**OPTICAL CHARACTER RECOGNITION USING NEURAL NETWORK**

**A MINI PROJECT REPORT**

***Submitted by***

Madhusoodan M Pataki 2SD14CS054

Anarghya K Jantali 2SD14CS128

***in partial fulfilment for the award of the degree***

**Bachelor of Engineering**

***in***

**Computer Science and Engineering**

**Under the Guidance of**

**Prof. Sharada H N**

**6th Semester B.E Academic Year: 2016-17**

**SDM COLLEGE OF ENGINEERING AND TECHNOLOGY,**

**DHARWAD-580002**

May 2017

SDM College of Engineering and Technology,

Dharwad-580002

Department of Computer Science & Engineering.

**CERTIFICATE**

Certified that the project work entitled “OPTICAL CHARACTER RECOGNITION USING NEURAL NETWORK” is an original work carried out by Mr. Madhusoodan M Pataki and Mr. Anarghya K Jantali in partial fulfillment for the award of degree of Bachelor of Engineering in Computer Science and Engineering of S.D.M College of Engineering and Technology, Dharwad-580002, during the year 2016-17. The project report has been approved as it satisfies the academic requirements in respect of mini project work prescribed for Bachelor of Engineering Degree.

Signature of the Guide Signature of HOD Signature of Principal

**Name of the Student :** Mr. Madhusoodan M Pataki

**University Seat Number :** 2SD14CS054

**Viva-Voce Committee:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Sl. No.** | **Name** | **Designation** | **Signature with Date** |
| 1 |  |  |  |
| 2 |  |  |  |
| 3 |  |  |  |
| 4 |  |  |  |

SDM College of Engineering and Technology,

Dharwad-580002

Department of Computer Science & Engineering.

**CERTIFICATE**

Certified that the project work entitled “OPTICAL CHARACTER RECOGNITION USING NEURAL NETWORK” is an original work carried out by Mr. Madhusoodan M Pataki and Mr. Anarghya K Jantali in partial fulfillment for the award of degree of Bachelor of Engineering in Computer Science and Engineering of S.D.M College of Engineering and Technology, Dharwad-580002, during the year 2016-17. The project report has been approved as it satisfies the academic requirements in respect of mini project work prescribed for Bachelor of Engineering Degree.

Signature of the Guide Signature of HOD Signature of Principal

**Name of the Student :** Mr. Anarghya K Jantali

**University Seat Number :** 2SD14CS128

**Viva-Voce Committee:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Sl. No.** | **Name** | **Designation** | **Signature with Date** |
| 1 |  |  |  |
| 2 |  |  |  |
| 3 |  |  |  |
| 4 |  |  |  |

**ABSTRACT**

Optical Character Recognition (OCR) deals with recognition and conversion of printed characters in a scanned document that is taken as an input to an editable, digitized text. This improves the storage, processing, mining that are pivotal in further development of essential applications that serve various domains. We are implementing such a process, which takes a scanned/captured image as an input, pre-process the image, recognize the characters of English language and output in a form of string to a text document. Current scope is limited to English characters and digits. Recognition is also limited to printed characters rather than manuscript/handwriting. This process can be improvised to recognize various languages as well as handwritten materials. The input image is put through pre-processing, segmentation and feature extraction. Feature extraction and recognition of characters is implemented with Neural-Network. Using neural network to train and recognize the characters increases the coverage as well as accuracy of the results compared to conventional horizontal and vertical feature extraction methods. With increased accuracy, this system can be implemented in applications such as form reading such as cheque, postal code, official letters etc. This can also be augmented with a dictionary and intelligent interpreter to evaluate answer scripts etc.

**TABLE OF CONTENTS**

**CHAPTER NO. TITLE PAGE NO.**

**ABSTRACT i**

**LIST OF TABLE iii**

**LIST OF FIGURES iii LIST OF SYMBOLS iii**

**1. INTRODUCTION 1**

GENERAL 1

1.1 OBJECTIVES 2

1.2 REQUIREMENTS AND LIMITATIONS 3

**2. WORKING AND IMPLEMENTATION 4**

2.1 Pre-Processing 4

2.2 Segmentation 7

2.3 Feature Extraction 8

2.4 Training 9

2.5 Testing 10

**3. ARCHITECTURE 11**

Pattern used and class diagram 11

**4. USER INTERFACE 13**

4.1 Input Image 14

4.2 Importing training set. 15

4.3 Importing testing set. 16

4.4 Neural-Network saving and loading 17

**5. NEURAL NETWORK. 18**

**6. LOOP-HOLES & IMPROVEMENTS 22**

6.1 Loop Holes 22

6.2 Some improvements 22

**7. SUB-PROCESSES and TOOLS 23**

7.1 Platform 23

7.2 Tools and Frameworks 23

7.4 Training set generation 23

**TRAINING AND TESTING SAMPLE 24**

**REFERENCES 25**

**LIST OF TABLES**

1. Trainingset size vs accuracy. 20
2. Learningrate vs. network convergence and accuracy. 21

**LIST OF FIGURES**

1. Use-case diagram 3
2. Binarization 6
3. Segmentation sample 7
4. Sequence diagram of training. 9
5. Neural-Network structure 9
6. Activity diagram of recognition 10
7. Class Diagram 12
8. Main UI 13
9. Binarization in UI 14
10. Import training set 15
11. Error graph 16
12. Testing set import 16
13. Load and Saving the Neural Network 17
14. Neuron structure 18
15. Incorrectly recognized characters 22

**CHAPTER 1. INRODUCTION**

Character recognition is one of the problem in computing. Due to sequential nature of the computing, this has been a conventional hurdle in image processing and pattern recognition domain. Several works have been published in regards to improve processing time and recognition accuracy. Drastic improvements have been made in two prominent methods that are pattern recognition of characters as well as feature extraction. Pattern recognition in general, has an inherent limitation to the model due to decreased accuracy, when faced with new font/handwriting. Maturing science of neural networks has made it possible to input a feature to neural- network, proving classification of characters with better accuracy.

Character recognition proposed in this paper is offline-recognition. An image is given to the system after it is scanned/captured. This also, involves problems such as noise, orientation, bleaching etc. As such, pre-processing of the image is a necessity in offline recognition systems. Pre-processing involves steps that are required to shape image into suitable form for segmentation. This step involves skewing, noise-reduction, bleaching etc. The cleaned up image is then given as input to segmentation

Segmentation is a process of identifying individual glyphs present in the image. The characters are read line by line, spaces, punctuations and characters are identified separately. The character glyphs are normalized to a standard pixel size, so that extraction of features of character glyphs is made easy.

The Selection of appropriate feature extraction method is probably the single most important factor in achieving high recognition performance. Features can be classified into two sets, statistical features and structural feature. In statistical feature, an image is decomposed into a set of n dimensional identifiers. In structural featuring, end points, intersections, convexity, strokes etc. define the geometrical attributes of a character. Several methods of feature extraction for character recognition have been reported in the literature [2]. The widely used feature extraction methods are Template matching, Unitary Image transforms, Projection Histograms, Contour profiles, Zoning, Nearest-Neighbor rounding etc.

The features extracted are mapped to characters to train the neural network. An artificial neural Network as the backend is used for performing classification and recognition tasks. Classification techniques have been applied to handwritten character recognition since the 1990s. These methods include statistical methods based on Bayes decision rule, Artificial Neural Networks (ANNs), Kernel Methods including Support Vector Machines (SVM) and multiple classifier combination [3], [4].

General process involved in Offline Character Recognition systems include

1. Preprocessing

2. Segmentation

3. Feature Extraction

4. Classification.

**1.1 Objectives**

* To convert an Image information to editable character stream.
* To reduce the size of information, by removing unnecessary information in image.
* To preserve the document, make the processing of document easier.
* Build OCR from the scratch, without any available API's.
* Make way for further addition of dictionary/spell correction, semantic analysis etc.

**1.2 Requirements**

**Functional Requirements**

* Input an image containing characters of English letters and digits.
* Output the recognized characters in a pertinent document.

**Non-Functional Requirements**

* Accuracy of the characters should not be a random guess.
* Train the Neural network with training set.
* Load the previously trained and stored network.

*Figure 1. Use case diagram of mpocr.*

**Limitations**

1. It is only limited only to English language letters a-z, A-Z, digits 0-9.
2. Pre-processing is not the major focus, hence noisy images, skewed images get low recognition accuracy.
3. Images should contain a stronger de-lineation between foreground and background.
4. The Neural-Network is customized to this software, reuse is not possible.

Currently only tested for printed characters, not for handwritten characters

**CHAPTER 2. WORKING AND IMPLEMENTATION.**

The core of character recognition lies in Neural-Network used. The use of ANN in the application divides the application into two phases. which are given below. Both the phases are same except one step where former has a training step and latter has a testing step. We don’t explain the phases in separate as they have many similarities.

1. Learning phase.

2. Working/Testing phase.

**2.1 Preprocessing.**

Since the images comprises of pixilated information, the information of each pixel is necessary to identify the present characters. Misrepresentation can easily lead to characters not being recognized. Hence, pre-processing plays a very important role in Offline Character Recognition systems. Pre-processing, as the name suggests comprises of the necessary tidying up job of the image before any actual processing can be carried out. Some of the basic operations carried out in pre-processing include

* Binarization: This process involves conversion of an image to two-tone, monochromatic image. Threshold of colored images are calculated, based on light intensity to either dark or light pixels. This removes unwanted information such as colors and represents the image as foreground and background planes. The foreground plane is represented by 1 and background plane by 0. To binarize our image, we have use Otsu’s method, which is a very clever method created by Nobuyuki Otsu. The algorithm assumes that the image to be binarized contains two classes of pixels or bi-modal histogram (e.g. foreground and background) then calculates the optimum threshold separating those two classes so that their combined spread (intra-class variance) is minimal. Its main advantages are speed (because we only use histograms and arrays of length 256) and the easiness of the implementation. To binarize an image, we first need to convert it to grayscale. To convert it to grayscale we use luminance preserving method.
  + **Gray Scaling the luminance preserving method** uses intensity of the light to be preserved as an approach to convert an image to grayscale. The Formula is given by:
    - Y= 0.2126 R + 0.7152 G + 0.0722 B

Here Y is the total intensity of a pixel. R,G,B represent the linear red, green and blue values of a pixel and the co-efficient indicate the proportionality of intensity contributed by each color to the overall intensity of the pixel. It is calculated by photometric method of luminance preserving. Now the image is converted to grayscale. To binarize an image, we need to select a threshold. Manual selection isn’t the good way. Otsu’s method finds that binary threshold that we are looking for (this is just one of the methods to binarize an image). The method involves iterating through all the possible threshold values and calculating a measure of spread for the pixel levels each side of the threshold, i.e. the pixels that either fall in foreground or background. The aim is to find the threshold value where the sum of foreground and background spreads is at its minimum.

**Mathematical background behind Otsu**

Let q_1 and q_2 represent the estimate of class probabilities defined as: [1](https://bostjan-cigan.com/java-image-binarization-using-otsus-algorithm/#footnote_0_1233)

q_1(t) = \sum_{i=1}^t P(x)

q_2(t) = \sum_{i=t+1}^I P(x)

And the class means:

\mu_1(t) = \sum_{i=1}^t \frac{i P(i)}{q_1 (t)}

\mu_2(t) = \sum_{i=t+1}^I \frac{i P(i)}{q_2 (t)}

P represents the image histogram. The problem of minimizing within class variance can be expressed as a maximization problem of the between class variance. It can be written as a difference of total variance and within class variance:

\sigma_b^2 = \sigma^2 - \sigma_w^2(t) = q_1(t)[1 - q_1(t)][\mu_1(t) - \mu_2(t)]^2

Finally, this expression can safely be maximized and the solution is t that is maximizing\sigma_b^2(t).

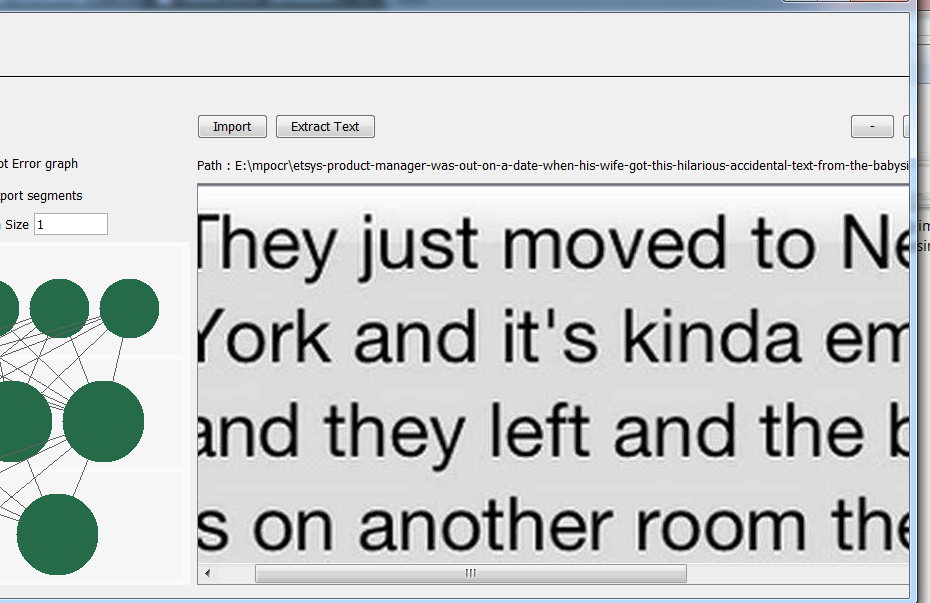
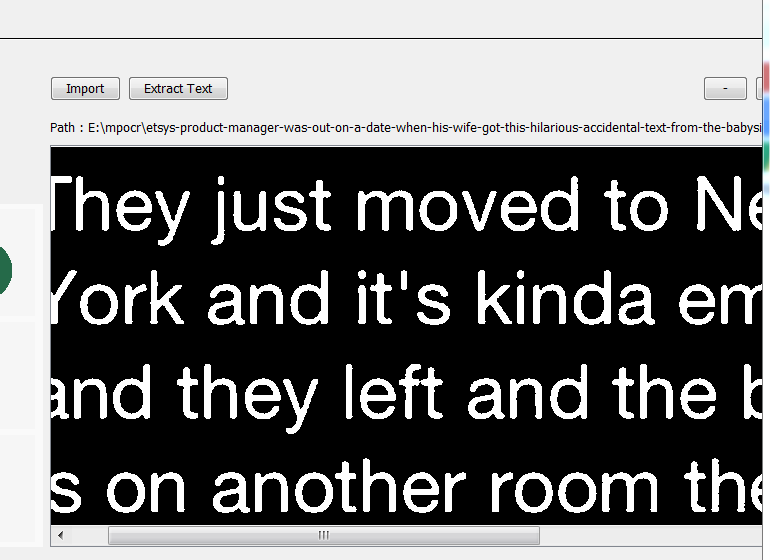
**The algorithm approach behind Otsu**

Whoa too much math. Lets simplify this. For each potential threshold T we:

* separate the pixels into two clusters according to the threshold
* find the mean of each cluster and square the difference between means
* multiply by the number of pixels in one cluster times the number in the other

If we incorporate the math from the previous section we get:

* compute histogram and probabilities of each intensity level
* set up initial q_i(0) and \mu_i(0)
* step through all possible threshold maximum intensity
* update q_i and \mu_i and compute \sigma_b^2(t)
* desired threshold corresponds to the maximum

****

*Fig 2a. Image before binarization. Fig 2b. Image after binarization*

* **Size Normalization** The feature we tried were zone-densities, Intersections, Raw-Pixels. These features need the image to be in a fixed size hence an operation called size normalization is applied to image. To normalize the images we simply scaled each pixel from source image to required sized image.

**2.2 Segmentation.**

As apparent as it is, the whole image comprising of lots of characters cannot be used for recognition. Hence the need to separate each character arises. One easy method is to use find the band comprising background pixels run along vertically to identify words as well as characters. Lines can be separated, by finding such a band which runs horizontally.

Each segment at the end contains pixel information of only a single character. Segment is a regular bounded square right about the end of a stroke of a character on all fronts i.e. to right, left, top and bottom. These segments are then used to extract relevant features. Necessary data structure is used to store the segments so as to access to each segment is in order, and information remain persistent.

**Steps we applied in segmentation**

* Iterate the image horizontally and vertically.
* Find the row or column with specified number of foreground pixels which serve as extreme edges of the segment.
* If the row has no foreground pixel, proceed to next row.
* Save the segment.
* Scan through all the image and store all the encountered segments in a list.

Below is an example of segmentation we have done.



Figure 3a. Original images

****

****

****

****

*Figure 3b. Segments generated*

**2.3 Feature Extraction**

Extracting meaningful information from segments that help in discriminating one character from another is a pivotal process in character recognition systems. We need to differentiate what information helps to define a character, and how to extract those. Once the features have been identified, they define characters. Some basic features include crossing, zonal features, pixel density, raw pixels etc.

The features we tried were pixel density per zone, intersections with a defined grid & raw pixels. The segments that are formed, are normalized/resized to fit a standard size of 10x10. While normalizing, utmost care is taken to ensure the feature of each character remain intact. This normalized image is the foremost feature that we have considered. Although a feature vector for a single character consists of 100 values, it poses no problem for processing for the neural network. This feature is sufficient enough to distinguish English characters and digits with an accuracy of over 90% according to [15].

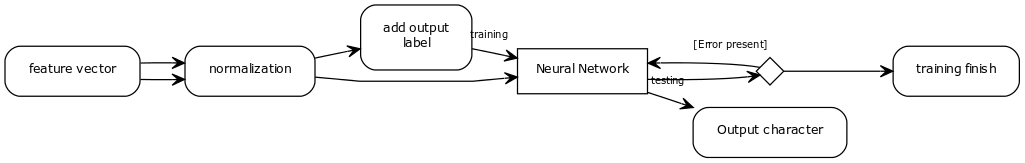
Below is the example of an image and its feature.

Image data Feature Vector

<1,0,1,0,1,1,1,1,1,1,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1>

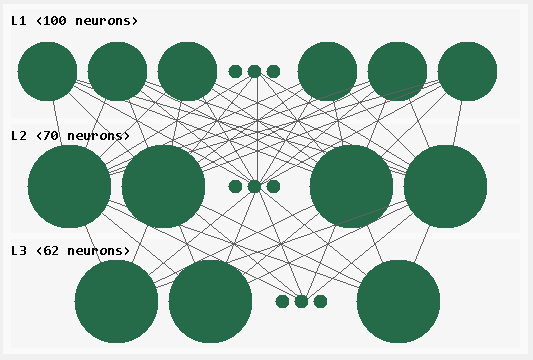
**2.4 Training**

The Neural Network is trained for recognizing the characters in this phase. This phase further contains the phases shown in the image below and are explained in following sections. The features and a label extracted from feature-extraction phase are input to the Neural Network and the output, cost, error is calculated and network is corrected according to the cost. Below is the sequence diagram of training. The training is continued until some defined maxIterations complete or the network reaches the defined maxError.

****

*Fig 4. Activity diagram of the training phase.*

The Neural network designed is a multilayer feed-forward network with learning algorithm as back-propagation. For error correction it uses Gradient Descent. Below is the diagram of the Neural-Network that we used. It has 3 layers where input layer has 100 neurons and output layer has 62 neurons. There is only one hidden layer which has 70 neurons. The backpropagation algorithm is discussed in further chapters.

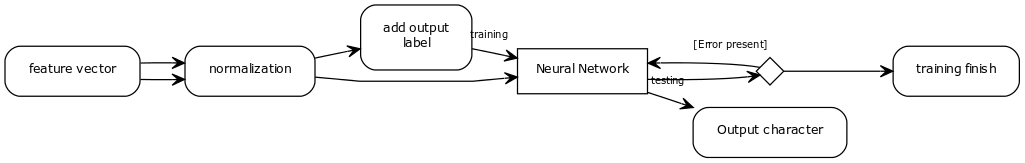
****

*Figure 5. Neural Network structure.*

**2.5 Testing/Working phase**

In this phase the Neural Network is used to recognize the characters and is verified for correctness. This helps in in testing the implementation and making the recognizer better. This involves some of the above steps in learning phase but with features with no labels while inputting the feature vector (We don’t know it at all!). We also don’t correct the network according to the cost as we do in learning phase.

The set of images which are used to test the OCR is called “Testing set” and it consists of various types of images that are needed to be recognized. It is similar to training set but we do not know the output i.e. the label is not provided in this case. It uses the memory model built in training done previously to recognize the input.

****

*Figure 6. Activity Diagram for recognition Process*

**CHAPTER 3. ARCHITECTURE**

We used layers pattern to solve the problem. The of design of classes such as OImage, Segment, Segmentation, Feature, FeatureSet were straightforward so we discuss the NeuralNetwork class design here. The class diagram for the application can be found below.

**NeuralNetwork class.** This class composes of the attributes of the NeuralNetwork (refer class diagram) and it provides methods to train and compute the output of network. It is the object which manages the **Layer**sand passes the activations and error between them.

**Layer class.** This class holds the weights as well as **Neurons** of this layer and has methods to pass the activations input from the **NeuralNetwork** to the **Neuron**s.

**Neuron class**. This class is the core of **NeuralNetwork**. It computes the activation by taking inputs from the **Layer** class.

IFeatureSet<<interface>>

+getFeatures():Features

+getMagic():int

Features

add(i:IFeatureSet):void

get(magicNumber:int):IFeatureSet

SigmoidFunction

+derivative():double

+fire():double

ActivationFunction

+derivative():double

+fire():double

Neurons

-activation:double

+getActivation():double

+getBias():double

Layer

+computeError():void

+getError():double [ ]

+process():void

NeuralNetwork

+layersCount():int

+getLayer(index:int):Layer

+getEpochSize():int

+getCost():double

INeuralNetwork<<interface>>

+layersCount():int

+getLayer(index:int):Layer

Segmentation

iData: OImage

segmentImage():void

Segment

iData:int [ ] [ ]

getBounded():int

printImage():void

extractFeatures():void

OImage

Threshold():int

Binarize(BasicImage):void

ImageHistogram:int []

readImage(path:String):

void

BasicImage

iData:int [ ] [ ]

+getHeight():int

+getWidth():int

+getImageData():int [ ][ ]

+getForeground():int

IImage<<interface>>

+getHeight():int

+getWidth():int

+getImageData():int [ ][ ]

*Figure 7. Class diagram*

**CHAPTER 4. USER-INTERFACE**

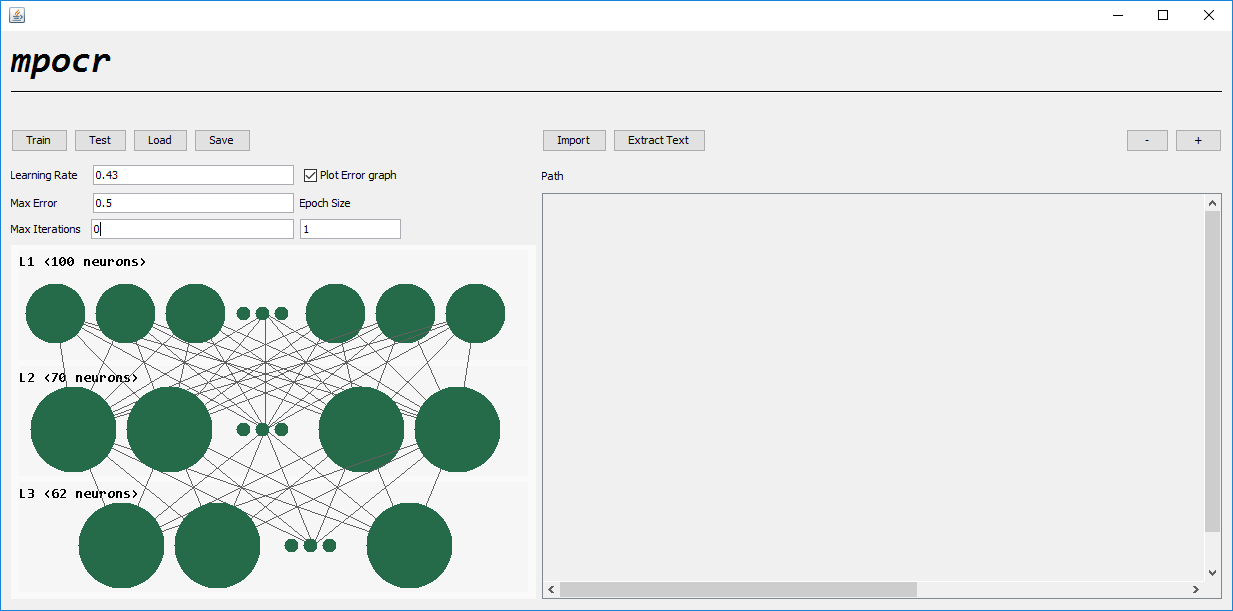
****

Figure 8. The main UI

The entire UI is formed using the JPanels. It provides functional buttons as well as some customizable metric to get different results. It also consists of a canvas.

1. Train: On click of this button, user is prompted to choose training set and the same is used to train the neural network.

2. Test: On click of this button, user is prompted to choose testing set and the same is used to test the neural network.

3. Load: This loads the previously stored training values into the neural network. This is to avoid training of neural network, every time we want to recognize characters.

4. Save: On click of this button, the current state of the Neural-Network is saved. This can be loaded next time using load button.

5. Import: On click of this button, user is prompted to choose an image for recognition.

Additional parameters like Learning rate, epoch size and iteration times are changeable, this makes training and recognition of characters simpler dynamically.

**4.1 Input Image for recognition**

Provision is made for the user to input any image containing text characters. Note that the text must be only consisting of English Characters and digits from 0-9. There is inherent limitation to how noisy the input image can be, due to non-advanced pre-processing softening of the noisy image. The image format can be any valid image format such as .jpg, .png, .bmp etc.

To input an image, click on **Import** button. Navigate to the file that you want to convert and select the file. After an image is selected, the image is displayed on the canvas. Now click **Extract** button to recognize the characters in the image. The extracted text is stored in output.html file located in same directory as executable.

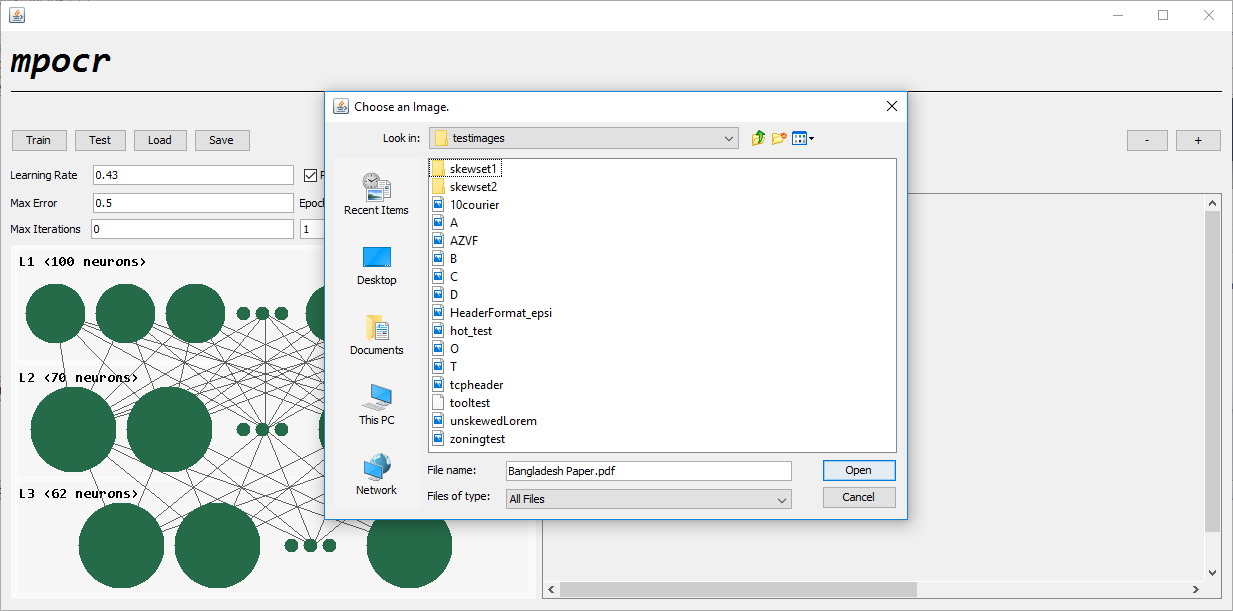


Figure 9a. Importing the image.

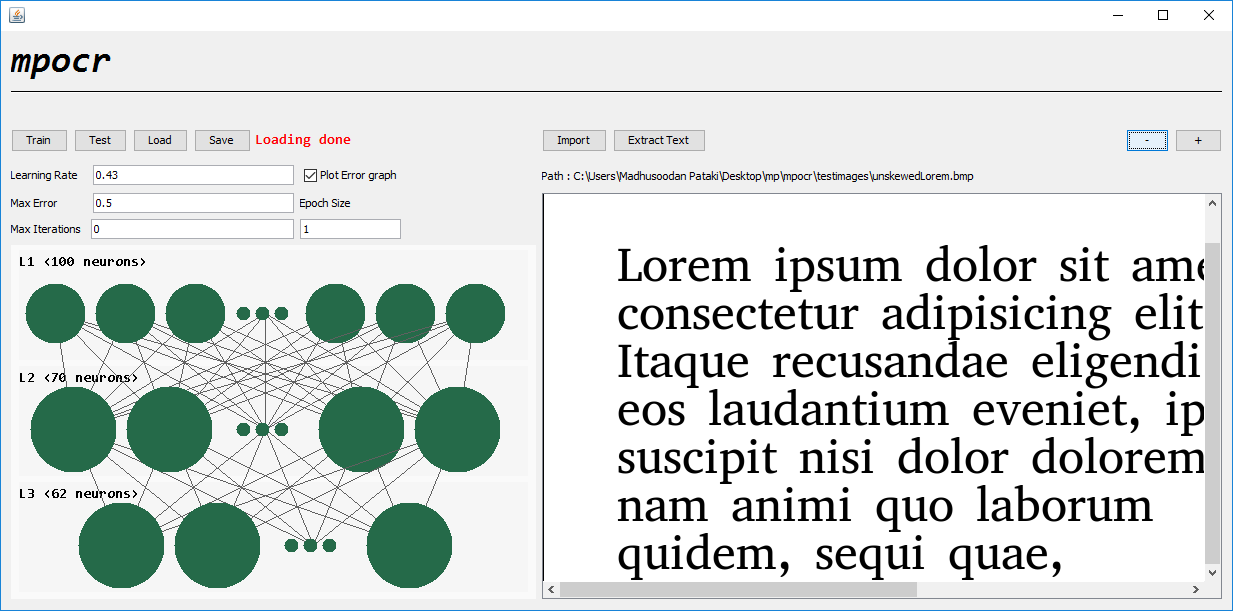


Figure 9b. Imported image on canvas.

**4.2 Training.**

To make a provision for user to change, update the training set, this functionality is added. On clicking the train button, a file chooser/browser will open. This will prompt the user to choose a folder that encompasses all the images for training. **Note** that all the necessary images must be grouped together inside a single folder. The images will be imported and used for training the neural network. The user can also specify learning rate, maxIterations, maxError and additional options such as whether he wants to see the error graph of training which are explained below.

* **Learning Rate:** It is a constant which indicates the amount by which, the Neural network adjusts the cost function. Here depending on the error propagated, we need to select the learning rate. Higher the learning rate , grater is the difference between two consecutive errors. We have determined that we get best results with learning rate kept between 0.42-0.55.
* **Max Error:** It is the maximum allowed error before the training halts. It can also be said as the minimum threshold, which the errors should reside low. Lower the error, higher is the accuracy of the Neural-Network.
* **Number of Iteration:** It is the number of times the error gets propagated in the neural network. Higher the no. of iterations, higher is the possibility of getting lesser error. Keeping it too high will train the network for a long duration. Recommended to keep it around 8000.
* **Epoch Size:** It is the number of cycles after which the error gets propagated through the network. Keeping it low, means error is propagated as frequent as possible and higher is the updation rate.

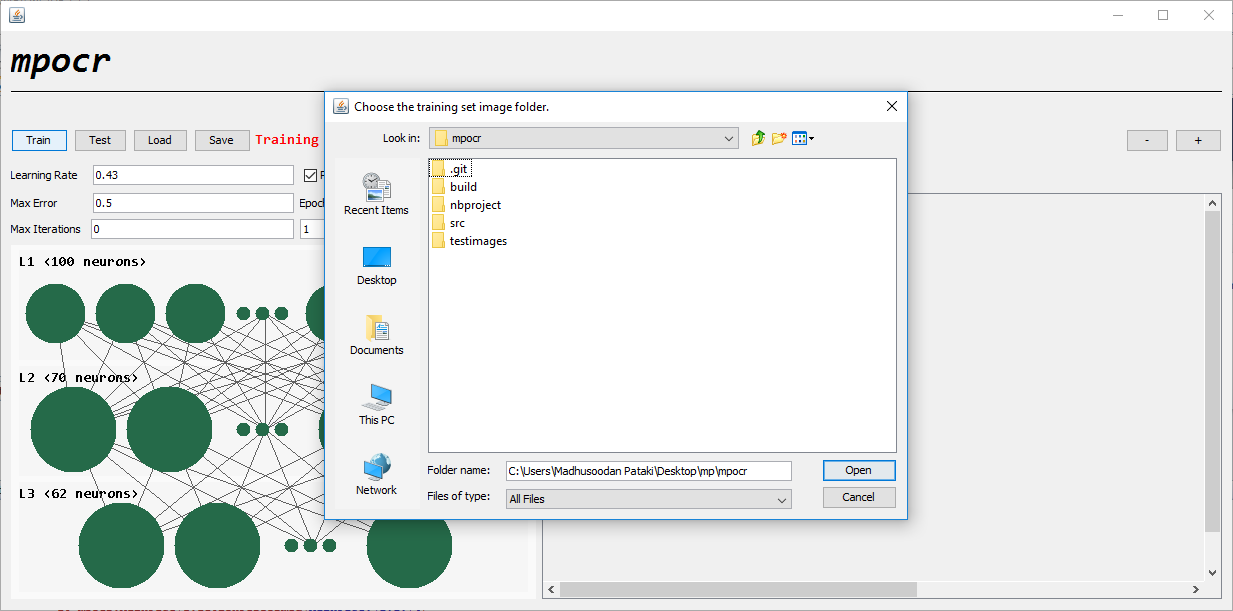
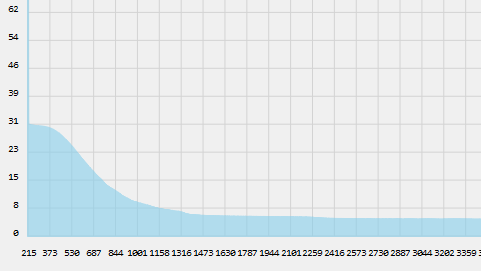


Figure 10. Importing Training set.

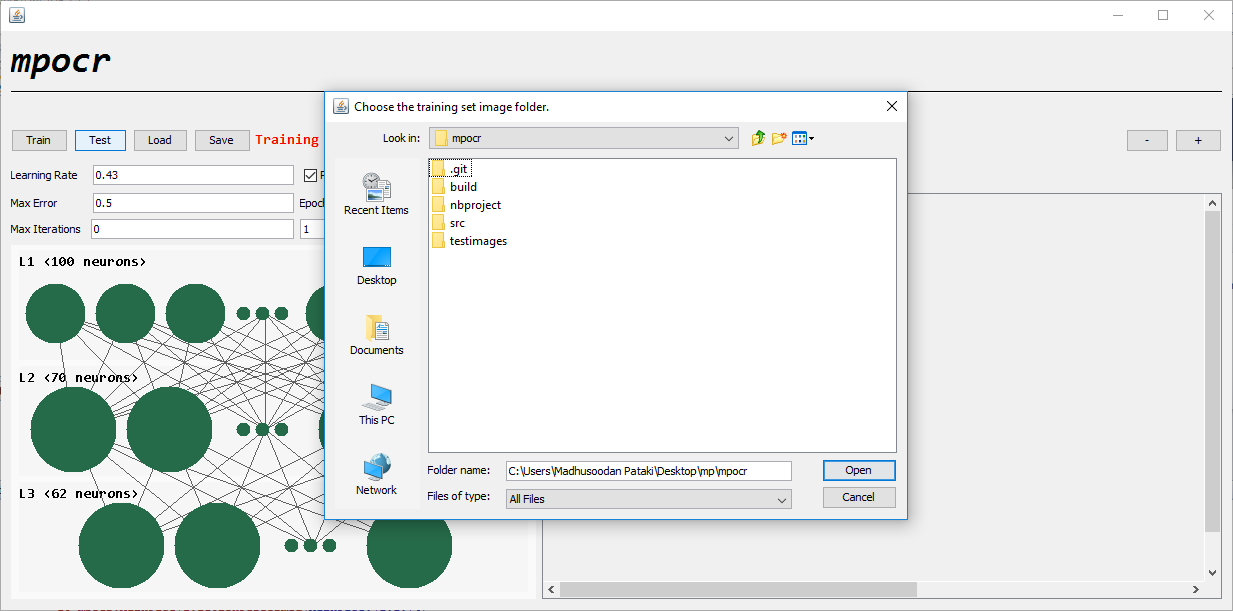
If the user has choosed to see the error plot the chart is stored in the same directory as the executable as an html file which has to opened in the web browser which may look like.

****

*Figure 11. Error graph.*

**4.3 Testing.**

This is a functionality that enables the user to verify to what extent the previous training has been successful. On-click of this button, a File chooser/browser will open to take an input image set that act as a testing set. **Note** that this is a different case than actual extraction. In this case the input images are taken along with the output label. Although the output label is not used to train the network, it helps to verify the accuracy of the software.

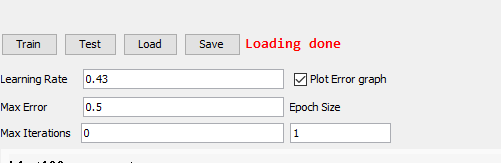


*Figure 12. Importing testing set.*

**4.4 Load and Save**

We can save the current state of Neural-Network to get rid of training the network every time we boot up. Just click on the “Save” and “Load” buttons.

Note we can only store one instance of the network from our GUI while the user can keep a copy of the saved network file if he wants to which is located in same path as executable is.



*Figure 13. Loading and saving the Neural Network*

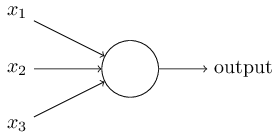
On click of load, there appears a message "Loading done" on successful load else it shows error message on not being successful to load previous training.

After training, on-click of "Save" the training result will be stored as an object, on a default path. This helps to overcome the necessity of training on each occasion.

**CHAPTER 5. ABOUT NEURAL NETWORKS**

The Neural Networks are mathematical model mimicking the human brain. It has many layers consisting of neurons and which take input from previous layer and output its activation which is the output of transfer function with input set as the sum. The main impetus for neural network is to take a large number of input samples with expected output, known as training examples, then develop a system that can learn from those training examples. In other words, the neural network uses the examples to automatically infer rules for recognizing alphabets and digits in our case. Furthermore, by increasing the number of training examples, the network can learn more, and so improve its accuracy.

Sigmoid neuron is the type of neuron used in out Neural-Network which means the transfer function used is “Sigmoid function”.



*Figure 14. Neuron structure*

A sigmoid neuron can take n inputs and result a single output which ranges between 0 and 1. The output is given based on the weights associated with each input as well as the tendency/bias induced towards an output.

‘w’ is the weight associated with a single input.

‘b’ is the bias of the neuron.

Output = 0 if w⋅x + b ≤ 0

* 1. if w⋅x + b > 0 ….[1]

In case of a Neural-Network consisting of multiple sigmoid neurons, we can write down [1] as

output = 0 if  ∑≤ threshold

1 if  ∑> threshold

One advantage of using sigmoid neurons is that it has smooth curve and output of it is continuous. Sigmoid neuron alters output in small margin if there is a small change in weight or bias. The change is associated with a function called sigmoid function denoted by.

σ(z)=

**The format of expected output.** We assigned some numbers 0-61 to each character we want to recognize which we know by the file name (refer Tools and sub-processes chapter) at the time we extract features. We wanted our neural network to raise the **n**th entry in the output vector if the expected output was **n**.

**Learning with Back-propagation and Gradient Descent**

The amount of change in the output is called the cost. For a small change in weights and biases, the cost changes. This change in cost is associated with a function called *The Cost Function*. Through this cost function, the network can learn the weights and biases needed to produce a required output. The Cost function used for our Neural-Network is:

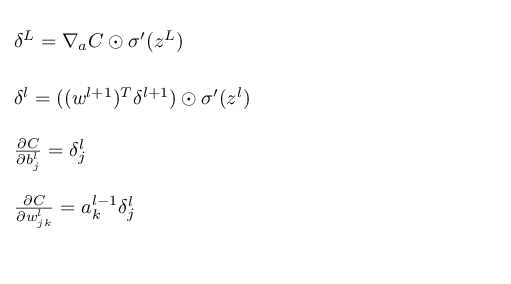
Here, ‘C’ is the Cost, w and b are the weights and biases for the input, y(x) is output generated for the given input and ‘a’ is the expected output vector. We used the quadratic form because it is always positive and negative cost may give a false impression of less cost when it summed for each output layer neuron.

We try to minimize the change in cost, as far as possible. Change in cost ∆C should be a negative number so as to reduce C+ ∆C value. As the ∆C varies, the network learns about the new weights and biases for that Cost, and as once C ≈ maxError, we stop training.

**Back Propagation**

The goal of back-propagation is to compute the change in weight and bias, with respect to change in cost separately. i.e. how much change in weight does it contribute to overall change in cost.

The Equations of backpropagation:



During training, we know the expected output. Once the resulting output is calculated based on initial random weights, the error at the output neuron is calculated. The backpropagation algorithm, propagates the error from the output neuron to backward layers. This error is the cost change for gradient descent algorithm and then adjusts the weights accordingly. Till the error attains a maxError we train the network.

**Results and conclusion**

The recognition of character primarily depends on the image quality of the input image. That said, we have observed that by increasing the time for learning, we are able to effectively reduce the error to some extent. This is the most important aspect to increase the accuracy of the neural network.

Below are the some statistics from our testing of the Neural network.

The neural network was trained with 40x40 images.

1. Now considering the best case (91.93% accuracy) we tested the Neural Network for different kind of test images where we got accuracies as follows.

|  |  |  |  |
| --- | --- | --- | --- |
| Training Images | Testing Images | Testing Image sizes (px) | Accuracy (%) |
| 62 (40x40) | 186 | 20x20, 40x40, 80x40 | 73.65 |
| 62 (40x40) | 124 | 20x20, 80x80 | 65.32% |
| 62 (40x40) | 62 | 20x20 | 54.83% |
| 62 (40x40) | 62 | 40x40 | 78.80% |

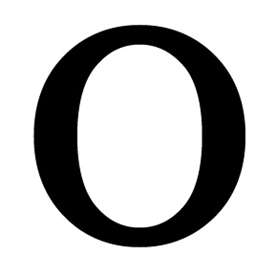
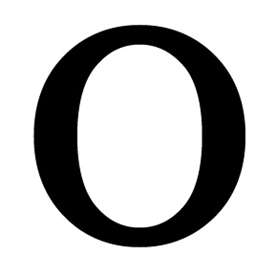
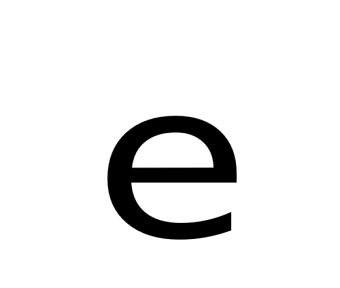
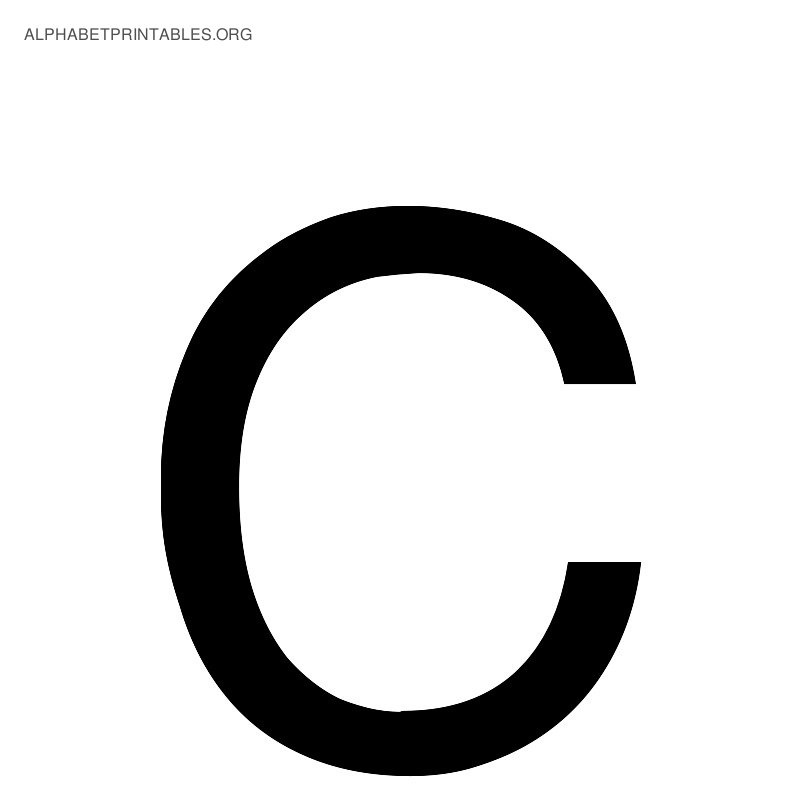
1. Below are the accuracies reported from our Neural-Network for different learning-rates.

|  |  |  |
| --- | --- | --- |
| **Learning Rate** | **Max Error** | **Accuracy** |
| 0.43 | 3.05 | 90.12 |
| 0.53 | 5.511 | 82.25 |
| 0.48 | 4.019 | 87.09 |
| 0.38 | 3.021 | 90.32 |
| 0.33 | 2.524 | 91.93 |
| 0.28 | 3.030 | 90.32 |
| 0.23 | 5.040 | 83.87 |

**CHAPTER 6. SOME KNOWN LOOP-HOLES AND IMPROVEMENTS**

**6.1 Loop holes**

* Distinguishing characters with same morphology for example 'e' from 'c', 'O' from '0' etc. This happens because when we normalize the image the characters with same morphology but of different size look almost same hence it takes a lot of training to distinguish them.



*Figure 15. Incorrectly recognized characters*

Here the latter is a much harder problem to distinguish between 'O' and '0', as distribution, loops and curvature are more similar than different. An Artificial Neural Network is used as classifier in this work since we find it easy to implement.

**6.2 Some improvements**

* The network’s output can be divided into words and checking whether the words exists in dictionary or not can give us a chance to check other outputs of the neural network too.

**CHAPTER 7. ABOUT SOME SUB-PROCESSES, TOOLS**

**7.1 Platform**

Java is the choice of platform for language. Reason being it is cross-platform hence no issue of compatibility, readily available libraries to read various types of Image formats. Java also provides some widgets for the GUI with little effort. The main reason we use Java as default platform is due to its Object-Oriented approach. It provides a middle ground to both higher level abstraction as well as low level mathematical processing involved in this project. This is an ideal be-all solution for the scale and domain of our problem. Object-Oriented nature also helps us to build our own customized objects from scratch without any use of external libraries. And finally we're familiar with Java.

**7.2 Tools and framework**

For testing the correctness of the Neural Network, we used Neuroph, a neural network library for Java. Neuroph is an open source library that simulates a Neural-Network. It simulates various types of neural-networks.

We also used a tool called “ocrimgen” to generate training and testing set of images.

**7.3 Training set generation**

Here the different types of input images are collected and are labeled with the expected output. We store these images in folders named by label. The reason for many images is as the number of images increase more will the variation in input and learning will be efficient and recognition accuracy increases. The criteria for collection was simple but different font-family and sizes.

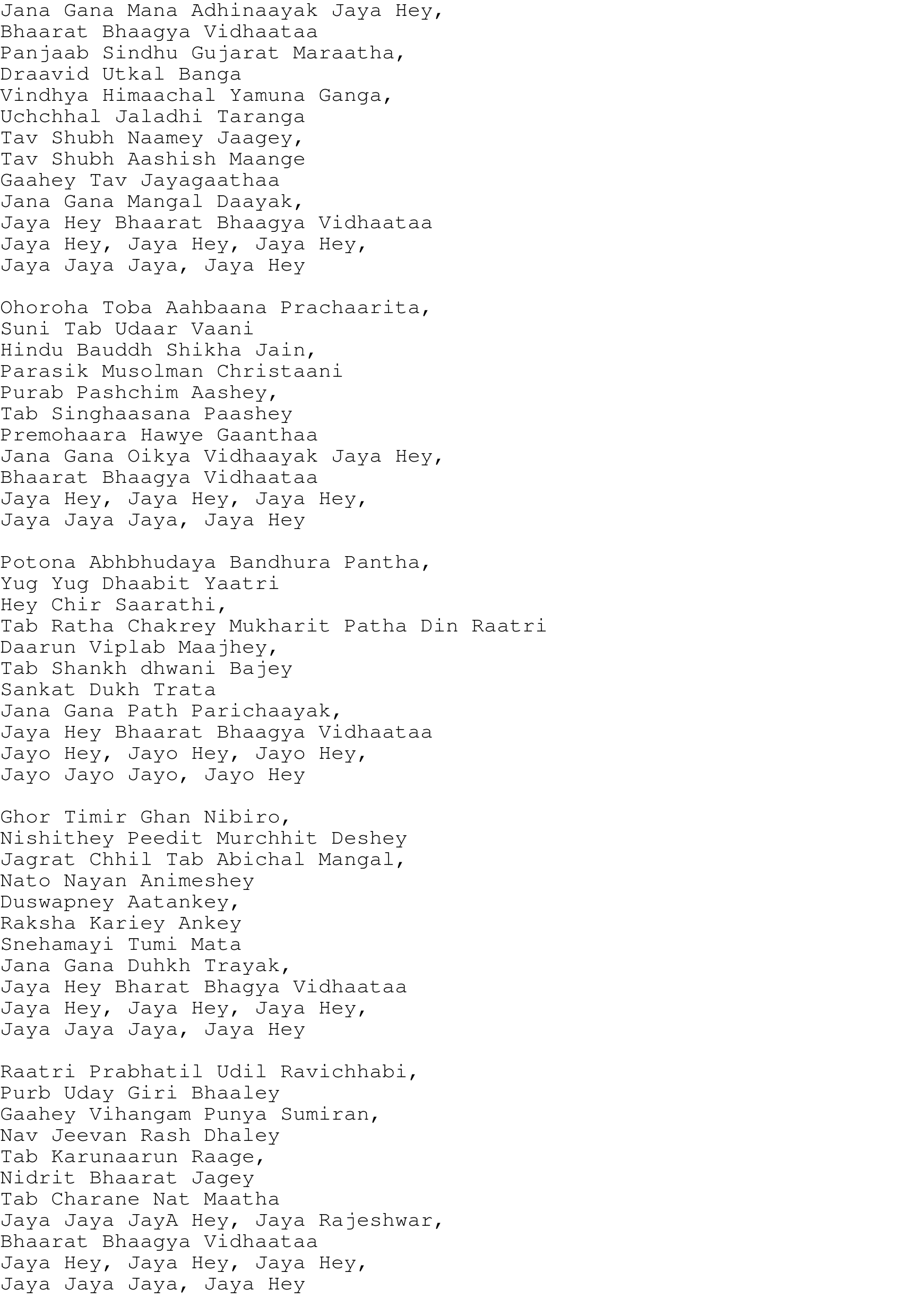
We generate the training data with the help of “ocrimgen”, which generates 62 different characters of the English language, along with digits and special characters. These 62 characters are generated with 12 different varying sizes and fonts. The tool also provides functionality to generate text images (multiple characters) .These generated set is used for training the neural network and also different set of data is generated for testing.

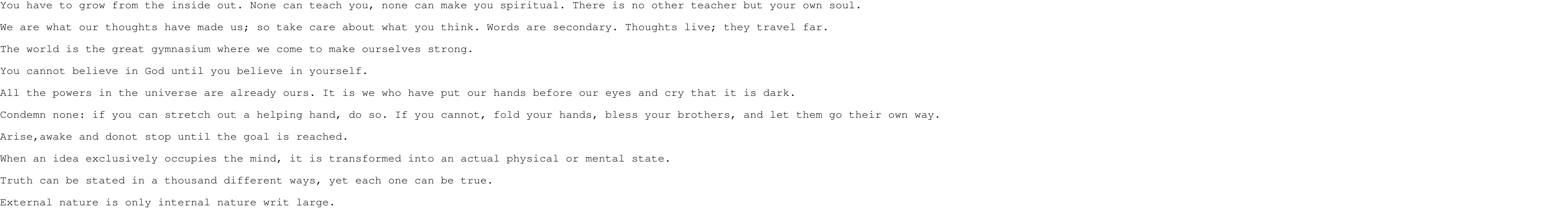
**SOME SAMPLES FROM TRAINING AND TESTING SET**

**Training set images**

****

**Testing images.**

****

****

**References**

*[1]. “Pattern Recognition and Machine Learning” by Cristopher M. Bishop. ISBN-13:978-0387-31073-2 Springer Science and Business Media LLC.*

*[2]. Seethalakshmi R et al. “Optical Character Recognition for printed Tamil text using Unicode”. Journal of Zhejiang University SCIENCE ISSN 1009-3095*

*[3]. Usha Tiwari et al. “Text Extraction from Images”. International Journal of Electronic and Electrical Engineering. ISSN 0974-2174Volume 7, Number 9 (2014), pp. 979-985*

*[4]. Sandhya Arora et al. “Combining Multiple Feature Extraction Techniques for Handwritten Devanagari Character Recognition” 2008 IEEE Region 10 Colloquium and the Third ICIIS, Kharagpur, India*

*[5]. Gaurav Y. Tawde, and Jayashree M. K. International Journal of Engineering Research and Applications (IJERA) ISSN: 2248-9622*

*[6]. Rafael C. Gonzalez, Richard E. woods and Steven L. Eddins. Digital Image Processing, Pearson Education, Dorling Kindersley, South Asia, 2004.*

*[7]. Anil.K.Jain and Torfinn Taxt, “Feature extraction methods for character recognition-A Survey” Pattern Recognition, vol. 29, no. 4, pp. 641-662, 1996*

*[8]. C. L. Liu, H. Fujisawa, “Classification and Learning for Character Recognition : Comparison of Methods and Remaining Problems”, Int. Workshop on Neural Networks and Learning in Document Analysis and Recognition, Seoul, 2005.*

*[9]. J Pradeep et al. "Diagonal based feature extraction for handwritten characters", International Journal of Computer Science & Information Technology (IJCSIT), Vol 3, No 1, Feb 2011*

*[10]. Dinesh Dileep et al. "A feature extraction technique based on character geometry for character recognition", International Journal of Advanced Research in Computer Science and Software Engineering, Vol. 2,2014.*

*[11]. M. Zahid Hossain et al. “Rapid Feature Extraction for Optical Character Recognition”*

*[12]* [*www.neuralnetworksanddeeplearning.com*](http://www.neuralnetworksanddeeplearning.com/)*. ,Michael Nielson.*