Handwritten Character Recognition using Convolutional Neural Networks in Python with Keras

Hanu Priya Indiran, *Student Member - IEEE, Bachelors in Electronics and Communication Engineering*
Kumaraguru College of Technology, Coimbatore, Tamil Nadu, India.
hanupriya28@gmail.com

Abstract— In the field of Deep Learning for Computer Vision, scientists have made many enhancements that helped a lot in the development of millions of smart devices. On the other hand, scientists brought a revolutionary change in the field of image processing and one of the biggest challenges in it is to identify documents in both printed as well as hand-written formats. One of the most widely used techniques for the validity of these types of documents is 'Character Recognition'. This project seeks to classify an individual handwritten word so that handwritten text can be translated to a digital form. It demonstrates the use of neural networks for developing a system that can recognize handwritten English alphabets. In this system, each English alphabet is represented by binary values that are used as input to a simple feature extraction system, whose output is fed to our neural network system. The CNN approach is used to accomplish this task: classifying words directly and character segmentation. For the former, Convolutional Neural Network (CNN) is used with various architectures to train a model that can accurately classify words. For the latter, Long Short Term Memory networks are used with convolution to construct bounding boxes for each character. We then pass the segmented characters to a CNN for classification, and then reconstruct each word according to the results of classification and segmentation.

Keywords : Computer Vision , CNN, Character Recognition, Classification, Deep Learning, Neural Networks

I. INTRODUCTION

Handwritten character recognition is a field of research in artificial intelligence, computer vision, and pattern recognition. A computer performing handwriting recognition is said to be able to acquire and detect characters in paper documents, pictures, touch-screen devices and other sources and convert them into machine-encoded form. Its application is found in optical character recognition and more advanced intelligent character recognition systems. Most of these systems nowadays implement machine learning mechanisms such as neural networks. Machine learning is a branch of artificial intelligence inspired by psychology and biology that deals with learning from a set of data and can be applied to solve wide spectrum of problems. A supervised machine learning model is given instances of data specific to a problem domain and an answer that solves the problem for each instance. When learning is complete, the model is able not only to provide answers to the data it has learned on, but also to yet unseen data with high precision. Neural networks are learning models used in machine learning. Their aim is to simulate the learning process that occurs in an animal or human neural system. Being one of the most powerful learning models, they are useful in automation of tasks where the decision of a human being takes too long, or is imprecise. A neural network can be very fast at delivering results and may detect connections between seen instances of data that humans cannot see. Having acquired the knowledge that is explained in this text, the neural network has been implemented on a low level without using libraries that already facilitate the process. By doing this, we evaluate the performance of neural networks in the given problem and provide source code for the network that can be used to solve many different classification problems. A small step towards this goal is explored in this work by training a neural network model to learn which parts of an image are interesting to human observers that search for a specific object. This knowledge can then be used to speed up object search in computer vision.

Adopting the principle of convolution to neural networks led to convolutional neural networks. The first driving force behind handwritten text classification was for digit classification for postal mail. Jacob Rabinowitz early postal readers incorporated scanning equipment and hardwired logic to recognize monospaced fonts [1]. Allum et. al improved this by making a sophisticated scanner which allowed for more variations in how the text was written as well as encoding the information onto a barcode that was printed directly on the letter [2].The first prominent piece of OCR software was invented by Ray Kurzweil in 1974 as the software allowed for recognition for any font [3]. This software used a more developed use of the matrix method (pattern matching). Essentially, this would compare bitmaps of the template character with the bitmaps of the read character and would compare them to determine which character it most closely matched with. The downside was this software was sensitive to variations in sizing and the distinctions between each individual's way of writing

II. PROBLEM IDENTIFICATION AND APPROACH

Despite the abundance of technological writing tools, many people still choose to take their notes traditionally: with pen and paper. However, there are drawbacks to handwriting text. It's difficult to store and access physical documents in an efficient manner, search through them efficiently and to share them with others. Thus, a lot of important knowledge gets lost or does not get reviewed because of the fact that documents never get transferred to digital format. We have thus decided to tackle this problem in our project because we believe the significantly greater ease of management of digital text compared to written text will help people more effectively access, search, share, and analyze their records, while still allowing them to use their preferred writing method. The aim of this project is to further explore the task of classifying handwritten text and to convert handwritten text into the digital format.

Handwritten text is a very general term, and we wanted to narrow down the scope of the project by specifying the meaning of handwritten text for our purposes. In this project, we took on the challenge of classifying the image of any handwritten word, which might be of the form of cursive or block writing. This project can be combined with algorithms that segment the word images in a given line image, which can in turn be combined with algorithms that segment the line images in a given image of a whole handwritten page. With these added layers, our project can take the form of a deliverable that would be used by an end user, and would be a fully functional model that would help the user solve the problem of converting handwritten documents into digital format, by prompting the user to take a picture of a page of notes. Note that even though there needs to be some added layers on top of our model to create a fully functional deliverable for an end user, I believe that the most interesting and challenging part of this problem is the classification part, which is why we decided to tackle that using the Convolutional Neural Networks. I approach this problem with complete handwritten alphabet images because CNN's tend to work better on raw input pixels rather than features or parts of an image [4]. Given our findings using handwritten alphabets, the program soughts improvement by extracting characters from the handwritten image and then classifying each character independently to reconstruct the digital letter. In summary, in both of our techniques, our models take in an image of an alphabet which is handwritten and output the alphabet digitally.

Two phase processes are involved in the overall processing of our proposed scheme: the Pre-processing and Neural network based Recognizing tasks. The pre-processing steps handle the manipulations necessary for the preparation of the characters for feeding as input to the neural network system. First, the required character or part of characters needs to be extracted from the pictorial representation. The splitting of alphabets into 25 segment grids, scaling the segments so split to a standard size and thinning the resultant character segments to obtain skeletal patterns. The following pre-processing steps may also be required to furnish the recognition process:

A.. The alphabets can be thinned and their skeletons obtained using well-known image processing techniques, before extracting their binary forms.

B. The scanned documents can be “cleaned” and “smoothed” with the help of image processing techniques for better performance.

III. Deep Learning

Deep Learning is an application of artificial intelligence (AI) that provides systems the ability to automatically learn and improve from experience without being explicitly programmed. Deep learning methods aim at learning feature hierarchies with features from higher levels of the hierarchy formed by the composition of lower level features. Automatically learning features at multiple levels of abstraction allow a system to learn complex functions mapping the input to the output directly from data, without depending completely on human-crafted features. Deep learning focuses on the development of computer programs that can access data and use it to learn for themselves. The process of learning begins with observations or data, such as examples, direct experience, or instruction, in order to look for patterns in data and make better decisions in the future based on the examples that we provide. The primary aim is to allow the computers to learn automatically without human intervention or assistance and adjust actions accordingly. The algorithms are often categorized as supervised or unsupervised.

(i) Supervised Learning Algorithm

This algorithm can apply what has been learned in the past to new data using labeled examples to predict future events. Starting from the analysis of a known training dataset, the learning algorithm produces an inferred function to make predictions about the output values. The system is able to provide targets for any new input after sufficient training. The learning algorithm can also compare its output with the correct, intended output and find errors in order to modify the model accordingly.

(ii) Unsupervised Learning Algorithm

In contrast, unsupervised machine learning algorithms are used when the information used to train is neither classified nor labeled. Unsupervised learning studies how systems can infer a function to describe a hidden structure from unlabeled data. The system doesn’t figure out the right output, but it explores the data and can draw inferences from datasets to describe hidden structures from unlabeled data.

Semi-supervised machine learning algorithms fall somewhere in between supervised and unsupervised learning, since they use both labeled and unlabeled data for training – typically a small amount of labeled data and a large amount of unlabeled data. The systems that use this method are able to considerably improve learning accuracy. Usually, semi-supervised learning is chosen when the acquired labeled data requires skilled and relevant resources in order to train it or learn from it. Otherwise, acquiring unlabeled data generally doesn’t require additional resources.

(iii) Reinforcement Learning Algorithm

It is a learning method that interacts with its environment by producing actions and discovers errors or rewards. Trial and error search and delayed reward are the most relevant characteristics of reinforcement learning. This method allows machines and software agents to automatically determine the ideal behavior within a specific context in order to maximize its performance. Simple reward feedback is required for the agent to learn which action is best; this is known as the reinforcement signal.

Machine learning enables analysis of massive quantities of data. While it generally delivers faster, more accurate results in order to identify profitable opportunities or dangerous risks, it may also require additional time and resources to train it properly. Combining machine learning with AI and cognitive technologies can make it even more effective in processing large volumes of information.

IV. NEURAL NETWORK

The simplest definition of a neural network, more properly referred to as an 'artificial' neural network (ANN), is provided by the inventor of one of the first neurocomputers, Dr. Robert Hecht-Nielsen. He defines a neural network as*"...a computing system made up of a number of simple, highly interconnected processing elements, which process information by their dynamic state response to external inputs.* In "Neural Network Primer: Part I" by Maureen Caudill, AI Expert, Feb. 1989. ANNs are processing devices (algorithms or actual hardware) that are loosely modeled after the neuronal structure of the mammalian cerebral cortex but on much smaller scales. A large ANN might have hundreds or thousands of processor units, whereas a mammalian brain has billions of neurons with a corresponding increase in magnitude of their overall interaction and emergent behavior. Although ANN researchers are generally not concerned with whether their networks accurately resemble biological systems, some have. For example, researchers have accurately simulated the function of the retina and modeled the eye rather well. Although the mathematics involved with neural networking is not a trivial matter, a user can rather easily gain at least an operational understanding of their structure and function. Neural networks are typically organized in layers. Layers are made up of a number of interconnected 'nodes' which contain an 'activation function'. Patterns are presented to the network via the 'input layer', which communicates to one or more 'hidden layers' where the actual processing is done via a system of weighted 'connections'.

Most ANNs contain some form of 'learning rule' which modifies the weights of the connections according to the input patterns that it is presented with. In a sense, ANNs learn by example as do their biological counterparts; a child learns to recognize dogs from examples of dogs. Although there are many different kinds of learning rules used by neural networks, this demonstration is concerned only with one; the delta rule. The delta rule is often utilized by the most common class of ANNs called 'backpropagation neural networks' (BPNNs). Backpropagation is an abbreviation for the backwards propagation of error. With the delta rule, as with other types of backpropagation, 'learning' is a supervised process that occurs with each cycle or 'epoch' (i.e. each time the network is presented with a new input pattern) through a forward activation flow of outputs, and the backwards error propagation of weight adjustments. The process flow of a neural network is as per the figure below.

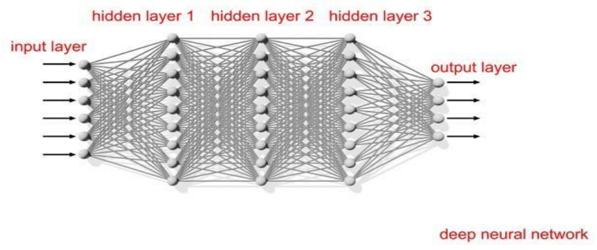


Figure 4.a

Neural networks take a different approach to problem solving than that of conventional computers. Conventional computers use an algorithmic approach i.e. the computer follows a set of instructions in order to solve a problem.

Unless the specific steps that the computer needs to follow are known the computer cannot solve the problem. That restricts the problem solving capability of conventional computers to problems that we already understand and know how to solve. But computers would be so much more useful if they could do things that we don't exactly know how to do. Neural networks process information in a similar way the human brain does. The network is composed of a large number of highly interconnected processing elements(neurones) working in parallel to solve a specific problem. Neural networks learn by example. They cannot be programmed to perform a specific task. The examples must be selected carefully otherwise useful time is wasted or even worse the network might be functioning incorrectly. The disadvantage is that because the network finds out how to solve the problem by itself, its operation can be unpredictable.

Neural networks and conventional algorithmic computers are not in competition but complement each other. These tasks are more suited to an algorithmic approach like arithmetic operations and tasks that are more suited to neural networks. Even more, a large number of tasks require systems that use a combination of the two approaches (normally a conventional computer is used to supervise the neural network) in order to perform at maximum efficiency. So this project involves the Convolutional Neural Networks to analyse the problem and provide the appropriate solution.

V. MODEL APPROACH

A neural network is made up of neurons connected to each other; at the same time, each connection of our neural network is associated with a weight that dictates the importance of this relationship in the neuron when multiplied by the input value. Each neuron has an activation function that defines the output of the neuron. The activation function is used to introduce non-linearity in the modeling capabilities of the network. We have several options for activation functions that we will present in this post. Training our neural network, that is, learning the values of our parameters (weights wij and bj biases) is the most genuine part of Deep Learning and we can see this learning process in a neural network as an iterative process of "going and returning" by the layers of neurons. The "going" is a forward propagation of the information and the "return" is a backpropagation of the information. The figure below illustrates the process.

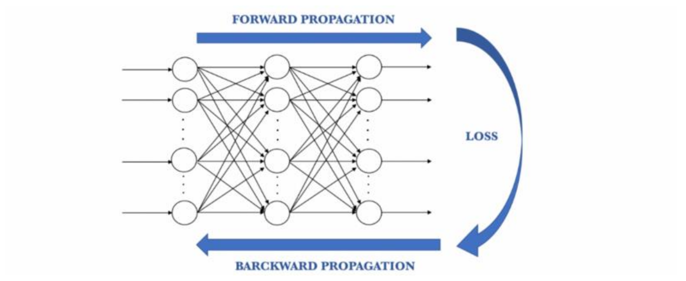


Figure 5.a

The first phase forward propagation occurs when the network is exposed to the training data and these cross the entire neural network for their predictions (labels) to be calculated. That is, passing the input data through the network in such a way that all the neurons apply their transformation to the information they receive from the neurons of the previous layer and sending it to the neurons of the next layer. When the data has crossed all the layers, and all its neurons have made their calculations, the final layer will be reached with a result of label prediction for those input examples. The loss function is used to estimate the loss (or error) and to compare and measure how good/bad our prediction result was in relation to the correct result (remember that we are in a supervised learning environment and we have the label that tells us the expected value). Ideally, we want our cost to be zero, that is, without divergence between estimated and expected value. Therefore, as the model is being trained, the weights of the interconnections of the neurons will gradually be adjusted until good predictions are obtained. Once the loss has been calculated, this information is propagated backwards. Hence, its name: backpropagation. Starting from the output layer, that loss information propagates to all the neurons in the hidden layer that contribute directly to the output. However, the neurons of the hidden layer only receive a fraction of the total signal of the loss, based on the relative contribution that each neuron has contributed to the original output. This process is repeated, layer by layer, until all the neurons in the network have received a loss signal that describes their relative contribution to the total loss. Visually, It can be summarized with this visual scheme the stages:

The various stages are as follows:

A. Back Propagation
B. Loss Function
C. Optimiser
D. Model Parameterisation
E. Epochs
F. Batch Size
G. Learning Rate
H. Initialization of parameter weights
I. Neural Network Methodology

A. *Back Propagation*

Backpropagation is a method to alter the parameters (weights and biases) of the neural network in the right direction. It starts by calculating the loss term first, and then the parameters of the neural network are adjusted in reverse order with an optimization algorithm taking into account this calculated loss.Three arguments are passed to the method: an optimizer, a loss function, and a list of metrics. In classification problems like our example, accuracy is used as a metric. Let's go a little deeper into these arguments.

B. *Loss Function*

A loss function is one of the parameters required to quantify how close a particular neural network is to the ideal weight during the training process. The choice of the best function of loss resides in understanding what type of error is or is not acceptable for the problem in particular.

C. *Optimisers*

The optimizer is another of the arguments required in the compile() method. Keras currently has different optimizers that can be used: SGD, RMSprop, Adagrad, Adadelta, Adam, Adamax, Nadam. In general, the learning process is seen as a global optimization problem where the parameters (weights and biases) must be adjusted in such a way that the loss function presented above is minimized.

D. *Model Parameterization*

It is also possible to increase the number of epochs, add more neurons in a layer or add more layers. However, in these cases, the gains in accuracy have the side effect of increasing the execution time of the learning process . We can check with the summary() method that the number of parameters increases (it is fully connected) and the execution time is significantly higher, even reducing the number of epochs. With this model, the accuracy reaches 94%. And if we increase to 20 epochs, a 96% accuracy is achieved.

E. *Epochs*

As we have already done, epochs tells us the number of times all the training data have passed through the neural network in the training process. A good clue is to increase the number of epochs until the accuracy metric with the validation data starts to decrease, even when the accuracy of the training data continues to increase (this is when we detect a potential overfitting).

F. *Batch Size*

As we have said before, we can partition the training data in mini batches to pass them through the network. In Keras, the batch_size is the argument that indicates the size of these batches that will be used in the fit() method in an iteration of the training to update the gradient. The optimal size will depend on many factors, including the memory capacity of the computer that we use to do the calculations.

G. Learning Rate

The gradient vector has a direction and a magnitude. Gradient descent algorithms multiply the magnitude of the gradient by a scalar known as learning rate (also sometimes called step size) to determine the next point.

H. Initialisation of parameter weights

Initialization of the parameters' weight is not exactly a hyperparameter, but it is as important as any of them and that is why we make a brief paragraph in this section. It is advisable to initialize the weights with small random values to break the symmetry between different neurons, if two neurons have exactly the same weights they will always have the same gradient; that supposes that both have the same values in the subsequent iterations, so they will not be able to learn different characteristics. Initializing the parameters randomly following a standard normal distribution is correct, but it can lead to possible problems of vanishing gradients (when the values of a gradient are too small and the model stops learning or takes too long due to that) or exploding gradients (when the algorithm assigns an exaggeratedly high importance to the weights).

I. Neural Network Methodology

This is the step by step building methodology of Neural Network (MLP with one hidden layer, similar to above-shown architecture). At the output layer, we have only one neuron as we are solving a binary classification problem (predict 0 or 1). We could also have two neurons for predicting each of both classes.
First look at the broad steps:
We take input and output

- X as an input matrix
- y as an output matrix

Step 1 : We initialize weights and biases with random values (This is one time initiation. In the next iteration, we will use updated weights, and biases). Let us define:

- wh as weight matrix to the hidden layer
- bh as bias matrix to the hidden layer
- wout as weight matrix to the output layer
- bout as bias matrix to the output layer

Step 2: We take matrix dot product of input and weights assigned to edges between the input and hidden layer then add biases of the hidden layer neurons to respective inputs, this is known as linear transformation:

hidden_layer_input= matrix_dot_product(X,wh) + bh

Step 3: Perform non-linear transformation using an activation function (Sigmoid). Sigmoid will return the output as 1/(1 + exp(-x)).

hiddenlayer_activations = sigmoid(hidden_layer_input)

Step 4: Perform a linear transformation on hidden layer activation (take matrix dot product with weights and add a bias of the output layer neuron) then apply an activation function (again used sigmoid, but you can use any other activation function depending upon your task) to predict the output

output_layer_input = matrix_dot_product (hiddenlayer_activations * wout) + bout

output = sigmoid(output_layer_input)

All above steps are known as "Forward Propagation"

Step 5: Compare prediction with actual output and calculate the gradient of error (Actual – Predicted). Error is the mean square loss = ((Y-t)^2)/2

E = y – output

Step 6: Compute the slope/ gradient of hidden and output layer neurons (To compute the slope, we calculate the derivatives of non-linear activations x at each layer for each neuron). Gradient of sigmoid can be returned as x * (1 – x).

slope_output_layer = derivatives_sigmoid(output)

slope_hidden_layer = derivatives_sigmoid(hiddenlayer_activations)

Step 7: Compute change factor(delta) at output layer, dependent on the gradient of error multiplied by the slope of output layer activation

d_output = E * slope_output_layer

Step 8: At this step, the error will propagate back into the network which means error at hidden layer. For this, we will take the dot product of output layer delta with weight parameters of edges between the hidden and output layer (wout.T).

Error_at_hidden_layer = matrix_dot_product(d_output, wout.Transpose)

Step 9: Compute change factor(delta) at hidden layer, multiply the error at hidden layer with slope of hidden layer activation

d_hiddenlayer = Error_at_hidden_layer * slope_hidden_layer

Step 10: Update weights at the output and hidden layer: The weights in the network can be updated from the errors calculated for training example(s).

wout = wout + matrix_dot_product(hiddenlayer_activations.Transpose, d_output)*learning_rate

wh = wh + matrix_dot_product(X.Transpose,d_hiddenlayer)*learning_rate-learning_rate: The amount that weights are updated is controlled by a configuration parameter called the learning rate)

Step 11: Update biases at the output and hidden layer: The biases in the network can be updated from the aggregated errors at that neuron.

- bias at output_layer =bias at output_layer + sum of delta of output_layer at row-wise * learning_rate
- bias at hidden_layer =bias at hidden_layer + sum of delta of output_layer at row-wise * learning_rate

bh = bh + sum(d_hiddenlayer, axis=0) * learning_rate

bout = bout + sum(d_output, axis=0)*learning_rate

Steps from 5 to 11 are known as “Backward Propagation“

One forward and backward propagation iteration is considered as one training cycle. As I mentioned earlier, When do we train second time then update weights and biases are used for forward propagation.

Above, we have updated the weight and biases for hidden and output layer and we have used full batch gradient descent algorithm.

VI.. Results And Inferences

The dataset was constructed from a number of scanned document dataset available from the National Institute of Standards and Technology (NIST). This is where the name for the dataset comes from, as the Modified NIST or MNIST dataset. Images of digits and alphabets were taken from a variety of scanned documents, normalized in size and centered. This makes it an excellent dataset for evaluating models, allowing the developer to focus on machine learning with very little data cleaning or preparation required. Each image is a 28 by 28 pixel square (784 pixels total). A standard spit of the dataset is used to evaluate and compare models, where 60,000 images are used to train a model and a separate set of 10,000 images are used to test it. It is a digit recognition task. [13] As such there are 24 Alphabets (A - Z) and (a-z) and 10 digits (0 to 9) or 10 classes to predict. Results are reported using prediction error, which is nothing more than the inverted classification accuracy. Excellent results achieve a prediction error of less than 1%. State-of-the-art prediction error of approximately 0.2% can be achieved with large Convolutional Neural Networks

A. Evaluation Parameters

The performance of the algorithms is measured as used in multilayer perceptrons: backpropagation and resilient propagation. We have considered the scenario of recognition from image, where the dataset consists only of 40 character image bitmaps per character. For this comparison, the datasets are only comprised of characters of digits, therefore the size of the dataset contains 400 examples. For relevant values, we have split the dataset into training and validation sets, with the ratio being 7:3.[14] Also, before using the learning algorithms, the dataset has been randomly shuffled. The configuration of the learning model whose results are presented here is:

- The regularization parameter is 0.
- The number of epochs is 100.
- In backpropagation, the learning rate is 0.3.
- In resilient backpropagation, $\eta -$, $\eta +$, and $\Delta 0$ are 0.5, 1.2, and 0.01, respectively.
- The perceptron architectures are as described in the plan of solution.

The measured error of the backpropagation and CNN algorithms on the training and validation sets. This has been tested using fractions of the dataset of various sizes and a learning curve has been plotted. Learning curve represents error as a function of the dataset size and is a perfect tool to visualize high bias or variance.

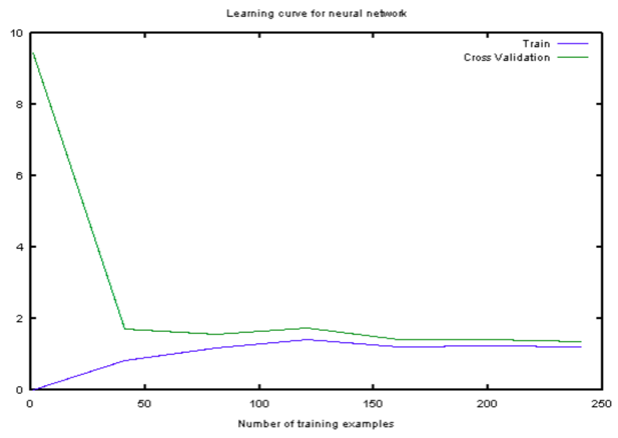


Figure 3.1

O

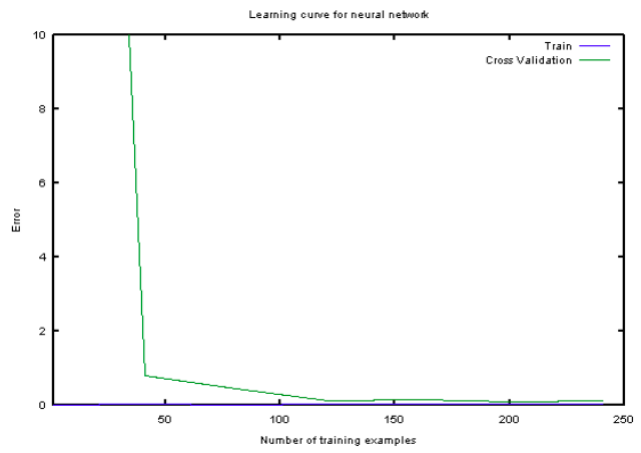


Figure 3.2

In the learning curves, no significant overfitting or underfitting is apparent. We can see that the RPROP algorithm manages to converge to a better minimum given 100 epochs than backpropagation. This is caused by the advantages of the RPROP algorithm to pure backpropagation that we explained earlier in this work. Table 1 confirms these findings.

Name	Classes	No. Training	No. Testing	Validation	Total
By_Class	62	697,932	116,323	No	814,255
By_Merge	47	697,932	116,323	No	814,255
Balanced	47	112,800	18,800	Yes	131,600
Digits	10	240,000	40,000	Yes	280,000
Letters	37	88,800	14,800	Yes	103,600
MNIST	10	60,000	10,000	Yes	70,000

Table 3.1.

B. *Experimental Methodology*

As we have more data available in the touch mode than a pure image bitmap, we have also decided to collect the bitmap of stroke end points to be able to better distinguish characters such as '8' and 'B', as mentioned in the overview. The resized bitmaps of these characters are often similar, but the writing style of each is usually different. By providing this extra bitmap with each example, we are giving a hint to the neural network classifier about what features to focus on when performing automatic feature extraction with the hidden layer. The pipeline for recognition based on an image or a camera frame is different:

1. Acquire the image bitmap in gray-scale colors.
2. Apply a median filter to the bitmap.
3. Segment the bitmap using thresholding to get a binary bitmap.
4. Find the bounding boxes of external contours in the bitmap.
5. Extract sub-bitmaps from the bounding boxes.
6. Resize the sub-bitmaps to 20x20 pixels.
7. Unroll the sub-bitmap matrices to feature vectors per 400 elements.
8. Feed each feature vector to a trained multilayer perceptron, giving us predictions.

C. *Validation*

The proposed alphabet recognition system was trained to recognize handwritten English alphabets. Since the alphabets are divided into 25 segments, neural network architecture is designed specially for the processing of 25 input bits. The network parameters used for training are: Learning rate coefficient = 0.05 No. of Units in Input layer = 25 No. of Hidden Layers = 2 No. of Units in Hidden layer= 25 Initial Weights = Random [0,1] Transfer Function Used for Hidden Layer 1 = “Logsig” Transfer Function Used for Hidden Layer 2 = “Tansig” The training set involves the binary codes of alphabets.[15] It was not practical to input these shapes individually when creating training sets, because the shape of a particular segment of the actual character depends on handwriting. Therefore, this was automated so that the entire letter is input to the system, and then the shape of the segment needed is extracted from this full letter instead of drawing the shape of the segment itself. The figure below shows the results for the training dataset (*Figure 3.3 and 3.4*) and the test dataset(*Figure 3.5*).

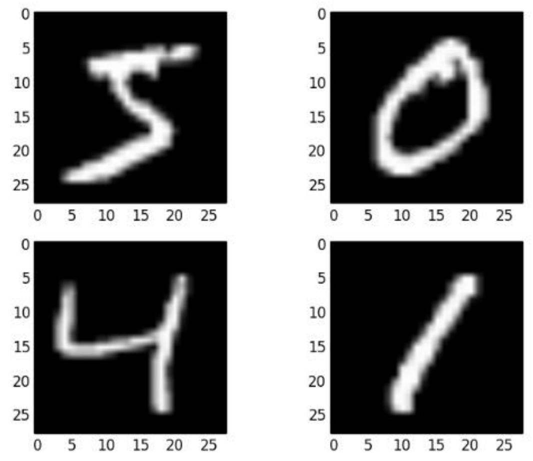
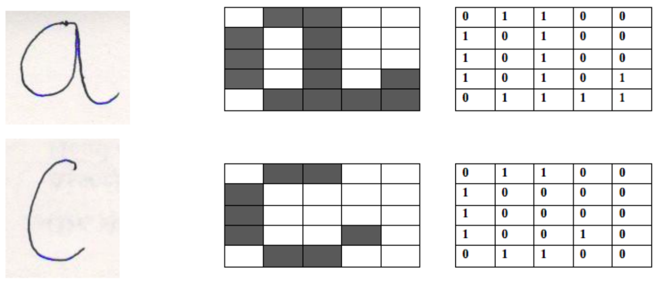


Figure 3.3



0	1	1	0	0
1	0	1	0	0
1	0	1	0	0
1	0	1	0	1
0	1	1	1	1

0	1	1	0	0
1	0	0	0	0
1	0	0	0	0
1	0	0	1	0
0	1	1	0	0

Figure 3.4

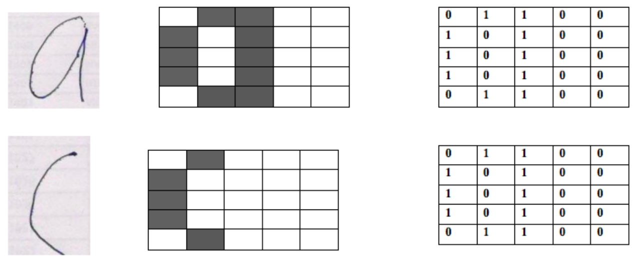


Figure 3.5

The binary input of each alphabet of different handwriting styles is then feed to all the units of input layer of the network at once and the network after processing through hidden layers and with the training algorithm, Gradient Descent Back propagation, can be trained for these inputs. Now, the trained network is capable to recognize the alphabet of different handwriting. The results are tabulated below.

Alphabet	No. of samples for training	No. of samples for testing	No. of epochs	% Recognition Accuracy
a	20	5	294	94.0
b	20	5	321	83.0
c	20	5	587	71.0
d	20	5	282	88.0
e	20	5	548	64.0
f	20	5	254	85.0
g	20	5	247	89.0
h	20	5	263	92.0
i	20	5	658	72.0
j	20	5	599	73.0
k	20	5	300	91.0
l	20	5	652	71.0
m	20	5	456	86.0
n	20	5	398	82.0
o	20	5	356	94.0
p	20	5	264	88.0
q	20	5	287	82.0
r	20	5	669	70.0
s	20	5	202	88.0
t	20	5	252	79.0
u	20	5	458	80.0
v	20	5	488	77.0
w	20	5	511	94.0
x	20	5	341	91.0
y	20	5	268	71.0
z	20	5	296	90.0

Table 3.2

We can observe from the table that recognition accuracy is lower for the similar input patterns like: c & e; i, j, l & r; and u & v. for these similar patterns in different handwriting, even human eye can not be able easily distinguish, hence the machine needs lots of training epochs to recognize them but with some misclassifications.

VII. Conclusion And Future Possibilities

This work has mostly been focused on the machine learning methods used in the project. At first, we reviewed the approaches that are nowadays used in similar applications. After that, we delved into the inner workings of a multilayer perceptron, focusing on backpropagation and resilient back, which has been implemented. Even though our dataset consists of the images of every word separately, some words within these images were slightly tilted. This was because the participants of the dataset were asked to write on blank paper with no lines, and some of the words were written in a more tilted fashion. This occasion happens very frequently in real life whether or not the page has lines, thus we decided to make our training data more robust to this issue by rotating an image towards the right by a very small angle with random probability and adding that image to our training set. The technique we have found to be useful during experimentation was keeping the number of epochs low when trying out different hyperparameters. This approach certainly saved us a lot of time in training; however, it also had its own disadvantages. This data augmentation technique helped us make our model more robust to some minor yet so frequent details that might come up in our test set. Before training the models with the dataset, we have applied various preprocessing and data augmentation techniques on our dataset in order to make our data more compatible with the models and to make our dataset more robust to real life situations. With the knowledge we had described, we specified the requirements of the project and planned the solution. Several improvements for the application or the learning model used within can be suggested. For example, the feature extraction performed by the neural network could be constrained to operate on more strictly preprocessed data. Also, several classifiers learning on different features could be combined to make the system more robust.

The proposed and developed a scheme for recognizing handwritten English alphabets and numbers. It has been tested on experiment over all English alphabets and numerical digits with several Handwriting styles. Experimental results shown that the machine has successfully recognized the alphabets and numbers with the average accuracy of 82.5%, which significant and may be acceptable in some applications. The machine found less accurate to classify similar alphabets and in future this misclassification of the similar patterns may improve and further a similar experiment can be tested over a large data set and with some other optimized networks parameters to improve the accuracy of the machine.

Firstly, to have more compelling and robust training, we could apply additional preprocessing techniques such as jittering. We could also divide each pixel by its corresponding standard deviation to normalize the data. Next, given time and budget constraints, we were limited to 20 training examples for each given word in order to efficiently evaluate and revise our model. Another method of improving our character segmentation model would be to move beyond a greedy search for the most likely solution. We would approach this by considering a more exhaustive but still efficient decoding algorithm such as beam search. We can use a character/word-based language-based model to add a penalty/benefit score to each of the possible final beam search candidate paths, along with their combined individual softmax probabilities, representing the probability of the sequence of characters/words. If the language model indicates perhaps the most likely candidate word according to the softmax layer and beam search is very unlikely given the context so far as opposed to some other likely candidate words, then the model can correct itself accordingly

References

[1] H. Al-Yousefi and S. S. Udpa, "Recognition of handwritten Arabic characters," in Proc. SPIE 32nd Ann. Int. Tech. Symp. Opt. Optoelectric Applied Sci. Eng. (San Diego, CA), Aug. 1988.

[2] K. Badi and M. Shimura, "Machine recognition of Arabic cursive script" Trans. Inst. Electron. Commun. Eng., Vol. E65, no. 2, pp. 107-114, Feb. 1982.

[3] M Altuwaijri , M.A Bayoumi , "Arabic Text Recognition Using Neural Network" ISCAS 94. IEEE International Symposium on Circuits and systems, Volume 6, 30 May-2 June 1994.

[4] C. Bahlmann, B. Haasdonk, H. Burkhardt., "Online Handwriting Recognition with Support Vector Machine – A Kernel Approach", In proceeding of the 8th Int. Workshop in Handwriting Recognition (IWHFR), pp 49- 54, 2002

[5] Homayoon S.M. Beigi, "An Overview of Handwriting Recognition," Proceedings of the 1st Annual Conference on Technological Advancements in Developing Countries, Columbia University, New York, July 24-25, 1993, pp. 30- 46.

[6] Nadal, C. Legault, R. Suen and C.Y, "Complementary Algorithms for Recognition of totally Unconstrained Handwritten Numerals," in Proc. 10th Int. Conf. Pattern Recognition, 1990, vol. 1, pp. 434-449.

[7] S. Impedovo, P. Wang, and H. Bunke, editors, "Automatic Bankcheck Processing," World Scientific, Singapore, 1997.

[8] CL Liu, K Nakashima, H Sako and H. Fujisawa, "Benchmarking of state-of- the-art techniques," Pattern Recognition, vol. 36, no 10, pp. 2271– 2285, Oct. 2003.

[9] M. Shi, Y. Fujisawa, T. Wakabayashi and F. Kimura, "Handwritten numeral recognition using gradient and curvature of gray scale image," Pattern Recognition, vol. 35, no. 10, pp. 2051–2059, Oct 2002.

[10] LN. Teow and KF. Loe, "Robust vision-based features and classification schemes for off-line handwritten digit recognition," Pattern Recognition, vol. 35, no. 11, pp. 2355–2364, Nov. 2002.

[11] K. Cheung, D. Yeung and RT. Chin, "A Bayesian framework for deformable pattern recognition with application to handwritten character recognition," IEEE Trans PatternAnalMach Intell, vol. 20, no. 12, pp. 382– 1388, Dec. 1998.

[12] I.J. Tsang, IR. Tsang and DV Dyck, "Handwritten character recognition based on moment features derived from image partition," in Int. Conf. image processing 1998, vol. 2, pp 939–942.

[13] H. Soltanzadeh and M. Rahmati, "Recognition of Persian handwritten digits using image profiles of multiple orientations," Pattern Recognition Lett, vol. 25, no. 14, pp. 1569–1576, Oct.2004.

[14] FN. Said, RA. Yacoub and CY Suen, "Recognition of English and Arabic numerals using a dynamic number of hidden neurons" in Proc. 5th Int Conf. document analysis and recognition, 1999, pp 237–240

[15] J. Sadri, CY. Suen, and TD. Bui, "Application of support vector machines for recognition of handwritten Arabic/Persian digits," in Proc. 2th Iranian Conf. machine vision and image processing, 2003, vol. 1,pp 300–307.