**CHAPTER 1**

**INTRODUCTION**

**1.1 MEDICAL IMAGING**

Medical Imaging is a process of creating visual representations of the interior of a body for clinical analysis and medical intervention, as well as visual representation of the function of some organs or tissues. Medical Imaging reveals the organs, tissues etc that are hidden beneath the skin to diagnose and treat the disease. Medical imaging also establishes a database of normal anatomy and physiology to make it possible to identify abnormalities.

**1.2 MEDICAL IMAGE COMPUTING**

Medical Image Computing is an interdisciplinary field at the intersection of computer science, data science, electrical engineering, physics, mathematics and medicine. This field develops computational and mathematical methods for solving problems pertaining to medical images and their use for biomedical research and clinical care.

The main goal of MIC is to extract clinically relevant information or knowledge from medical images. While closely related to the field of medical imaging, MIC focuses on the computational analysis of the images, not their acquisition.

**1.3 SEGMENTATION**

Segmentation is the process of partitioning an image into different meaningful segments. In medical imaging, these segments often correspond to different tissue classes, organs, pathologies, or other biologically relevant structures. Medical image segmentation is made difficult by low contrast, noise, and other imaging ambiguities. Although there are many computer vision techniques for image segmentation, some have been adapted specifically for medical image computing. The types of segmentation are classified into four categories.

**1.3.1 Atlas-Based Segmentation**

For many applications, a clinical expert can manually label several images; segmenting unseen images is a matter of extrapolating from these manually labeled training images. Methods of this style are typically referred to as atlas-based segmentation methods. Parametric atlas methods typically combine these training images into a single atlas image, while nonparametric atlas methods typically use all of the training images separately. Atlas-based methods usually require the use of image registration in order to align the atlas image or images to a new, unseen image.

**1.3.2 Shape-Based Segmentation**

Many methods parametrize a template shape for a given structure, often relying on control points along the boundary. The entire shape is then deformed to match a new image. Two of the most common shape-based techniques are Active Shape Models and Active Appearance Models. These methods have been very influential and have given rise to similar models.

**1.3.3 Image-Based segmentation**

Some methods initiate a template and refine its shape according to the image data while minimizing integral error measures, like the Active contour model and its variations.

**1.3.4 Interactive Segmentation**

Interactive methods are useful when clinicians can provide some information, such as a seed region or rough outline of the region to segment. An algorithm can then iteratively refine such segmentation, with or without guidance from the clinician. Manual segmentation, using tools such as a paint brush to explicitly define the tissue class of each pixel, remains the gold standard for many imaging applications. Recently, principles from feedback control theory have been incorporated into segmentation, which give the user much greater flexibility and allow for the automatic correction of errors.

**1.4 MACHINE LEARNING**

Machine learning is a field of computer science that gives computer systems the ability to "learn" with data, without being explicitly programmed. Machine learning explores the study and construction of algorithms that can learn from and make predictions on data. Such algorithms overcome following strictly static program instructions by making data-driven predictions or decisions, through building a model from sample inputs.

Machine learning is closely related to computational statistics, which also focuses on prediction-making through the use of computers. It has strong ties to mathematical optimization, which delivers methods, theory and application domains to the field. Machine learning is sometimes conflated with data mining, where the latter subfield focuses more on exploratory data analysis and is known as unsupervised learning. Machine learning can also be unsupervised and be used to learn and establish baseline behavioral profiles for various entities and then used to find meaningful anomalies. Machine learning tasks are typically classified into two broad categories, depending on whether there is a learning "signal" or "feedback" available to a learning system.

**1.4.1 Supervised learning**

The computer is presented with example inputs and their desired outputs, given by a "teacher", and the goal is to learn a general rule that maps inputs to outputs. As special cases, the input signal can be only partially available, or restricted to special feedback.

**1.4.2 Semi-supervised learning**

The computer is given only an incomplete training sample, a training set with some (often many) of the target outputs missing.

**1.4.3 Active learning**

Computer can only obtain training labels for a limited set of instances, and also has to optimize its choice of objects to acquire labels for. When used interactively, these can be presented to the user for labeling.

**1.4.4 Reinforcement learning**

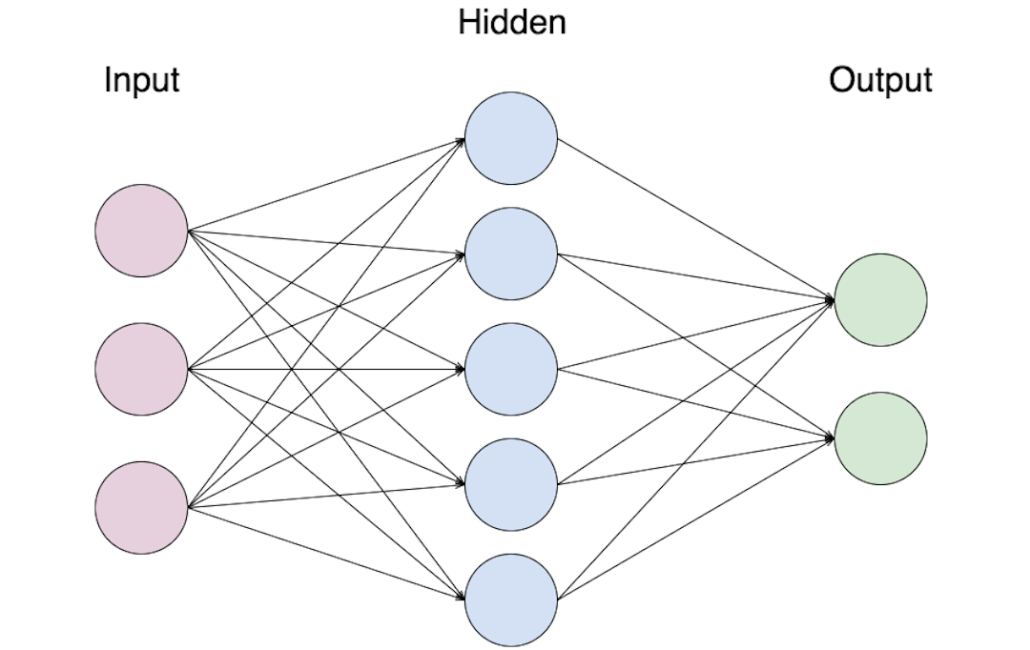
Training data (in form of rewards and punishments) is given only as feedback to the program's actions in a dynamic environment, such as driving a vehicle or playing a game against an opponent.

**1.4.5 Unsupervised learning**

No labels are given to the learning algorithm, leaving it on its own to find structure in its input. Unsupervised learning can be a goal in itself or a means towards an end.

**1.5 NEURAL NETWORK**

Neural network is a computing system inspired by the biological neural networks that constitute animal brains. It is commonly known as Artificial Neural Network (ANN). These networks "learn" tasks by considering examples, generally without task-specific programming.



**Figure 1.1 Basic Architecture of a Neural Network**

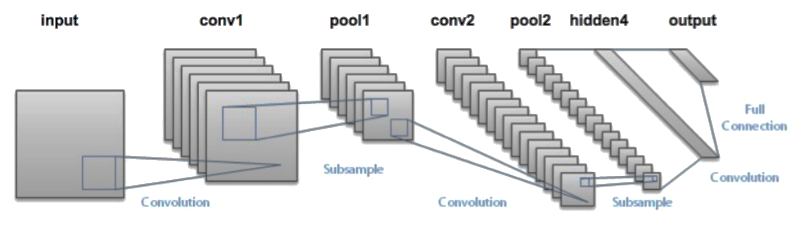
Figure 1.1 depicted above is the representation of a neural network. An ANN is based on a collection of connected units or nodes called artificial neurons. Each connection between artificial neurons can transmit a signal from one to another. The artificial neuron that receives the signal can process it and then signal artificial neurons connected to it.

During the implementation of an ANN, the signal at a connection between artificial neurons is a real number, and the output of each artificial neuron is calculated by a nonlinear function of the sum of its inputs. Artificial neurons and connections typically have a weight that adjusts as learning proceeds. The weight increases or decreases the strength of the signal at a connection. Artificial neurons may have a threshold such that only if the aggregate signal crosses that threshold is the signal sent.

Typically, artificial neurons are organized in layers. Different layers may perform different kinds of transformations on their inputs. Signals travel from the first (input), to the last (output) layer, possibly after traversing the layers multiple times.

**1.5.1 Convolutional Neural Network**

Convolutional Neural Network (CNN) is a class of deep, feed-forward artificial neural networks that has been used to analyze visual imagery. Convolutional networks were inspired by biological processes in that the connectivity pattern between neurons resembles the organization of the animal visual cortex. Individual cortical neurons respond to stimuli only in a restricted region of the visual field known as the receptive field. The receptive fields of different neurons partially overlap such that they cover the entire visual field. CNNs use relatively little pre-processing compared to other image classification algorithms. This means that the network learns the filters that in traditional algorithms were hand-engineered. This independence from prior knowledge and human effort in feature design is a major advantage.



**Figure 1.2 Basic Architecture of a Convolutional Neural Network**

Figure 1.2 illustrates the architecture of a CNN. A CNN consists of an input and an output layer, as well as multiple hidden layers. The hidden layers of a CNN typically consist of convolutional layers, pooling layers, fully connected layers and normalization layers.

Description of the process as a convolution in neural networks is by convention. Mathematically it is a cross-correlation rather than a convolution. This only has significance for the indices in the matrix, and thus which weights are placed at which index. Indeed, the CNN has a variety of features out of which some are given below.

**1.5.1.1 3D volumes of neurons**

The layers of a CNN have neurons arranged in 3 dimensions: width, height and depth. The neurons inside a layer are connected to only a small region of the layer before it, called a receptive field. Distinct types of layers, both locally and completely connected, are stacked to form a CNN architecture.

**1.5.1.2 Local connectivity**

It follows the concept of receptive fields. CNNs exploit spatial locality by enforcing a local connectivity pattern between neurons of adjacent layers. The architecture thus ensures that the learnt "filters" produce the strongest response to a spatially local input pattern. Stacking many such layers leads to non-linear filters that become increasingly global (i.e. responsive to a larger region of pixel space) so that the network first creates representations of small parts of the input, then from them assembles representations of larger areas.

**1.5.1.3 Shared weights**

In CNNs, each filter is replicated across the entire visual field. These replicated units share the same parameterization (weight vector and bias) and form a feature map. This means that all the neurons in a given convolutional layer respond to the same feature within their specific response field. Replicating units in this way allows for features to be detected regardless of their position in the visual field, thus constituting the property of translation invariance.

Together, these properties allow CNNs to achieve better generalization on vision problems. Weight sharing dramatically reduces the number of free parameters learned, thus lowering the memory requirements for running the network and allowing the training of larger, more powerful networks.

**1.6 PROBLEM DEFINITION**

To develop an application that would segment the vascular tree from the retinal fundus images thereby increasing the chance to find the cause of the disease.

**1.7 PROPOSED APPROACH**

The proposed system is to create an application that would train the images and the result of the trained image is then used to test the patient’s retinal fundus image and the vascular tree is segmented and various accuracy-based calculations are performed.

**CHAPTER 2**

**LITERATURE SURVEY**

Michael D. Abramoff et al (2010) reviews the methods on retinal image analysis and their impacts in the clinical procedures. A detailed study report on 2-D fundus imaging, Optical Coherence Tomography (OCT) imaging and their challenges were present. They also discussed about detection of vessel segmentation using fundus photography to identify diseases like age-related macular degeneration, diabetic retinopathy, and glaucoma. Method for segmentation of retinal images in 3-D OCT was also explained in detail. Their work reviews the image capturing methods and analysis and their clinical importance.

BoWu (2017) et al proposed automatic detection of microaneurysms in retinal images which helps to identify the diabetic retinopathy. Extracted features from the images are fed into K Nearest Neighbour Classifier, which classifies the true microaneurysms from the false candidates. They suggested that their model can be used in clinics to diagnose diabetic retinopathy as it provides good accuracy.

T.Walter (2002) et al developed computer assisted diagnosis to diagnose diabetic retinopathy. Presence of exudates is identified by high grey level variation from the images. Morphological filtering and watershed transformation are applied to detect the optic disc.

Their model finds the presence of exudates within the macular region with high sensitivity and predictive value.

Benson Shu Yan Lam (2008) et al proposed a method for vessel segmentation in retinal images. First they locate the center lines using normalized gradient vector field and then blood vessel like objects are detected using gradient vector field. Pruning is done to remove false blood vessel like images. Their proposed model experimented with Structured Analysis of the Retina (STARE) images and the results are compared with the ground truth.

Qing Li (2014) explained in-depth of CNN and its architecture and how it can be used in Medical Image Classiﬁcation. They customized CNN also compared the results with scale invariant feature transform, rotation-invariant local binary patterns feature with three resolutions and unsupervised feature learning using restricted Boltzmann machine. The existing approaches are coupled with Support Vector Machine (SVM) classiﬁcation model. Their customized CNN outperforms the other classification model which is not coupled with SVM.

A Cross-modality Learning Approach for Vessel Segmentation in Retinal Images was proposed by Qiaoliang Li (2016) et al. A deep neural network model segments the vessel from the input retinal images. Cross modality data transformation is done for vessel mapping and it automatically learns the vascular feature. This model produces high accuracy and robustness.

**CHAPTER 3**

**PROPOSED SYSTEM**

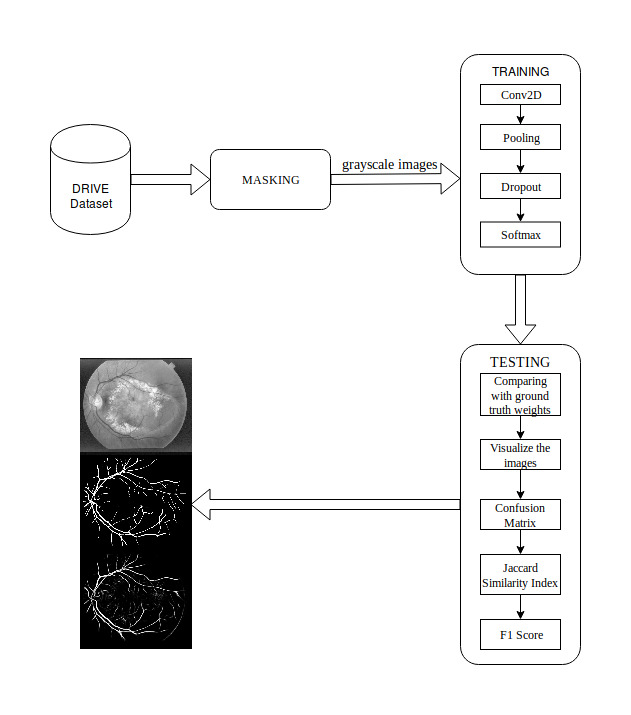
**3.1 INTRODUCTION**

This chapter deals with design aspects of the proposed system. System design is the process of defining the architecture, components, modules interfaces and data for a system to satisfy specified requirements. The main purpose of the design phase is to plan a solution to the problem specified in the requirement document.

In system design, the design functions and operations are described in detail, including the screen layouts, process diagrams, and other documentation. The output of this phase is the functional design document and will describe the new system as a collection of modules or subsystems. The design documents created for the proposed system consists of a system architecture, functional architecture, sequence diagrams, activity diagrams, aims to identify the modules that should be in the system, the specification of these modules, how they interact with each other to produce the desired results.

A distinction is sometimes drawn between high level or architectural design, which is concerned with the main components of the system and their roles and interrelationships, and detailed design, which is

concerned with internal structure and operation of individual components. The system architecture deals with the physical components and their interaction of the proposed system. The functional architecture defines the functionality of each module and their logical interaction. These diagrams are used as a reference point for further system development.

**3.2 PROPOSED SYSTEM ARCHITECTURE**

**Figure 3.1 Proposed System Architecture**

Figure 3.1 depicts the proposed architecture. The DRIVE Dataset is loaded into the memory as input and is masked using the mask image present in the dataset. Also, the retinal fundus and the mask images are compared for their size and other features. The training phase is where the masked image is given as input to the Convolutional Neural Network where activation functions and dropouts are applied to create a model for the given training dataset which is used in the testing stage. The testing program takes the model as input and segments the vascular tree from the retinal fundus images of the test dataset. The segmented image is compared with the ground truth images and scores are evaluated accordingly.

**3.3 MODULE DESCRIPTION**

**3.3.1 Masking**

Masking is a process of combining the retinal fundus images with the masks so that it forms an image which resembles the tracing activity where a tracing paper is placed on top of a desired image or whatsoever. Once the images are masked, they are converted from Red Green Blue (RGB) colour format to Grayscale colour format to reduce the number of dimensions and store only the required pixel value without its RGB value as it tends to take up memory while storing which decreases the performance of the program that is using the image data.

During the masking process, the retinal fundus images are compared with the mask images for their dimensions and cropping factor. If they are found to be different, proper resizing and adjustments are made without losing the crucial part of the retinal fundus image.

Once the images are prepared, they are converted to Hierarchical Data Format 5 (HDF5) file format and it is stored for further use. The usage of HDF5 file format is to store large amounts of data and assign object mapping so that the processing speed increases and the conventional file size limit is overridden.

**Pseudocode for Masking**

print "imgs max: " +str(np.max(imgs))

print "imgs min: " +str(np.min(imgs))

assert(np.max(groundTruth)==255 and p.max(border\_masks)==255)

assert(np.min(groundTruth)==0 and np.min(border\_masks)==0)

imgs = np.transpose(imgs,(0,3,1,2))

assert(imgs.shape == (Nimgs,channels,height,width))

groundTruth = np.reshape(groundTruth,(Nimgs,1,height,width))

border\_masks = np.reshape(border\_masks,(Nimgs,1,height,width))

assert(groundTruth.shape == (Nimgs,1,height,width))

assert(border\_masks.shape == (Nimgs,1,height,width))

*return* ***imgs, groundTruth, border\_masks***

**3.3.2 Training**

In this module, the masked retinal fundus images are given as inputs to the Convolutional Neural Network in which each stage has the following layers

* Convolution Layer
* Max Pooling Layer
* Rectified Linear Unit (ReLU) Layer
* Fully Connected Layer

**3.3.2.1 Convolution Layer**

This layer is responsible for passing the output of the convolution operation to the next layer. This is done by processing the data which is available in its receptive field or the sliding window. During the forward pass, each filter is convolved across the width and height of the input volume, computing the dot product between the entries of the filter and the input and producing a 2-dimensional activation map of that filter. As a result, the network learns filters that activate when it detects some specific type of feature at some spatial position in the input.

**3.3.2.2 Max Pooling Layer**

This layer is responsible for combining the outputs of the neuron clusters at one layer into a single neuron in the next layer. When the outputs are combined, the maximum value is taken and that becomes a neuron in the next layer. It partitions the input image into a set of non-overlapping rectangles and, for each such sub-region, outputs the maximum. The intuition is that the exact location of a feature is less important than its rough location relative to other features. The pooling layer serves to progressively reduce the spatial size of the representation, to reduce the number of parameters and amount of computation in the network, and hence to also control overfitting. It is common to periodically insert a pooling layer between successive convolutional layers in a CNN architecture. The pooling operation provides another form of translation invariance.



**Figure 3.2 Matrix Representation of a Max Pooling Layer**

**3.3.2.3 ReLU Layer**

Rectified Linear Unit layer is used to apply the activation function …… (3.1)

which increases the nonlinear properties of the decision function without affecting the receptive fields of the convolution layer. Other functions are also used to increase nonlinearity. For example, saturating hyperbolic tangent …… (3.2)

……. (3.3)

And the sigmoid function …… (3.4)

 ReLU is often preferred to other functions, because it trains the neural network several times faster without a significant penalty to generalisation accuracy.

**3.3.2.3 Fully Connected layer**

This layer is used during the last stage of the CNN. This is done for high level reasoning. Neurons in a fully connected layer have connections to all activations in the previous layer and their activations can then be computed with a matrix multiplication. Finally, after several convolutional and max pooling layers, the high-level reasoning in the neural network is done via fully connected layers. Neurons in a fully connected layer have connections to all activations in the previous layer, as seen in regular neural networks. Their activations can hence be computed with a matrix multiplication followed by a bias offset.

In addition to all these layers, a Dropout function is used to avoid overfitting of data while processing. Individual nodes are either "dropped out" of the net with probability or kept with probability so that a reduced network is left.

In the training stages, the probability that a hidden node will be dropped is usually 0.5 for input nodes, this should be much lower, intuitively because information is directly lost when input nodes are ignored.

At testing time after training has finished, find a sample average of all possible  {\displaystyle 2^{n}} dropped-out networks. However, it is possible to find an approximation by using the full network with each node's output weighted by a factor of  *p*{\displaystyle p}, so the expected value of the output of any node is the same as in the training stages. This is the biggest contribution of the dropout method, although it effectively generates {\displaystyle 2^{n}}22222 neural nets, and as such allows for model combination, at test time only a single network needs to be tested.

By avoiding training all nodes on all training data, dropout decreases overfitting. The method also significantly improves training speed. This makes model combination practical, even for deep neural nets. The technique seems to reduce node interactions, leading them to learn more robust features that better generalize to new data. Only the reduced network is trained on the data in that stage. The removed nodes are then reinserted into the network with their original weights.

Lastly, a Softmax function is used to normalize the data in the fully connected layer. It is the logistic function that squashes a *K*-dimensional vector **z** of arbitrary real values to a *K*-dimensional vector σ ( z ) of real values in the range (0, 1). Equation 3.5 is used for the calculation of the softmax activation function.

……. (3.5)

…… (3.6)

**Pseudocode for Training**

inputs = Input((n\_ch, patch\_height, patch\_width))

conv1 = Convolution2D(32, 3, 3, activation='relu', border\_mode='same')(inputs)

conv1 = Dropout(0.2)(conv1)

conv1 = Convolution2D(32, 3, 3, activation='relu', border\_mode='same')(conv1)

up1 = UpSampling2D(size=(2, 2))(conv1)

**3.3.3 Testing**

In this module, the images that are to be tested is loaded by the testing program and are divided into patches to decrease the load of the system and to produce better results. The masked retinal fundus images are tested with the trained model by applying hyperparameters like stride height, stride width, number of epochs and number of patches. The intermediate result is compared with the ground truth image data and the pixel scores are calculated. Using the pixel scores, the following are calculated.

**3.3.3.1 Confusion Matrix**

It is a specific table layout that allows visualization of the performance of an algorithm. Each row of the matrix represents the instances in a predicted class while each column represents the instances in an actual class. It is a special kind of contingency table, with two dimensions "actual" and "predicted", and identical sets of classes in both dimensions. Figure 3.3 depicts the confusion matrix.



**Figure 3.3 Confusion Matrix**

In predictive analytics, a table of confusion, is a table with two rows and two columns that reports the number of  *false positives*, *false negatives*, *true positives*, and *true negatives*. This allows more detailed analysis than mere proportion of correct classifications (accuracy). Accuracy is not a reliable metric for the real performance of a classifier, because it will yield misleading results if the data set is unbalanced (that is, when the numbers of observations in different classes vary greatly).

Using the confusion matrix shown in figure 3.3 values obtained like True Positives (TP), False Positives (FP), False Negatives (FN), True Negatives (TN), the Sensitivity, Specificity, Precision value and the Accuracy is calculated.

**3.3.3.2 Jaccard Similarity Index**

This is a statistic used for comparing the similarity and diversity of sample sets. The Jaccard coefficient measures similarity between finite sample sets and is defined as the size of the intersection divided by the size of the union of the sample sets. Equation 3.7 shows the calculation of the scores.

….. (3.7)

**3.3.3.3 Sensitivity**

It measures the proportion of positives that are correctly identified as such. It is also known as True Positive Rate (TPR) which is calculated using equation 3.8

….. (3.8)

**3.3.3.4 Specificity**

It measures the proportion of negatives that are correctly identified as such and calculated using equation 3.9

… (3.9)

**3.3.3.5 F1 Score**

It is a measure of a test's accuracy. It considers both the precision *p* and the recall *r* of the test to compute the score: *p* is the number of correct positive results divided by the number of all positive results returned by the classifier, and *r* is the number of correct positive results divided by the number of all relevant samples. The F1 score is the harmonic average of the precision and recall, where an F1 score reaches its best value at 1 and worst at 0. Equation 3.10 shows the formula to calculate the F1 score.

2 ….. (3.10)

**3.3.4 Choosing Hyperparameters**

**3.3.4.1 Number of filters**

Since feature map size decreases with depth, layers near the input layer will tend to have fewer filters while higher layers can have more. To equalize computation at each layer, the feature x pixel position product is kept roughly constant across layers. Preserving more information about the input would require keeping the total number of activations (number of feature maps times number of pixel positions) non-decreasing from one layer to the next. The number of feature maps directly controls capacity and depends on the number of available examples and task complexity.

**3.3.4.2 Filter Shape**

Common field shapes found in the literature vary greatly and are usually chosen based on the dataset. The challenge is thus to find the right level of granularity so as to create abstractions at the proper scale, given a particular dataset.

**3.3.4.3 Max Pooling Shape**

Typical values are 2x2. Very large input volumes may warrant 4x4 pooling in the lower-layers. However, choosing larger shapes will dramatically reduce the dimension of the signal, and may result in excess information loss. Often, non-overlapping pooling windows perform best.

**Pseudocode for Testing**

accuracy = 0

if float(np.sum(confusion))!=0:

accuracy = float(confusion[0,0]+confusion[1,1])/float(np.sum(confusion))

print "Global Accuracy: " +str(accuracy)

specificity = 0

if float(confusion[0,0]+confusion[0,1])!=0:

specificity = float(confusion[0,0])/float(confusion[0,0]+confusion[0,1])

print "Specificity: " +str(specificity)

sensitivity = 0

if float(confusion[1,1]+confusion[1,0])!=0:

sensitivity= float(confusion[1,1])/float(confusion[1,1]+confusion[1,0])

print "Sensitivity: " +str(sensitivity)

precision = 0

if float(confusion[1,1]+confusion[0,1])!=0:

precision =float(confusion[1,1])/float(confusion[1,1]+confusion[0,1])

print "Precision: " +str(precision)

***#Jaccard similarity index***

jaccard\_index = jaccard\_similarity\_score(y\_true, y\_pred, normalize=True)

print "\nJaccard similarity score: " +str(jaccard\_index)

***#F1 score***

F1\_score = f1\_score(y\_true, y\_pred, labels=None, average='binary', sample\_weight=None)

print "\nF1 score (F-measure): " +str(F1\_score)

**CHAPTER 4**

**SYSTEM REQUIREMENTS**

**4.1 INTRODUCTION**

The requirements specification is a technical specification of requirements for the software products. It is the first step in the requirements analysis process and it lists the requirements of a particular software system including functional, performance requirements. The requirements also provide usage scenarios from a user, an operational and an administrative perspective.

**4.2 HARDWARE REQUIREMENTS**

|  |  |
| --- | --- |
| 1. CPU - | Intel Core i7 4th gen @1.8GHz |
| 2. RAM - | 8GB |
| 3. Storage - | 10GB or higher |

**4.3 SOFTWARE REQUIREMENTS**

|  |  |
| --- | --- |
| 1. Operating System - | Ubuntu 17.10 |
| 2. Programming Language - | Python 2.7 |

**4.3.1 Python**

Python is developed under an OSI-approved open source license, making it freely usable and distributable, even for commercial use. Python's

license is administered by the Python Software Foundation. Python is used in many application domains. Python is a widely used high-level programming language for general-purpose programming, created by Guido van Rossum and first released in 1991. An interpreted language, Python has a design philosophy which emphasizes code readability (notably using whitespace indentation to delimit code blocks rather than curly braces or keywords), and a syntax which allows programmers to express concepts in fewer lines of code than possible in languages such as C++ or Java. The language provides constructs intended to enable writing clear programs on both a small and large scale.

Python interpreters are available for many operating systems, allowing Python code to run on a wide variety of systems. CPython, the reference implementation of Python, is open source software and has a community-based development model, as do nearly all of its variant implementations. CPython is managed by the non-profit Python Software Foundation.

**4.3.2 Tensorflow**

TensorFlow is an open source software library for machine learning across a range of tasks and developed by Google to meet their needs for systems capable of building and training neural networks to detect and decipher patterns and correlations, analogous to the learning and reasoning which humans use. It is currently used for both research and production at Google products, often replacing the role of its closed-source predecessor, DistBelief. TensorFlow was originally developed by the Google Brain team for internal Google use before being released under the Apache 2.0 open source license on November 9, 2015.

While the reference implementation runs on single devices, TensorFlow can run on multiple CPUs and GPUs (with optional CUDA extensions for general-purpose computing on graphics processing units). TensorFlow is available on 64-bit Linux, macOS, and mobile computing platforms including Android and iOS.

**4.3.3 Keras**

Keras is an open source neural network library written in Python. It is capable of running on top of TensorFlow, Microsoft Cognitive Toolkit, Theano, or MXNet. Designed to enable fast experimentation with deep neural networks, it focuses on being user-friendly, modular, and extensible. It was developed as part of the research effort of project ONEIROS (Open-ended Neuro-Electronic Intelligent Robot Operating System).

It offers a higher-level, more intuitive set of abstractions that make it easy to develop deep learning models regardless of the computational backend used.

Keras contains numerous implementations of commonly used neural network building blocks such as layers, objectives, activation functions, optimizers, and a host of tools to make working with image and text data easier.

Keras allows users to productize deep models on smartphones (iOS and Android), on the web, or on the Java Virtual Machine. It also allows use of distributed training of deep learning models on clusters of GPUs.

**4.3.4 Numpy**

NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.

Using NumPy in Python gives functionality comparable to MATLAB since they are both interpreted, and they both allow the user to write fast programs as long as most operations work on arrays or matrices instead of scalars. In comparison, MATLAB boasts a large number of additional toolboxes, notably Simulink, whereas NumPy is intrinsically integrated with Python, a more modern and complete programming language. Moreover, complementary Python packages are available. SciPy is a library that adds more MATLAB-like functionality and Matplotlib is a plotting package that provides MATLAB-like plotting functionality.

**CHAPTER 5**

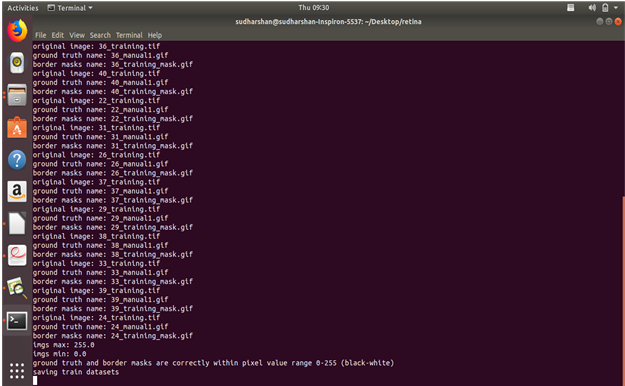
**SYSTEM IMPLEMENTATION**

**5.1 OVERVIEW**

This chapter describes the implementation of the model and screenshots of the modules that are used.

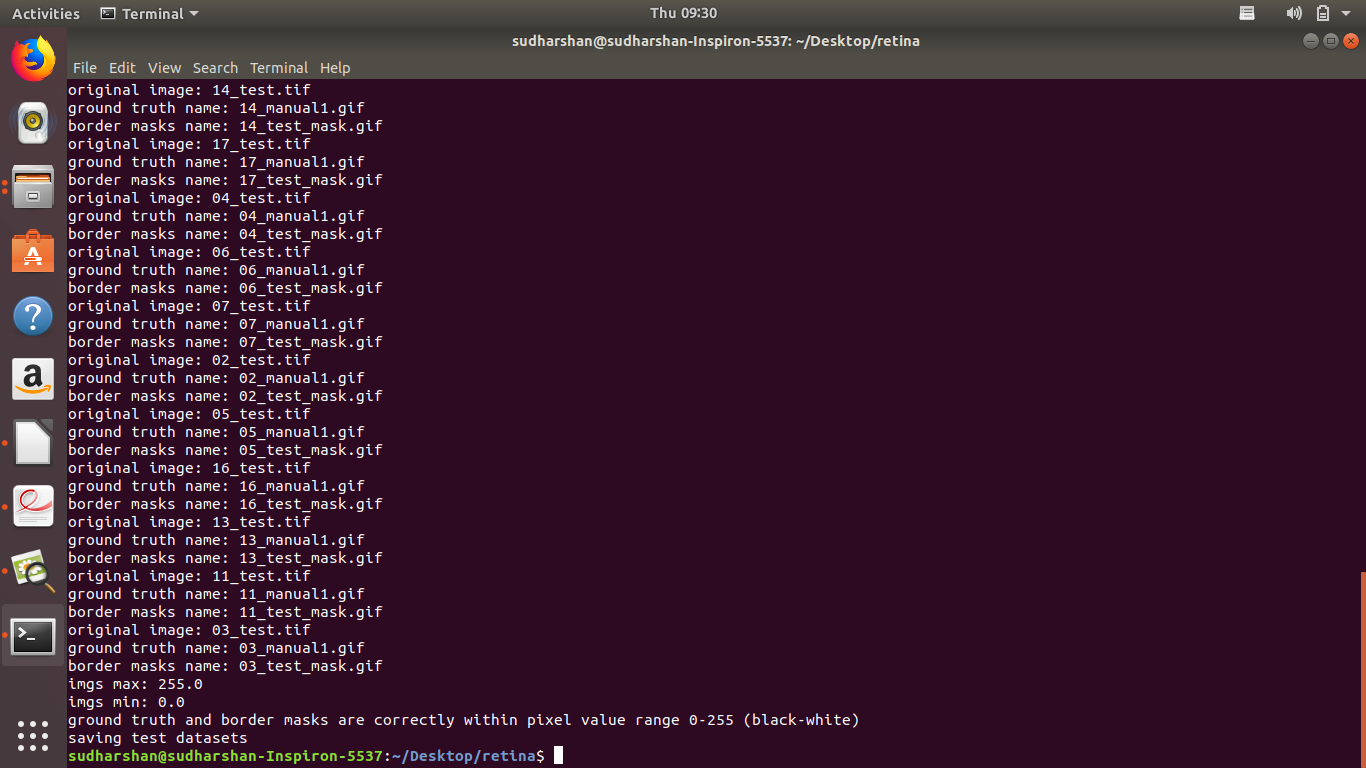
**5.2 IMPLEMENTATION**

**5.2.1 Preparing the datasets**

****

**Figure 5.1 Masked images**

**5.2.1.1 Ground truth**

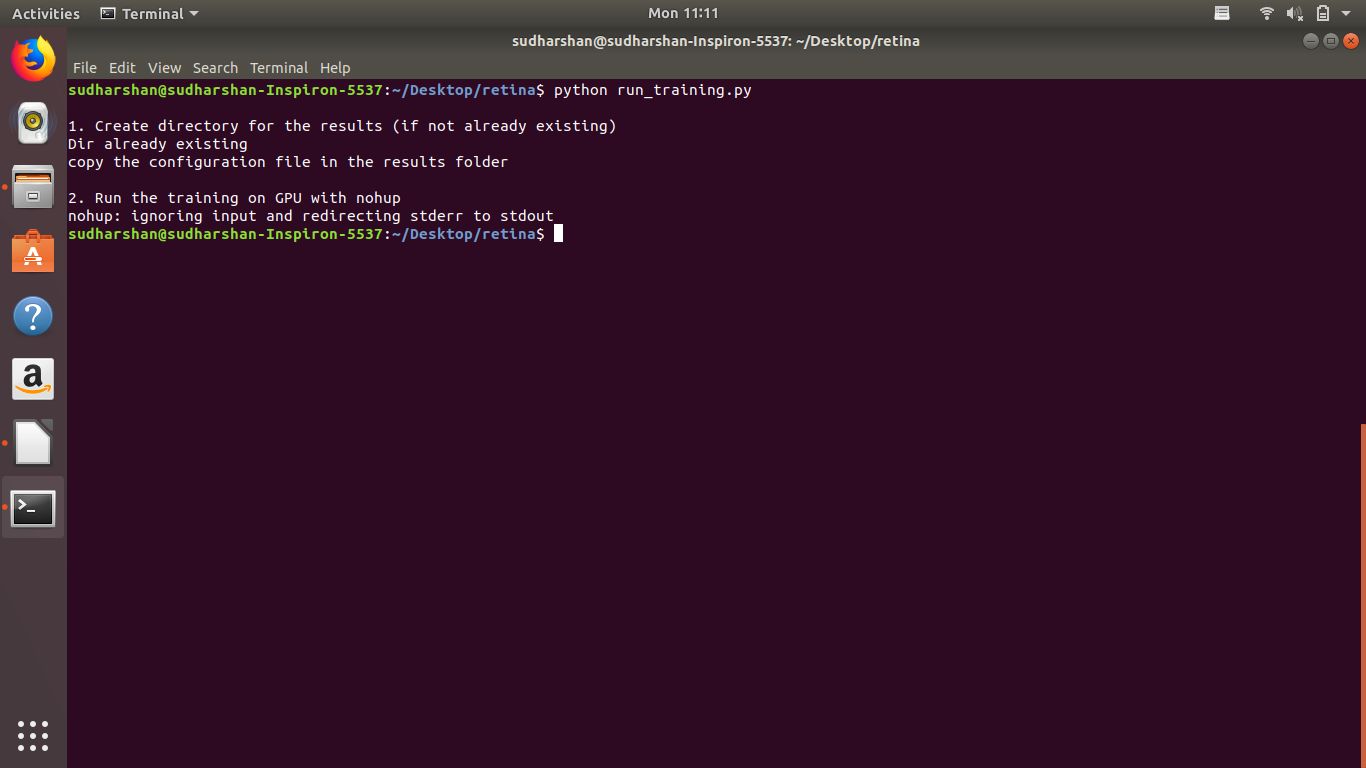
****

**Figure 5.2 Ground truth and masks in the pixel value range**

The images Figure 5.1 and Figure 5.2 shows the output of the datasets that are prepared for masking and to be converted to grayscale. The ground truth images are also prepared which is then used to compare with the final results.

The border masks are applied to the respective images and are checked for their resolution and its cropping factor. It is also checked for its pixel range so that there is no mismatch while the border masks are used to mask the retinal fundus images.

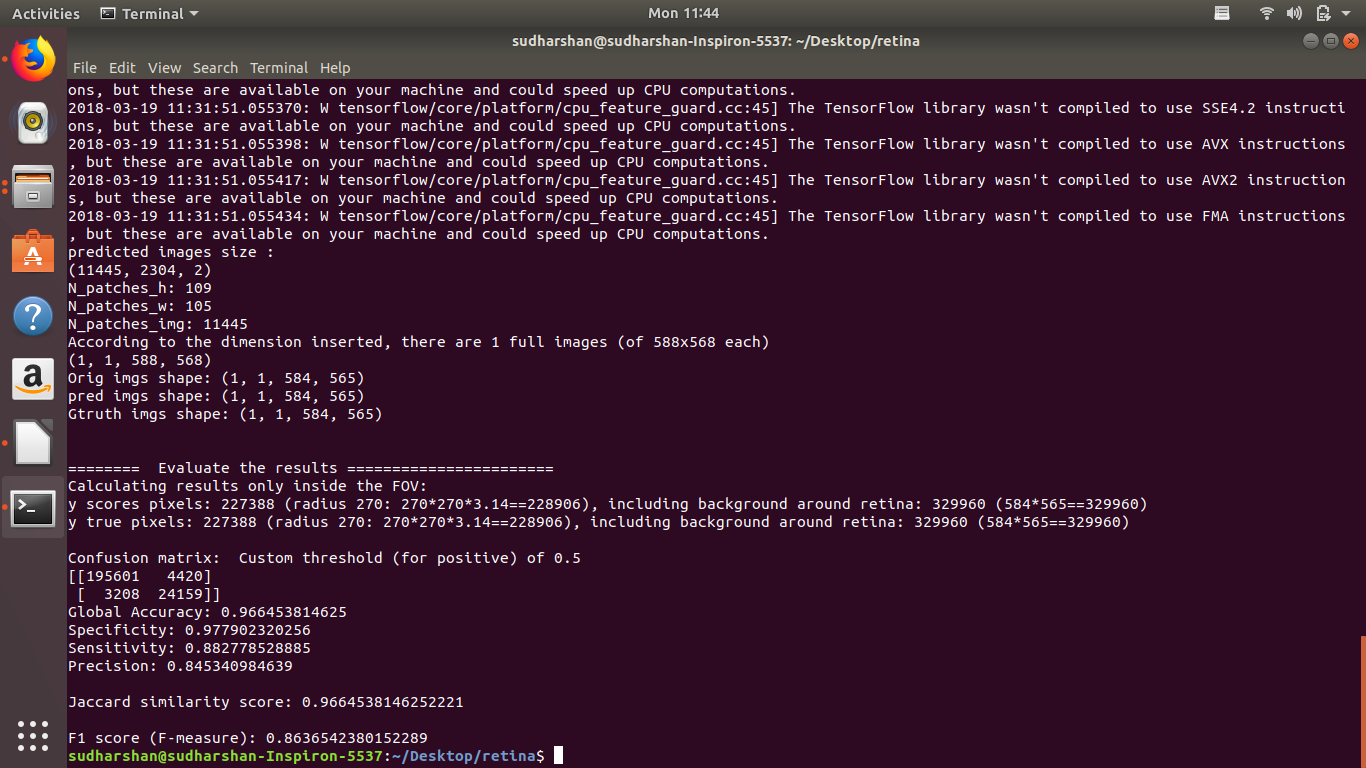
**5.2.2 Training**



**Figure 5.3 Output of the training program**

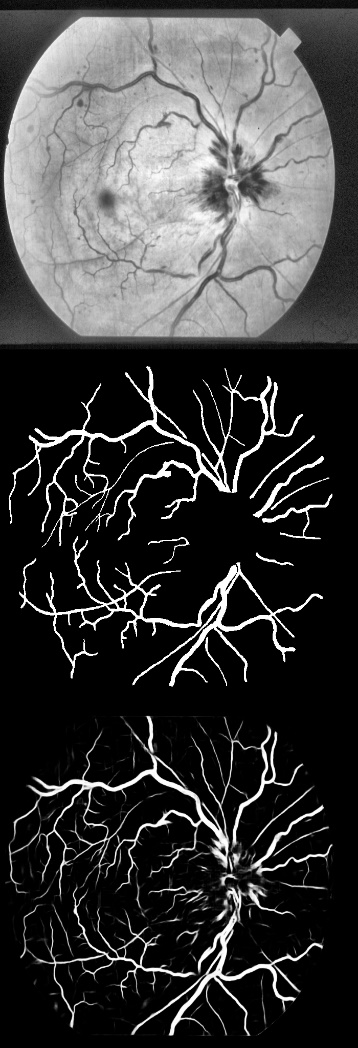
Once the prepared images are trained, the program writes the output to a directory which is created if not present. The output serves as a model for testing the images. Nohup, which is mentioned in the Figure 5.3 is a hand signal issued when the operating system doesn’t make calls for the explicit termination of the program. The results are then stored in the respective directory after the nohup signal is issued.

**5.2.3 Testing**



**Figure 5.4** **Output of the test results**

Figure 5.4 shows the results of the tested images. Various calculations are performed to numerically show various attributes like accuracy, specificity, Jaccard similarity index and F1 score. The segmented images are stored in a separate directory.

****

**Figure 5.5 Analysis of the segmented image with the ground truth and original image**

Figure 5.5 shows the comparison of the segmented image (right) to the ground truth image (center), it shows a near accurate segmented result but also has some wrongly segmented areas such as the macula region where the blood vessels are not present. This can be found out by comparing the original image (left) to the ground truth image.

**CHAPTER 6**

**CONCLUSION AND FUTURE WORK**

The proposed system was tested with 11445 patches of retinal fundus images. The vascular trees were accurately segmented for most of the ground truths provided and the F1 score was found to be 0.86.

In future, this application can be hosted in a remote server and the images for testing can be obtained by using optometric lenses which can then be used to photograph the retinal fundus images even by ones who do not have specialized training. This would be very useful while testing in remote areas where the conventional mydriatic camera cannot be taken.

**REFERENCES**

1. Abramoff M.D. (2013), ‘Automated analysis of retinal images for detection of referable diabetic retinopathy’, JAMA Ophthalmology., Vol. 131, No.3, pp. 351-357.

2. Bowei Feng, Huisheng Zang, LienPei Xei, Ping Liang, Qioliang Li, Tianfu Wang(2016), ‘A Cross-modality Learning Approach for Vessel Segmentation in Retinal Images’,IEEE Transactions on Medical Imaging, Vol.35, No.1,pp. 109-118.

3. Carreira M.J, Espona .L, Ortega .M,Penedo M.G. (2008)‘Retinal vessel tree segmentation using a deformable contour model’,  19th International Conference on Pattern Recognition, pp. 2128-2131.

4. Fraz M.M. (2012) ‘Blood vessel segmentation methodologies in retinal images—A survey’, Computer Methods and Programs in Biomedicine, Vol. 108, No.1, pp. 407-433.

5. Fraz M.M. (2012), ‘An approach to localize the retinal blood vessels using bit planes and centerline detection’, Computer Methods and Programs in Biomedicine, Vol. 108, No:2,pp. 600-616.

6. Kirbas C and Quek F. (2004), ‘A review of vessel extraction techniques and algorithms’, ACM Computing Surveys, Vol. 36, No.2, pp. 81-121.

7. Klein J.C. and Zana F. (1999), ‘A multimodal registration algorithm of eye fundus images using vessels detection and Hough transform’, IEEE Transactions on Medical Imaging, Vol. 18, No. 5, pp. 419-428.

8. Lam B.S.Y and Yan H. (2008), ‘A novel vessel segmentation algorithm for pathological retina images based on the divergence of vector fields’, IEEE Transactions on Medical Imaging, Vol. 27, No. 2, pp. 237-246.

9. DRIVE Dataset. https://www.isi.uu.nl/Research/Databases/DRIVE/

10. Tensorflow. https://www.tensorflow.org/

11. Keras. https://keras.io/

12. NumPy. https://docs.scipy.org/doc/numpy-dev/user/quickstart.html

13. Python 2.7. https://docs.python.org/2/

14. ConvolutionalNeuralNetwork https://en.wikipedia.org/wiki/Convolutional\_neural\_network

15. CNN Basics https://pythonprogramming.net/convolutional-neural-network-cnn-machine-learning-tutorial/

16. Machine Learning Basics https://machinelearningmastery.com/about/

17. Feature Learning http://ufldl.stanford.edu/tutorial/supervised/ConvolutionalNeuralNetwork/

**APPENDIX**

**Source Code**

for path, subdirs, files in os.walk(imgs\_dir): #list all files, directories in the path

fori in range(len(files)):

print "original image: " +files[i]

img = Image.open(imgs\_dir+files[i])

imgs[i] = np.asarray(img)

groundTruth\_name = files[i][0:2] + "\_manual1.gif"

print "ground truth name: " + groundTruth\_name

g\_truth = Image.open(groundTruth\_dir + groundTruth\_name)

groundTruth[i] = np.asarray(g\_truth)

border\_masks\_name = ""

iftrain\_test=="train":

border\_masks\_name = files[i][0:2] + "\_training\_mask.gif"

eliftrain\_test=="test":

border\_masks\_name = files[i][0:2] + "\_test\_mask.gif"

else:

print "specify if train or test!!"

exit()

if not os.path.exists(dataset\_path):

os.makedirs(dataset\_path)

#getting the training datasets

imgs\_train, groundTruth\_train, border\_masks\_train = get\_datasets(original\_imgs\_train,groundTruth\_imgs\_train,borderMasks\_imgs\_train,"train")

print "saving train datasets"

write\_hdf5(imgs\_train, dataset\_path + "DRIVE\_dataset\_imgs\_train.hdf5")

write\_hdf5(groundTruth\_train, dataset\_path + "DRIVE\_dataset\_groundTruth\_train.hdf5")

write\_hdf5(border\_masks\_train,dataset\_path + "DRIVE\_dataset\_borderMasks\_train.hdf5")

conv10 = Convolution2D(2, 1, 1, activation='relu', border\_mode='same')(conv9)

conv10 = core.Reshape((2,patch\_height\*patch\_width))(conv10)

conv10 = core.Permute((2,1))(conv10)

conv10 = core.Activation('softmax')(conv10)

model = Model(input=inputs, output=conv10)

model.compile(optimizer='sgd', loss='categorical\_crossentropy',metrics=['accuracy'])

return model

patches\_imgs\_train, patches\_masks\_train = get\_data\_training(

DRIVE\_train\_imgs\_original = path\_data + config.get('data paths', 'train\_imgs\_original'),

DRIVE\_train\_groudTruth = path\_data + config.get('data paths', 'train\_groundTruth'), #masks

patch\_height = int(config.get('data attributes', 'patch\_height')),

patch\_width = int(config.get('data attributes', 'patch\_width')),

N\_subimgs = int(config.get('training settings', 'N\_subimgs'))

n\_ch = patches\_imgs\_train.shape[1]

patch\_height = patches\_imgs\_train.shape[2]

patch\_width = patches\_imgs\_train.shape[3]

printmodel.output\_shape

plot(model, to\_file='./'+name\_experiment+'/'+name\_experiment + '\_model.png')

json\_string = model.to\_json()

open('./'+name\_experiment+'/'+name\_experiment +'\_architecture.json', 'w').write(json\_string)

path\_data = config.get('data paths', 'path\_local')

DRIVE\_test\_imgs\_original = path\_data + config.get('data paths', 'test\_imgs\_original')

test\_imgs\_orig = load\_hdf5(DRIVE\_test\_imgs\_original)

full\_img\_height = test\_imgs\_orig.shape[2]

full\_img\_width = test\_imgs\_orig.shape[3]

DRIVE\_test\_border\_masks = path\_data + config.get('data paths', 'test\_border\_masks')

test\_border\_masks = load\_hdf5(DRIVE\_test\_border\_masks)

patch\_height = int(config.get('data attributes', 'patch\_height'))

patch\_width = int(config.get('data attributes', 'patch\_width'))

stride\_height = int(config.get('testing settings', 'stride\_height'))

stride\_width = int(config.get('testing settings', 'stride\_width'))

assert (stride\_height<patch\_height and stride\_width<patch\_width)

name\_experiment = config.get('experiment name', 'name')

path\_experiment = './' +name\_experiment +'/'

Imgs\_to\_test = int(config.get('testing settings', 'full\_images\_to\_test'))

N\_visual = int(config.get('testing settings', 'N\_group\_visual'))

average\_mode = config.getboolean('testing settings', 'average\_mode')

patches\_imgs\_test = None

new\_height = None

new\_width = None

masks\_test = None

patches\_masks\_test = None

ifaverage\_mode == True:

patches\_imgs\_test, new\_height, new\_width, masks\_test = get\_data\_testing\_overlap(

DRIVE\_test\_imgs\_original = DRIVE\_test\_imgs\_original, #original

DRIVE\_test\_groudTruth = path\_data + config.get('data paths', 'test\_groundTruth'), #masks

Imgs\_to\_test = int(config.get('testing settings', 'full\_images\_to\_test')),

patch\_height = patch\_height,

patch\_width = patch\_width,

stride\_height = stride\_height,

stride\_width = stride\_width

)

else:

patches\_imgs\_test, patches\_masks\_test = get\_data\_testing(

DRIVE\_test\_imgs\_original = DRIVE\_test\_imgs\_original,

DRIVE\_test\_groudTruth = path\_data + config.get('data paths', 'test\_groundTruth'), #masks

Imgs\_to\_test = int(config.get('testing settings', 'full\_images\_to\_test')),

patch\_height = patch\_height,

patch\_width = patch\_width,

)

threshold\_confusion = 0.5

print "\nConfusion matrix: Custom threshold (for positive) of " +str(threshold\_confusion)

y\_pred = np.empty((y\_scores.shape[0]))

fori in range(y\_scores.shape[0]):

ify\_scores[i]>=threshold\_confusion:

y\_pred[i]=1

else:

y\_pred[i]=0

confusion = confusion\_matrix(y\_true, y\_pred)

print confusion

accuracy = 0

if float(np.sum(confusion))!=0:

accuracy = float(confusion[0,0]+confusion[1,1])/float(np.sum(confusion))

print "Global Accuracy: " +str(accuracy)

specificity = 0

if float(confusion[0,0]+confusion[0,1])!=0:

specificity = float(confusion[0,0])/float(confusion[0,0]+confusion[0,1])

print "Specificity: " +str(specificity)

sensitivity = 0

if float(confusion[1,1]+confusion[1,0])!=0:

sensitivity = float(confusion[1,1])/float(confusion[1,1]+confusion[1,0])

print "Sensitivity: " +str(sensitivity)

precision = 0

if float(confusion[1,1]+confusion[0,1])!=0:

precision = float(confusion[1,1])/float(confusion[1,1]+confusion[0,1])

print "Precision: " +str(precision)

#Jaccard similarity index

jaccard\_index = jaccard\_similarity\_score(y\_true, y\_pred, normalize=True)

print "\nJaccard similarity score: " +str(jaccard\_index)

#F1 score

F1\_score = f1\_score(y\_true, y\_pred, labels=None, average='binary', sample\_weight=None)

print "\nF1 score (F-measure): " +str(F1\_score)

#Save the results

file\_perf = open(path\_experiment+'performances.txt', 'w')

file\_perf.write("Jaccard similarity score: " +str(jaccard\_index)

+ "\nF1 score (F-measure): " +str(F1\_score)

+"\n\nConfusion matrix:"

+str(confusion)

+"\nACCURACY: " +str(accuracy)

+"\nSENSITIVITY: " +str(sensitivity)

+"\nSPECIFICITY: " +str(specificity)

+"\nPRECISION: " +str(precision)

)

file\_perf.close()