(org_l_array, N=10, show=True)

        print("-----")
        print("Visualize the grouped images with their masks")
        visualize_grouped_images(org_d_array, N=10, mask=N, visual=True, patch_experiment_name='patch', show=True)
        visualize_grouped_images(org_l_array, N=10, mask=N, visual=True, patch_experiment_name='patch', show=True)

        print("-----")
        print("Assign the image shapes")
        org_d_array = assign_im_shapes(org_d_array)
        org_l_array = assign_im_shapes(org_l_array)

        print("-----")
        print("Get the total image path")
        org_d_array = get_im_path(org_d_array)
        org_l_array = get_im_path(org_l_array)

        print("-----")
        print("Visualize the total image path")
        visualize_total_im_path(experiment_name, experiment='patch', show=True)

```

FIGURE 42: TEST 3

```

#----- Evaluate the results -----#
#----- predictions only inside the FOV -----#
y_scores = np.array([pred[0].ravel(), pred[1].ravel(), pred[2].ravel()])
test_border_masks = np.array([test[0].ravel(), test[1].ravel(), test[2].ravel()])
# returns data only inside the FOV
#----- print some slices -----#
print("y_scores shape: ", y_scores.shape[0]) # (radius 270: 270*270*1=216000)
print("y_scores slices [0]: ", y_scores.shape[0]) # (radius 270: 270*270*1=216000), Including background around retina: * +str(pred_imgs.shape[0])+pred_imgs.shape[0]
print("y_scores true pixels = ", y_scores.shape[0]) # (radius 270: 270*270*1=216000), including background around retina: * +str(y_scores.shape[0])+y_scores.shape[0]
print("y_scores true pixels = ", y_scores.shape[0]) # (radius 270: 270*270*1=216000), including background around retina: * +str(y_scores.shape[0])+y_scores.shape[0]

#----- Area under the ROC curve -----#
for fpr, tpr, thresholds in zip(fpr, tpr, thresholds):
    auc = metrics.auc(fpr, tpr)
    print("AUC: %f" % auc)
    if auc > 0.5:
        plt.figure()
        plt.title("ROC curve")
        plt.xlabel("False Positive Rate")
        plt.ylabel("True Positive Rate")
        plt.plot(fpr, tpr)
        plt.vlines(0, 0, 1, "red", "dashed")
        plt.hlines(1, 0, 1, "red", "dashed")
        plt.text(0.05, 0.95, "Area under the Curve (%s) = %s" % (label, auc))
        plt.savefig(path_experiments + "ROC.png")
        plt.close()

#----- precision-recall curves -----#
precision_recall_curve = np.array([precision_recall_curve(y_scores, y_scores)])
precision = np.array([precision_recall_curve[0][0]]) # Also the array is Increasing (you won't get negative AUC)
precision = np.array([precision_recall_curve[0][1]]) # Also the array is Increasing (you won't get negative AUC)
Auc_precision = np.array([precision_recall_curve[0][2]]) # Also the array is Increasing (you won't get negative AUC)
Add_auc_precision = np.array([precision_recall_curve[0][3]]) # Also the array is Increasing (you won't get negative AUC)
precision_recalls = np.array([precision_recall_curve[1][0]])
precision_recalls = np.array([precision_recall_curve[1][1]])
precision_recalls = np.array([precision_recall_curve[1][2]])
precision_recalls = np.array([precision_recall_curve[1][3]])

#----- print some slices -----#

```

## APPENDIX D

```

1 #preparing the hdf5 datasets of the DRIVE database
2
3
4 import os
5 import h5py
6 import numpy as np
7 from PIL import Image
8
9
10
11 def write_hdf5(arr,outfile):
12     with h5py.File(outfile,"w") as f:
13         f.create_dataset("image", data=arr, dtype=arr.dtype)
14
15
16 #####-----PATH TO THE IMAGES-----#####
17
18 #####-----train-----#####
19
20 original_ims_train = "./DRIVE/training/images/"
21 groundTruth_ims_train = "./DRIVE/training/1st_manual/"
22 borderMasks_ims_train = "./DRIVE/training/mask/"
23
24
25 original_ims_test = "./DRIVE/test/images/"
26 groundTruth_ims_test = "./DRIVE/test/1st_manual/"
27 borderMasks_ims_test = "./DRIVE/test/mask/"
28
29 #####-----test-----#####
30
31 Nims = 20
32 channels = 3
33 height = 480
34 width = 560
35 dataset_path = "./DRIVE/datasets_training_testing/"
36
37 def get_datasets(imgs_dir,groundTruth_dir,borderMasks_dir,train,test="null"):
38     img = np.empty((Nims,height,width,channels))
39     groundTruth = np.empty((Nims,height,width))
40     border_masks = np.empty((Nims,height,width))
41     for path, subdirs, files in os.walk(imgs_dir):#list path's directories and files
42         for name in files:
43             if name == "im0.png":#skip first file
44
45                 #Actual Data
46                 print("original image : ",files[i])
47                 img = Image.open(imgs_dir+files[i])
48                 img1 = np.asarray(img)
49
50                 #corresponding ground truth
51                 groundTruth_name = files[i][0:-4] + ".gtFine_labelTrainIds.png"
52                 print("Ground truth name : ",groundTruth_name)
53                 g_truth = Image.open(groundTruth_dir + groundTruth_name)
54                 g_truth1 = np.asarray(g_truth)
55
56                 #corresponding border masks
57                 border_masks_name = files[i][0:-4] + "_borderMask.gif"
58                 print("border masks name : " + border_masks_name)
59                 b_masks = Image.open(borderMasks_dir + border_masks_name)
60                 b_masks1 = np.asarray(b_masks)
61
62                 print("img max : ",str(np.max(img)))
63                 print("img min : ",str(np.min(img)))
64                 assert(np.max(groundTruth1)<255 and np.min(groundTruth1)>0)
65                 assert(np.max(b_masks1)<255 and np.min(b_masks1)>0)
66                 assert("gtFine" in groundTruth_name and "border" in border_masks_name)
67
68                 print("gtFine mask and border mask are correctly within pixel value range 0-255 (black-white)")
69
70                 #reshaping for my standard tensor
71                 img = np.transpose(img,(0,3,1,2))
72                 assert(img.shape == (Nims,channels,height,width))
73                 groundTruth = np.reshape(groundTruth,(Nims,1,height,width))
74                 border_masks = np.reshape(border_masks,(Nims,1,height,width))
75                 assert(groundTruth.shape == (Nims,1,height,width))
76                 assert(border_masks.shape == (Nims,1,height,width))
77                 assert(img.shape == groundTruth.shape)
78
79                 imgs.append(img)
80                 groundTruths.append(groundTruth)
81                 border_masks.append(border_masks)
82
83 if not os.path.exists(dataset_path):
84     os.makedirs(dataset_path)
85
86 #getting the training datasets
87 imgs_train, groundTruth_train, border_masks_train = get_datasets(original_ims_train,groundTruth_ims_train,borderMasks_ims_train,"train")
88 print("aving train datasets")
89 write_hdf5(imgs_train,dataset_path + "DRIVE_dataset_imgs_train.hdf5")
90 write_hdf5(groundTruth_train,dataset_path + "DRIVE_dataset_groundTruth_train.hdf5")
91 write_hdf5(border_masks_train,dataset_path + "DRIVE_dataset_borderMasks_train.hdf5")
92
93 #getting the testing datasets
94 imgs_test, groundTruth_test, border_masks_test = get_datasets(original_ims_test,groundTruth_ims_test,borderMasks_ims_test,"test")
95 print("aving test datasets")
96 write_hdf5(imgs_test,dataset_path + "DRIVE_dataset_imgs_test.hdf5")
97 write_hdf5(groundTruth_test,dataset_path + "DRIVE_dataset_groundTruth_test.hdf5")
98 write_hdf5(border_masks_test,dataset_path + "DRIVE_dataset_borderMasks_test.hdf5")
99
```

FIGURE 45: CONVERSION TO HDF5-1

```

55
56     #corresponding border masks
57     border_masks_name = ""
58     if train==True:
59         border_masks_name = files[i][0:-4] + "_borderMask.gif"
60     elif train==False:
61         border_masks_name = files[i][0:-4] + "_testMask.gif"
62     else:
63         print("Specify if train or test!")
64         exit()
65     print("border masks name : " + border_masks_name)
66     b_masks = Image.open(borderMasks_dir + border_masks_name)
67     b_masks1 = np.asarray(b_masks)
68
69     print("img max : ",str(np.max(img)))
70     print("img min : ",str(np.min(img)))
71     assert(np.max(groundTruth1)<255 and np.min(groundTruth1)>0)
72     assert(np.max(b_masks1)<255 and np.min(b_masks1)>0)
73     assert("gtFine" in groundTruth_name and "border" in border_masks_name)
74
75     print("gtFine mask and border mask are correctly within pixel value range 0-255 (black-white)")
76
77     #reshaping for my standard tensor
78     img = np.transpose(img,(0,3,1,2))
79     assert(img.shape == (Nims,channels,height,width))
80     groundTruth = np.reshape(groundTruth,(Nims,1,height,width))
81     border_masks = np.reshape(border_masks,(Nims,1,height,width))
82     assert(groundTruth.shape == (Nims,1,height,width))
83     assert(border_masks.shape == (Nims,1,height,width))
84
85     imgs.append(img)
86     groundTruths.append(groundTruth)
87     border_masks.append(border_masks)
88
89 if not os.path.exists(dataset_path):
90     os.makedirs(dataset_path)
91
92 #getting the training datasets
93 imgs_train, groundTruth_train, border_masks_train = get_datasets(original_ims_train,groundTruth_ims_train,borderMasks_ims_train,"train")
94 print("aving train datasets")
95 write_hdf5(imgs_train,dataset_path + "DRIVE_dataset_imgs_train.hdf5")
96 write_hdf5(groundTruth_train,dataset_path + "DRIVE_dataset_groundTruth_train.hdf5")
97 write_hdf5(border_masks_train,dataset_path + "DRIVE_dataset_borderMasks_train.hdf5")
98
99 #getting the testing datasets
100 imgs_test, groundTruth_test, border_masks_test = get_datasets(original_ims_test,groundTruth_ims_test,borderMasks_ims_test,"test")
101 print("aving test datasets")
102 write_hdf5(imgs_test,dataset_path + "DRIVE_dataset_imgs_test.hdf5")
103 write_hdf5(groundTruth_test,dataset_path + "DRIVE_dataset_groundTruth_test.hdf5")
104 write_hdf5(border_masks_test,dataset_path + "DRIVE_dataset_borderMasks_test.hdf5")
105
```

FIGURE 46: CONVERSION TO HDF5-2

## CODE AND RESULTS ARE ALSO AVAILABLE ON

<https://github.com/Huzmorgoth/Aritificial-Neural-Networks>