Offline Handwritten Telugu Character Recognition using Convolutional Neural Networks

A Project Report

submitted by

TANIKELLA TEJASWI

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

BACHELOR OF TECHNOLOGY

DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY MADRAS.

May 2016

CERTIFICATE

This is to certify that the report titled **Offline Handwritten Telugu Character Recognition using Convolutional Neural Networks**, submitted by **Tanikella Tejaswi**, to the Indian Institute of Technology, Madras, towards the partial fulfilment of **B. Tech** Degree requirements, is a bona fide record of the work done by him under my supervision. The contents of this report, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Prof. Srinivasa Chakravarthy V.
Project Guide
Professor
Dept. of Biotechnology
IIT-Madras, 600 036

Prof. Radha Krishna Ganti
Co-Guide
Assistant Professor
Dept. of Electrical Engineering
IIT-Madras, 600 036

Place: Chennai

Date: 10th May 2016

ACKNOWLEDGEMENTS

I would first like to thank my advisor Prof. Srinivasa Chakravarthy for his invaluable support and for steering me in the right direction whenever I needed help.

I would also like to acknowledge Prof. Radha Krishna Ganti as the co-guide of this project, and I am grateful for his valuable insights into the project.

Finally, I must express my gratitude to my family and my friends for their unfailing support and encouragement without which none of this would have been possible. Thank You.

Tanikella Tejaswi

ABSTRACT

KEYWORDS: Pattern Recognition; Handwritten Character Recognition; OCR; Telugu; Convolutional Neural Networks; Deep Learning; Dropout.

Handwriting recognition is a very active research area in the field of pattern recognition. However lots of studies have been done on scripts like English, Japanese, etc. and very little work has been done on Indian languages. But lately, there is a growing interest in handwriting recognition of Indian languages.

This report uses convolutional neural networks to proposes a model that recognises Telugu characters. Telugu is one of India's twenty-one officially recognised languages and is primarily spoken in two states. It is a Dravidian Language and is syllabic in nature. The Telugu script consists of 16 vowels and 36 consonants. Each character may be a single vowel, a consonant, a consonant and vowel modifier or a consonant with a consonant modifier and a vowel modifier. The last case has been ignored in this report to reduce complexity. The network uses deep learning and convolutional networks to recognise the characters in each image of size 28x28. A dataset with approximately 20,000 images was used to train the model, and the model gives an accuracy of 90% over all test images.

This project aims at making a better model for Telugu character recognition. This model can next be integrated into software with a front end which recognises the user's writing. The structure of the model and its training and regularization have been presented. The corresponding results and comparative performance have also been discussed to show it's effectiveness.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 The problem

The main goal of this project is to create a model which can efficiently recognise Telugu characters from images using Deep Learning. Although a lot of work is being done in the field of handwriting recognition, very little work can be found in the area of recognition of Indian languages. Also, most of the work found regarding Telugu character recognition is mostly about recognising printed characters (OCR). Thus, this model tries to fill this gap and recognise handwritten Telugu characters.

Telugu is a Dravidian language with speakers mainly in southern India. The Telugu script is an abugida, i.e. it is a segmental writing system where consonant-vowel sequences are written as a unit. Each segment can be one of these types:

- V type: contains a Vowel.
- C type: contains a unmodified consonant
- CV type: contains a consonant and a vowel modifier
- CCV type: contains a consonant, a consonant modifier and a vowel modifier

This makes recognizing Telugu characters a difficult task when compared to languages like English or Chinese. We have ignored the CCV type for our project for simplicity. Telugu has a total of 16 vowels and 36 consonants. Each consonant can have a vowel modifier, i.e. around 600 characters of CV type. The consonant is written first with the modifiers around it.

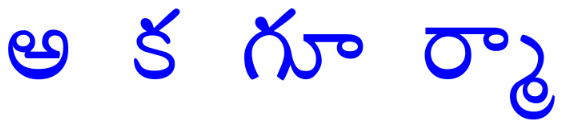

Figure 1.1: Four examples of Telugu Characters. Plain Vowel (V type), unmodified Consonant (C type), Consonant with a vowel modifier (CV type) and a Consonant with a consonant and vowel modifier (CCV type).

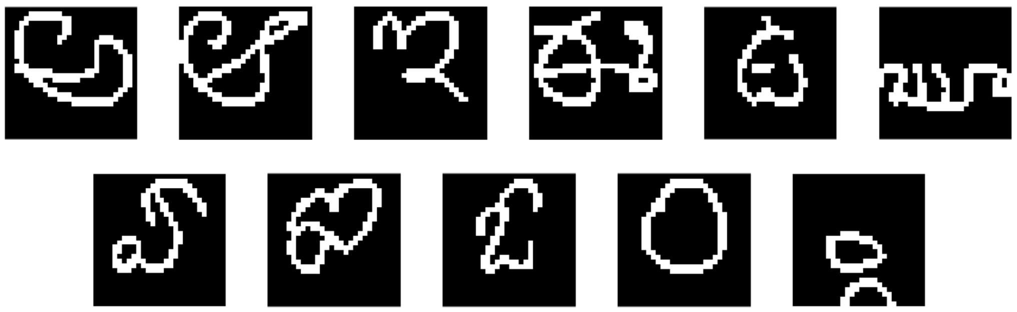

Figure 1.2: Some examples from vowel classes.

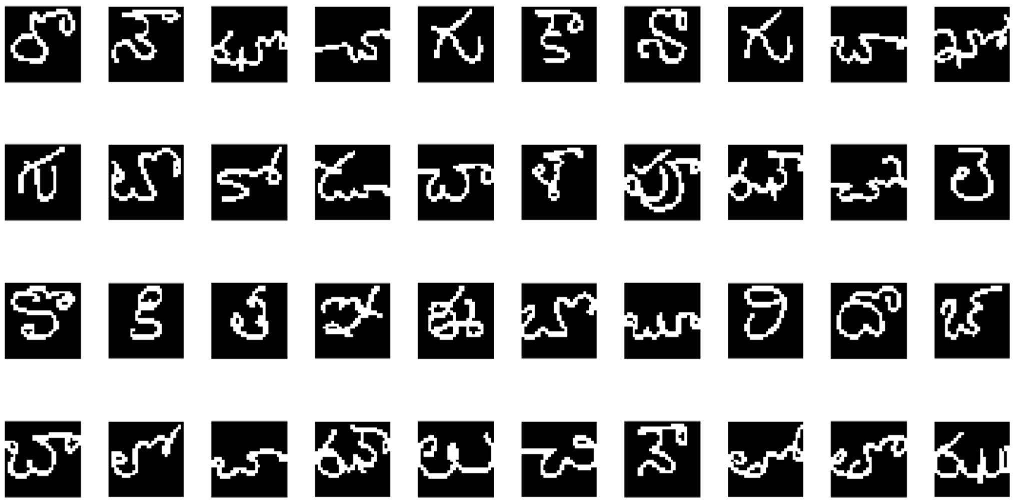

Figure 1.3: Some C and CV type examples from the Telugu Handwritten dataset.

1.2 Training Data

The training data consists of 20,000 images. Each image is 28 pixels wide and 28 pixels in length. Each image is associated with a single number if it is a V type and pair of numbers about the consonant and vowel modifier if it is a C or CV type.

The number of examples of certain classes was found to be very little when compared to others, so we use affine transformations (described in Section 3.1.1) to fill the gap. Also, some specific characters were omitted from the training dataset since they needed higher resolution images to make any sense of them.

1.3 Previous Attempts

The interest in recognising and identifying Indian scripts is increasing lately. Some papers like Rajashekararadhya and Ranjan (2008) work on offline handwritten numeral recognition and have proposed a technique based on zone and image centroid. The dataset was limited and the accuracy hence very high. Pal *et al.* (2007), used a quadratic classifier based scheme. An offline Telugu handwritten character recognition using multilayer perceptrons proposed by Vikram *et al.* reports accuracies of 85%.

An online character recognition system proposed by Rajkumar *et al.* (2012) gives very high accuracy for a similar handwritten data. A project similar to the one being presented was proposed by Achanta and Hastie (2015), where optical character recognition system using convolutional neural networks and deep learning is used. But the problem in our project has a higher variance as the data is handwritten and hence is even harder to classify.

CHAPTER 2

About Convolutional Neural Networks

In this section firstly, the construction of a Convolutional Neural Network is explained. This followed by a brief introduction to the regularisation used in the networks and why they work. Finally, a small CNN is taken as an example to demonstrate some ideas.

2.1 Convolutional Neural Networks Construction

Convolutional Neural Networks are biologically inspired feed-forward neural networks. Its construction is based on the way animal visual cortex is built where each neuron is receptive to a small part of the image. The information gained by a layer of such neurons is passed onto another similar layer. This exploits any spatial correlation which is used to identify the image. Each CNN starts off with a convolution layer. The image is convolved with a randomly initialized filter. This is equivalent to a neuron gaining some information about the input data. Then this new image is sent through an activation function. The transformed data is then sent to *pooling* layers, where this data is pooled. This pooled data is again sent through convolution, activation and pooling layers repeatedly. Consecutive convolution and pooling layers are followed by a classifier to classify the data. Each of these layers is explained below.

2.1.1 Convolution Layer

Convolution of two functions is defined for a function $f(x)$ with $g(x)$ as

$$s(x) = f * g(x) = \int f(t)\, g(x-t)\, dt = \int g(t)\, f(x-t)\, dt \tag{2.1}$$

As seen from Eq 2.1, the operation is commutative. The same can be defined for two discrete series $x[n]$ and $y[n]$ as

$$s[n] = x * y(x) = \sum_{m=-\infty}^{\infty} x[m]\, y[n-m] = \sum_{m=-\infty}^{\infty} y[m]\, x[n-m] \tag{2.2}$$

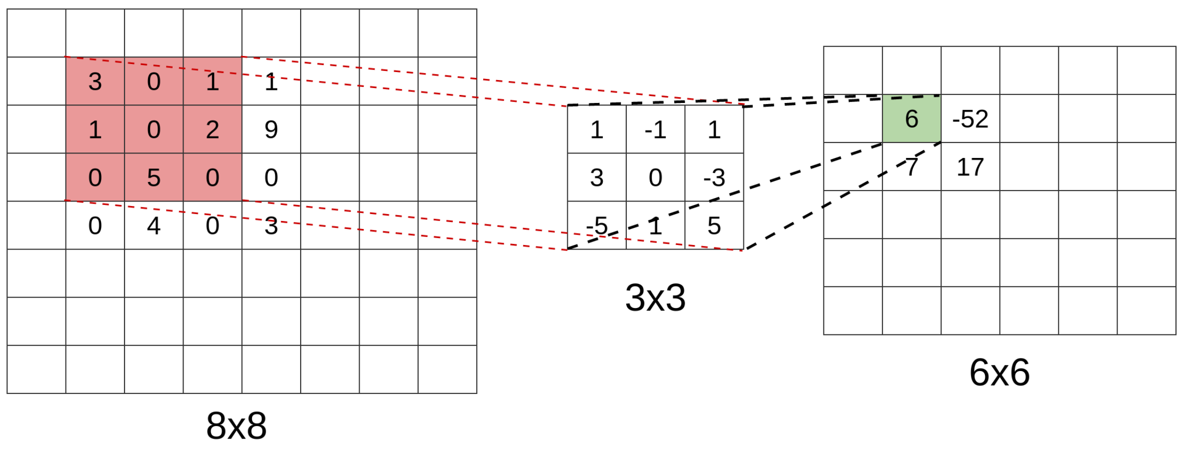

Figure 2.1: Visualizing the convolution of a 8×8 matrix with a kernel of size 3×3 to results in a 6×6 image.

The operation can be explained as $y[n]$ being flipped and repeatedly being multiplied to $x[n]$. These products are added to get $s[n]$. Convolution is done similarly in a convolution layer. The input data, say an image I of size $N \times N$, is convolved with a two-dimensional kernel K of size $M \times M$. The resulting image J is

$$J[i,j] = \sum_m \sum_n I[m,n]\ K[i-m, j-n] \tag{2.3}$$

Some neural network libraries use cross-correlation instead, which is the same as convolution but without the flipping of the kernel

$$J[i,j] = \sum_m \sum_n I[m,n]\ K[m+i, n+j] \tag{2.4}$$

The convolution between the image and kernel can also be seen as the multiplication of the image and the kernel as it moves in across the image. The step size of the kernel can be set to any value appropriately. This has been visualized in Figure 2.1.

Convolution layer uses three important ideas that help the machine learn to recognise a given image:

- *Sparse Interactions*:
  Compared to conventional feed-forward networks, CNNs have fewer interactions with the input data. Traditional Neural Networks have matrix multiplication between input data and the weights at each node. Compared to this, the interaction between the image and the smaller size kernel is much less. This means lesser memory requirements and fewer operations.
- *Parameter Sharing*:

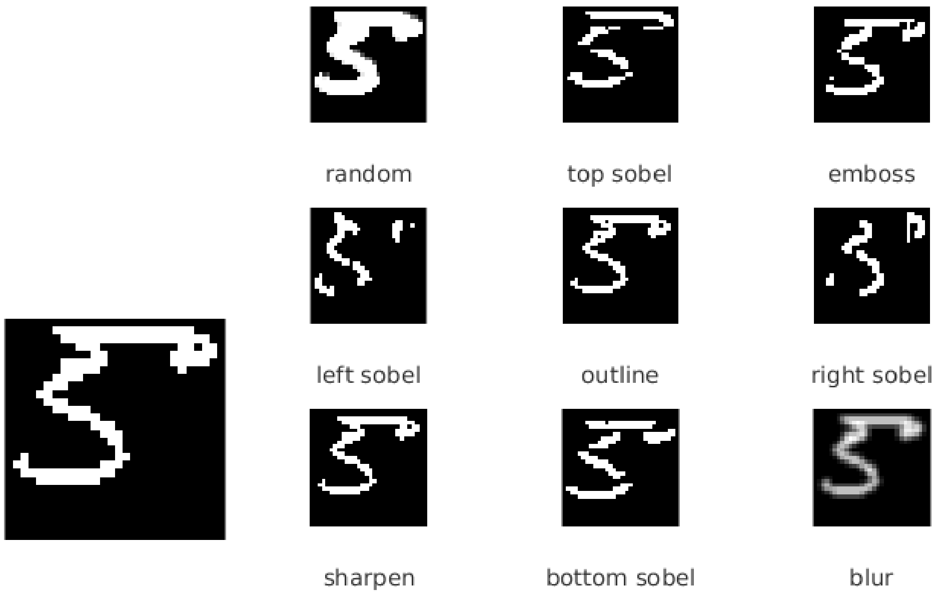

Figure 2.2: The results when an example from the dataset is convolved with some standard 3×3 image processing kernels. The name of the kernel with which the image has been convolved with is written beneath it.

In CNNs, the parameters are used more than once. In traditional feed-forward networks, the weight between the input element and node is used once and never revisited. While convolving however, the kernel's elements are repeatedly used. Thus convolution is more efficient than dense matrix multiplication in terms of memory requirements and statistical efficiency.

- *Equivariant Representations*:
  Due to convolution the layer has equivariance to translation, i.e. if the image is translated the output will also translate accordingly. This can be seen clearly from the equations above. But, convolution is not equivariant to scaling or rotation. Other tricks are needed for this.

When a input image of size $N \times N$ is taken and convolved with kernel of size $k \times k$ with a step size of s , then the size of the output image is $M \times M$ where,

$$M = ceil\left(\frac{N-k}{s} + 1\right) \tag{2.5}$$

Some libraries also use $floor(\cdot)$ or $round(\cdot)$ instead.

2.1.2 Activation Function

The outputs of the Convolution Layer are sent through an activation function. Traditional activation functions like $sigmoid(\cdot)$ and $tanh(\cdot)$ are usually not used. Rectified linear activation function or ReLU is the most widely used activation function while training deep networks (Nair and Hinton (2010)). ReLU is defined as

$$ReLU(x) = max(0, x) \tag{2.6}$$

Some papers also report using a “leaky” ReLU,

$$leaky\ ReLU(x) = \begin{cases} x & x \geq 0 \\ ax & x \leq 0 \end{cases} \tag{2.7}$$

where ‘a’ is a small constant. This creates a small slope when the node is not activated. These activation functions have been compared in the Figure 2.2.

Using ReLU has the following advantages:

- Training with it is faster. Unlike the sigmoid or the tanh function, exponential computation is not needed which makes it computationally faster. For a ReLU, a comparison (and a multiplication if it is a Leaky ReLU) is sufficient.
- No gradient vanishing problem. The gradient of the sigmoid functions tends to zero after a certain value. This means the learning of the neurons below is directly affected, causing slower learning. The ReLU, if activated, always has a constant positive gradient. Hence, the learning is never affected.
- Nodes are sparsely activated. On random activation, in a given set of nodes, only half of them are activated. Thus, the models can be trained with lesser regularisation.

2.1.3 Pooling

The next step is to “pool” the information and pass it on. In this layer, some information or statistic of a group of inputs is taken and passed on as the outputs. For example, the pooling used in the project - max pooling, reports the maximum value in a given neighbourhood. Other popular pooling techniques include average or a weighted average of the neighbourhood.

Pooling is done either in fixed neighbourhoods, by moving around in steps or can also be done so that the number of statistics received is independent of the input image size. For example, we can take four statistics by dividing the image into four quadrants no matter its size.

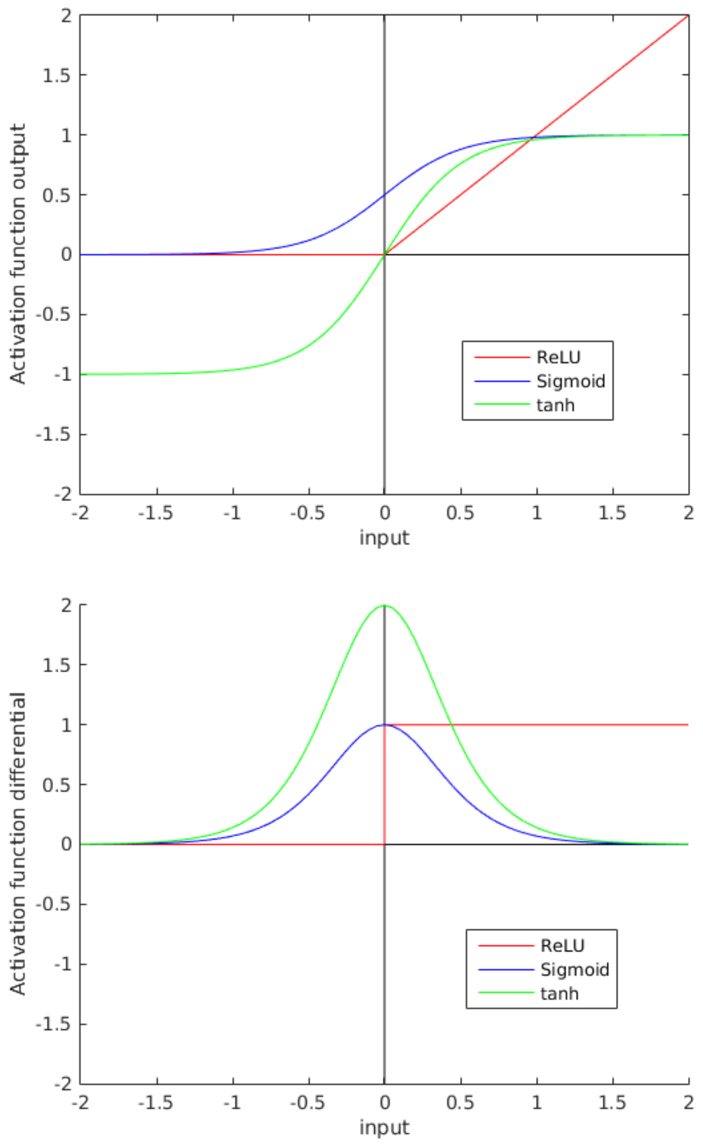

Figure 2.3: The activation function values and the gradient of three popular activation functions

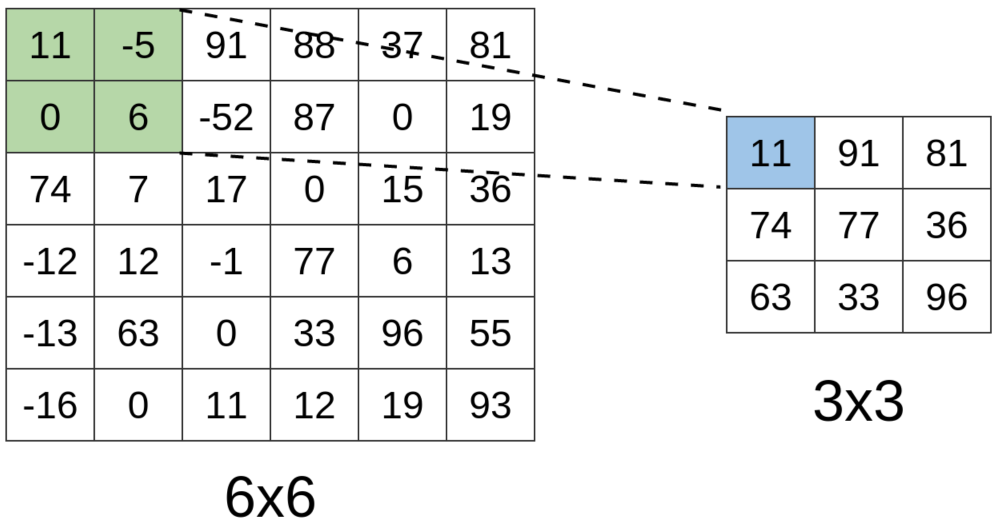

Figure 2.4: Visualizing a 6×6 image being max pooled over a neighbourhood of 2×2 by moving in steps of 2 in either directions which results in a 3×3 image.

Pooling the data makes its output invariant to small disturbances in the input. This is a vital property of the Convolutional Neural Networks. The output of pooling layer is again a 2-D matrix, an image for our discussion. This has been visualised in Figure 2.4.

When an input image of size $N \times N$ is taken and pooling is done in a neighbourhood of size $k \times k$ with a step size of s , then the size of the output image is $M \times M$ where,

$$M = floor\left(\frac{N-k}{s} + 1\right) \tag{2.8}$$

Some libraries also use $ceil(\cdot)$ or $round(\cdot)$ instead.

After pooling the image is sent to the next layer, either a convolution layer or is “reshaped” into a single array and connected to ordinary fully connected linear layers and ending with a softmax layer.

2.2 Regularisation

In this project, regularisation has played a major role in increasing the accuracy of the model. Instead of small CNNs, large CNNs with regularisation have been used which gave better results. Two ways of introducing regularisation is Dropout (Hinton *et al.* (2012)) and Batch Normalization (Ioffe and Szegedy (2015)). Both of which were used in the project.

2.2.1 Dropout

During training, dropout “drops” the inputs following a Bernoulli distribution with probability ‘p’, i.e. its value is made zero, else the outputs are scaled by a factor of $1/(1-p)$. This multiplicative noise results in a penalty that is data dependent thus incorporating the distribution of input data into the regularisation process.

When large neural networks are trained without any regularisation, multiple neurons learn the same or similar characteristics. This overfitting greatly decreases the model’s test accuracy. This can be reduced with dropout. When some of the outputs are dropped, the neuron learns to detect very specific features which make the input unique. By dropping features with a probability of p, we are preventing co-adaptation of features.

In CNNs, we add dropout to the convolved images, which is then sent to an activation function. Random dropout has been proven to be very useful and the improvements have been well documented.

2.2.2 Batch Normalization

While training deep neural networks, the distribution of each layers’ inputs changes the as the parameters change. When the models are deep, even small changes can amplify and cause large changes. One example of this could be that the training data and the test data follow different distributions. This causes disturbances in the distributions of the internal nodes. This is called *internal covariate shift*. This causes learning to be slower.

Batch Normalization offers to eliminate this and hence accelerate the training. It normalizes the input with parameters which are learnt over the course of training. This has been presented by S.Ioffe and C. Szegedy in their paper “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift” **[bibliography]**. It is shown to make training faster. The batch normalisation is done before sending data from the convolution layer into the activation function.

2.3 An example

A typical Convolutional Neural Network with two layers of convolution-ReLU-pooling layers followed by a fully connected layer and ending with a softmax layer has been shown in Table 2.1.

Sl.No.	Layer Type and Parameters
1	Convolution Layer with 4 kernels, of size 4×4, with a step size 1×1
2	ReLU (Activation Layer)
3	Max Pooling over 2×2 neighbourhood with a step size of 2×2
4	Convolution Layer with 8 kernels, of size 4×4, with a step size 1×1
5	ReLU (Activation Layer)
6	Max Pooling over 2×2 neighbourhood with a step size of 1×1
7	Reshape the 8 node with 8×8 images into a single 512×1 matrix
8	Linear Fully connected layer from 512 nodes to 100 nodes
9	ReLU (Activation Layer)
10	Linear Fully connected layer from 100 nodes to 3 nodes
11	SoftMax layer

Table 2.1: A small Convolutional Neural Network.

If the input image is of size 28×28, then using equations discussed previously, we can calculate the size of the image after convolution (layer 1) is 25×25. This image after max pooling (layer 3) gives an image of size 12×12. This image is sent through convolution (layer 4) again giving an image of size 9 This is then sent through a max pooling layer (layer 6) which gives an image of size 8×8. There are 8 kernels, each with a 8×8 image, i.e. $8 \times 8 \times 8$ dimensional information. This is made into a single array of size 512 nodes. This is followed by an ordinary fully connected network, which finally ends in a SoftMax layer with three outputs.

CHAPTER 3

Training the CNNs

This chapter describes the data and how it was increased using affine transformations. This is followed by the overview of the model, the structure of the networks and various methods to train faster.

3.1 Character Data

The dataset contains a total of 17744 images each of size 28×28. It contains 1039 vowels and 16705 consonants (C and CV type).
There are 16 classes of vowels. And there are 36 classes of consonant and 15 classes of vowel modifiers.

As shown in the histograms in Figures 3.2,3.3 and 3.4 ,some of the classes are empty, and some have too few examples to train with. Thus, affine transformations were used on examples to increase the numbers.

3.1.1 Affine Transformations

Given a image 'I', (a 2-D matrix), it can be translated and rotated slightly to get similar images for training. This can be achieved easily with Affine translations. Given some points in XY axis whose each point (x, y) is moved to a a point $(x'y')$ of the new axis $X'Y'$ by the matrix M.

$$M = \begin{bmatrix} a_x & a_y \\ b_x & b_y \\ c_x & c_y \end{bmatrix} \tag{3.1}$$

Transformation	M	(x', y')
Translation	$\begin{bmatrix} 1 & 0 \\ 0 & 1 \\ t_x & t_y \end{bmatrix}$	$x' = x + t_x, y' = y + t_y$
Rotation	$\begin{bmatrix} \cos\theta & \sin\theta \\ -\cos\theta & \sin\theta \\ 0 & 0 \end{bmatrix}$	$x' = x\cos\theta + y\sin\theta, y' = -x\cos\theta + y\sin\theta$
Scaling	$\begin{bmatrix} s_x & 0 \\ 0 & s_y \\ 0 & 0 \end{bmatrix}$	$x' = s_x x,\ y' = s_y y$

Table 3.1: Affine Transformations along with the transformation matrices.

(x', y') and (x, y) are related as,

$$\begin{bmatrix} x' & y' \end{bmatrix} = \begin{bmatrix} x & y & 1 \end{bmatrix} \begin{bmatrix} a_x & a_y \\ b_x & b_y \\ c_x & c_y \end{bmatrix} = \begin{bmatrix} a_x x + b_x y + c_x & a_y x + b_y y + c_y \end{bmatrix} \tag{3.2}$$

Each point in the new axis is uniquely related to a point in the old axis. Some of the transformations used are given in Table 3.1.

For randomly chosen images from each class, slight translation and a small rotation is done. The values of translation and rotation are chosen randomly. A sample image and it's Affine transformations are shown in Figure 3.1.

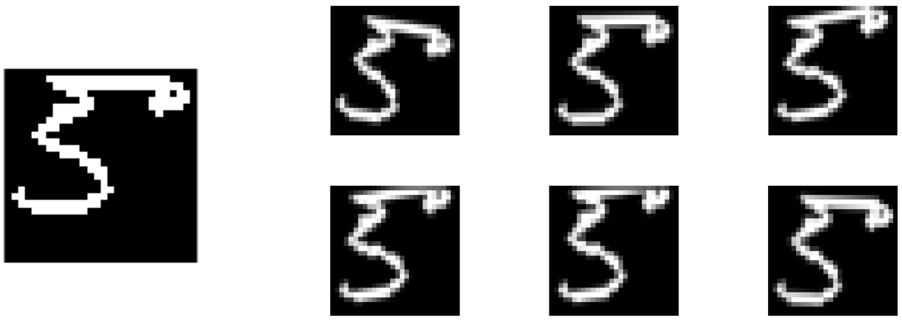

Figure 3.1: An image of the original dataset on the left and on the right six of its transformed images.

After the transformed images are added, especially to classes with very low numbers, the number of examples in each class become comparable. In the next section, the training of networks is described.

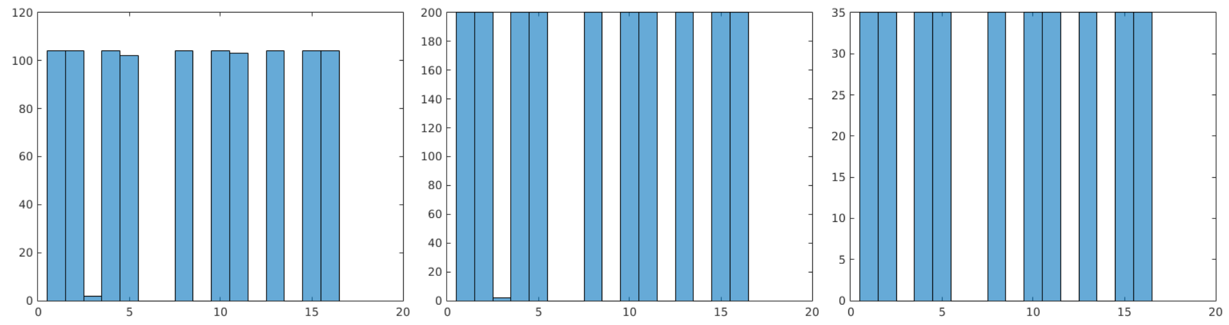
Figure 3.2: The number of examples in each vowel class in the complete untransformed dataset, the training dataset after adding transformations and the test dataset after adding the transformations are shown in these histograms.

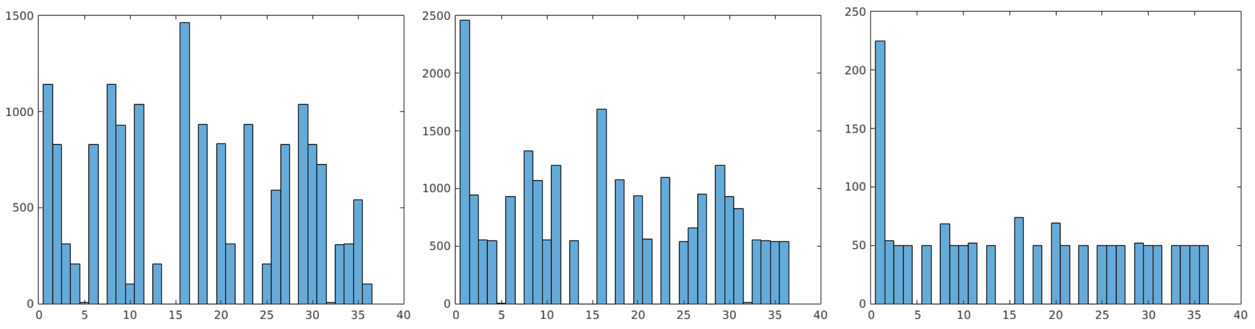
Figure 3.3: The number of examples in each base consonant class in the complete untransformed dataset, the training dataset after adding transformations and the test dataset after adding the transformations are shown in these histograms.

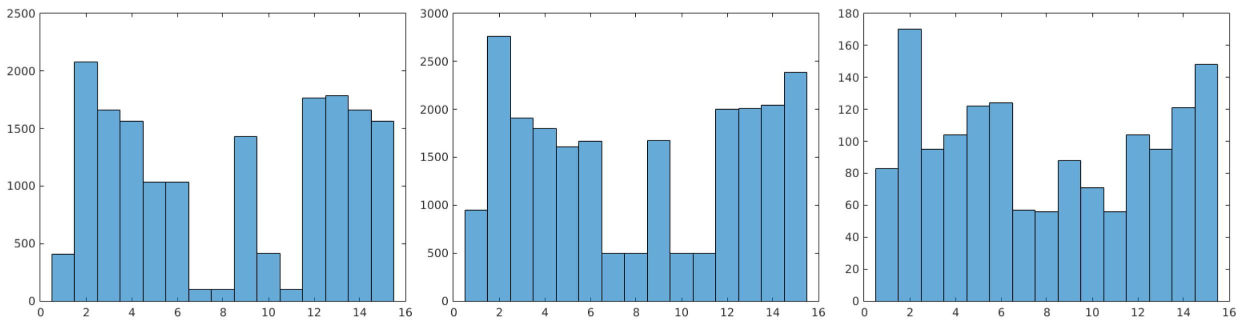
Figure 3.4: The number of examples in each vowel modifier class in the complete untransformed dataset, the training dataset after adding transformations and the test dataset after adding the transformations are shown in these histograms.

3.2 Overview of the Model

In the model being described in this project, four CNNs have been trained. They are:

- N_{VC} : classifies a image into a vowel (V type) or a consonant (C or CV type).
- N_{Vow} : if N_{VC} classifies the image as a vowel, it identifies the vowel in the image.
- N_{con1} : if N_{VC} classifies the image as a consonant, it identifies the base consonant in the image.
- N_{con2} : if N_{VC} classifies the image as a consonant, it identifies the vowel modifier in the image.

When an image is given to the model, it follows the flowchart shown in Figure 3.5, to identify the character.

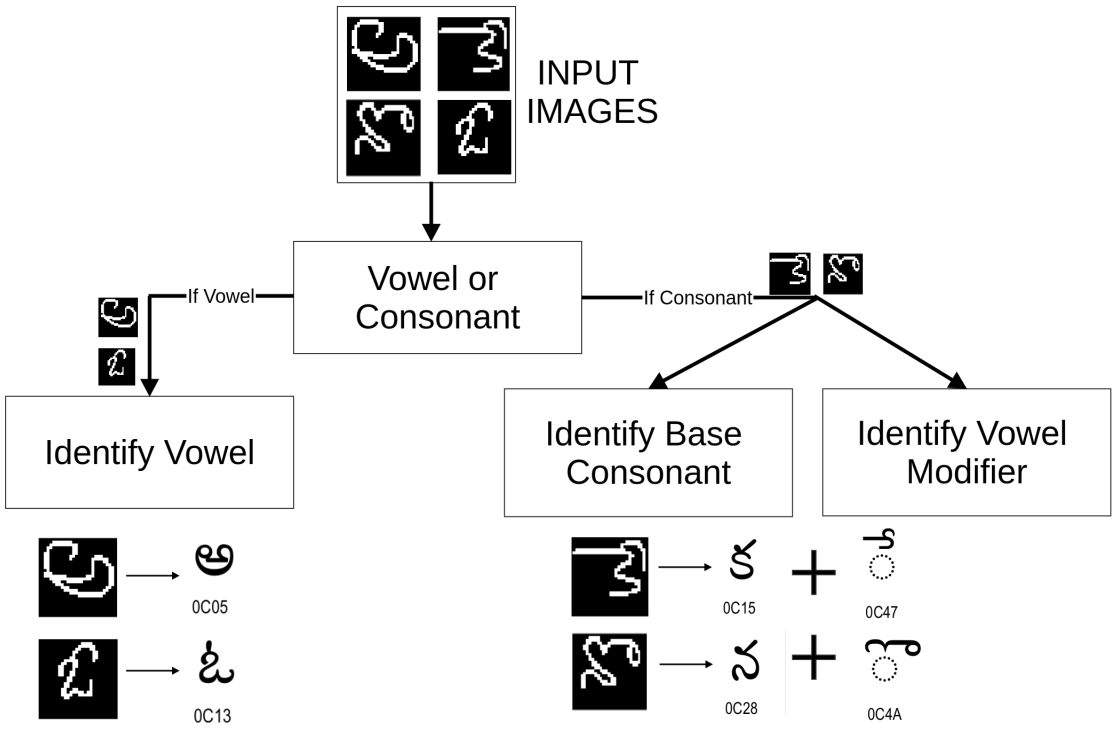

Figure 3.5: In this image, the basic flowchart of the model is described. Given some images it passes them through the first Network N_{VC}, which classifies them into vowels and consonants. The vowels are then passes through the second network N_{Vow} which identifies the vowel in the image. Similarly the consonants are sent through the networks N_{con1} and N_{con2} which identify the base consonant and the vowel modifier.

Thus four CNNs need to be trained. The way they were trained has been reported in the next section.

3.3 Training the CNNs

In each of the subsection below, the network's structure, the values of it's parameters, etc have been given. Before that, below are some of the common features of these networks

- All the networks are trained using stochastic gradient descent (SGD). Although faster methods like L-BFGS (Broyden-Fletcher-Goldfarb-Shanno algorithm using a limited) are available, SGD has been shown to be a very stable method of training. Various papers even recommend SGD over L-BFGS (Simard *et al.* (2003)). With the same inspiration, momentum was not used. Gradient descent was kept as simple as possible.
- Negative Log Likelihood criterion was used as the error criterion. It is a widely used error criterion for classification problems. This is particularly useful when the data has unbalanced training set. Thus, the last layer of each network needs to be a SoftMax layer. If inputs to the softmax layer are $\overline{z}$, then the output of the softmax layer is $\overline{x}$
$$x_i = \frac{e^{-\beta z_i}}{\sum_n e^{-\beta z_n}} \tag{3.3}$$
Thus each x_i is essentially the probability of the output being class 'i'. As x_i increases, the example's probability of being in class 'i' increases. The negative log likelihood error if the input from the Softmax layer is $\overline{x}$ and the target class is c,
$$loss(\overline{x}, c) = -\ln(\overline{x}_c) \tag{3.4}$$
The error tends to zero as $x_c \to 1$. Thus error tends to zero as the confidence of its being in the right class increases.
- Batch size used is 30 to 40. Instead of training with examples one after the other modern computers use parallel processing and train faster if the data is given in batches. This has been exploited by using massive graphic cards with thousands of threads. We have used graphic cards to accelerate training our model. In the Figures 3.6 and Figure 3.7 we compare the average time an example takes when trained in batches.
- Regularization, both batch normalization and dropout have been been used.

The structure of each of the networks has been presented in Tables 3.2, 3.3, 3.4 and 3.5.

The way to train CNNs is heavily inspired from the paper Simard *et al.* (2003).

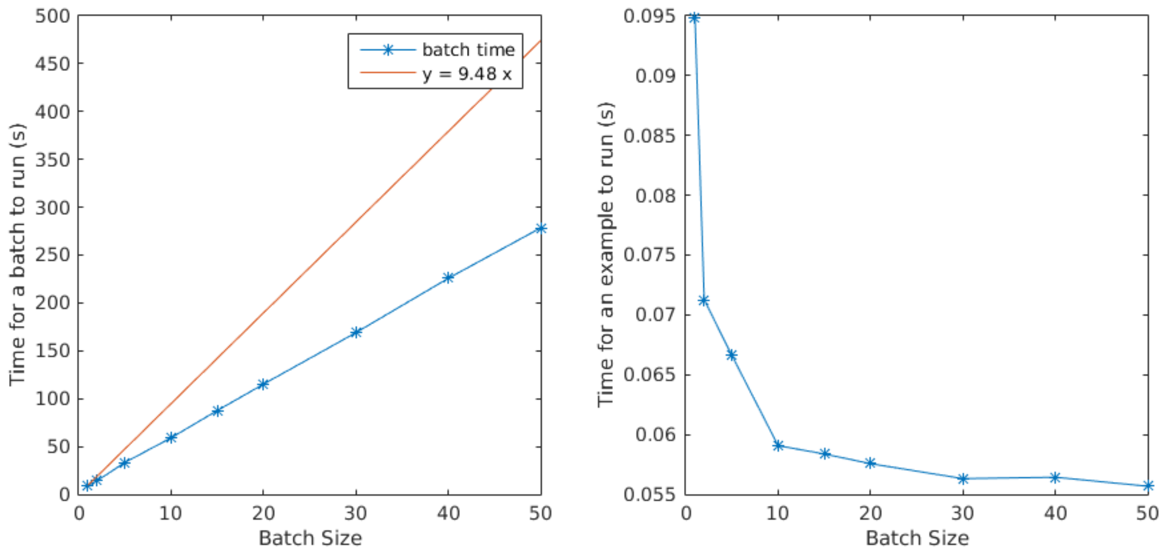

Figure 3.6: The graph on left shows the time it takes for 100 batches to run as the batch size is changed from 1 to 50 on a ordinary processor (Intel i5, 4 core, 4GiB Ram). The graph on the right shows the effective time individual examples take to run.

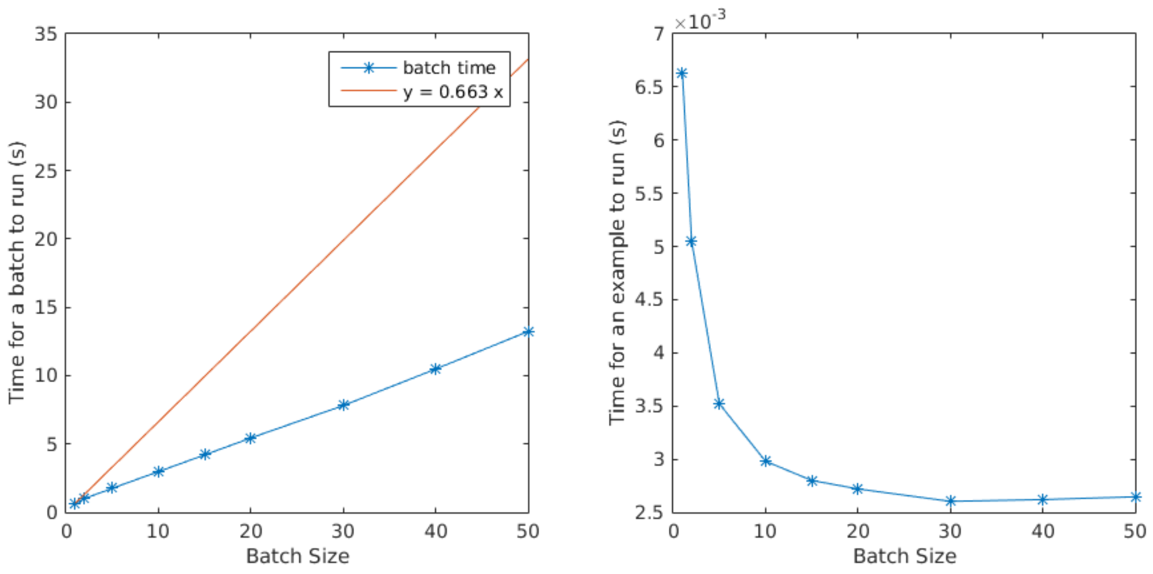

Figure 3.7: The graph on left shows the time it takes for 100 batches to run as the batch size is changed from 1 to 50 on a ordinary processor (NVIDIA GeForce GT 630M, 1GiB Ram). The graph on the right shows the effective time individual examples take to run.

Sl.No.	Layer Type and Parameters
1	Convolution Layer with 64 kernels, of size 4×4, with a step size 1×1
2	Batch Normalization (Regularisation Layer)
3	ReLU (Activation Layer)
4	Max Pooling over 2×2 neighbourhood with a step size of 2×2
5	Convolution Layer with 128 kernels, of size 4×4, with a step size 1×1
6	Batch Normalization (Regularisation Layer)
7	ReLU (Activation Layer)
8	Max Pooling over 2×2 neighbourhood with a step size of 1×1
8	Reshape the 128 node with 8×8 images into a single 8192×1 matrix
10	Linear Fully connected layer from 8192 nodes to 200 nodes
11	ReLU (Activation Layer)
12	Linear Fully connected layer from 200 nodes to 2 nodes
13	SoftMax layer

Table 3.2: The structure of the first CNN N_{VC}, which identifies whether the image contains a vowel or a consonant.

Sl.No.	Layer Type and Parameters
1	Convolution Layer with 32 kernels, of size 3×3, with a step size 1×1
2	Batch Normalization (Regularisation Layer)
3	ReLU (Activation Layer)
4	Dropout with probability 0.3 (Regularisation Layer)
5	Max Pooling over 2×2 neighbourhood with a step size of 2×2
6	Convolution Layer with 64 kernels, of size 3×3, with a step size 1×1
7	Batch Normalization (Regularisation Layer)
8	ReLU (Activation Layer)
9	Dropout with probability 0.3 (Regularisation Layer)
10	Max Pooling over 2×2 neighbourhood with a step size of 1×1
11	Reshape the 64 nodes with 10×10 images into a single 6400×1 matrix
12	Linear Fully connected layer from 6400 nodes to 200 nodes
13	ReLU (Activation Layer)
14	Linear Fully connected layer from 200 nodes to 16 nodes
15	SoftMax layer

Table 3.3: The structure of the second CNN N_{Vow}, which identifies the vowel .

Sl.No.	Layer Type and Parameters
1	Convolution Layer with 60 kernels, of size 4×4, with a step size 1×1
2	Batch Normalization (Regularisation Layer)
3	ReLU (Activation Layer)
4	Dropout with probability 0.25 (Regularisation Layer)
5	Max Pooling over 2×2 neighbourhood with a step size of 2×2
6	Convolution Layer with 120 kernels, of size 4×4, with a step size 1×1
7	Batch Normalization (Regularisation Layer)
8	ReLU (Activation Layer)
9	Dropout with probability 0.25 (Regularisation Layer)
10	Max Pooling over 2×2 neighbourhood with a step size of 1×1
11	Reshape the 120 nodes with 8×8 images into a single 7680×1 matrix
12	Linear Fully connected layer from 7680 nodes to 400 nodes
13	$\tanh()$ (Activation Layer)
14	Linear Fully connected layer from 400 nodes to 36 nodes
15	SoftMax layer

Table 3.4: The structure of the Third CNN N_{con1}, which identifies the base consonant.

Sl.No.	Layer Type and Parameters
1	Convolution Layer with 60 kernels, of size 4×4, with a step size 1×1
2	Batch Normalization (Regularisation Layer)
3	ReLU (Activation Layer)
4	Dropout with probability 0.25 (Regularisation Layer)
5	Max Pooling over 2×2 neighbourhood with a step size of 2×2
6	Convolution Layer with 120 kernels, of size 4×4, with a step size 1×1
7	Batch Normalization (Regularisation Layer)
8	ReLU (Activation Layer)
9	Dropout with probability 0.25 (Regularisation Layer)
10	Max Pooling over 2×2 neighbourhood with a step size of 1×1
11	Reshape the 120 nodes with 8×8 images into a single 7680×1 matrix
12	Linear Fully connected layer from 7680 nodes to 400 nodes
13	ReLU (Activation Layer)
14	Linear Fully connected layer from 400 nodes to 15 nodes
15	SoftMax layer

Table 3.5: The structure of the fourth CNN N_{con2}, which identifies the vowel modifier.

CHAPTER 4

Results

As discussed previously, four CNNs are trained for the overall model to work. The performance of these networks individually and the performance of the model after they are made to work together has been discussed in the following sections.

4.1 Individual Network Performance

The individual performance of each of the CNNs has been reported.

4.1.1 N_{VC} : Vowel or Consonant

Trained on a data set with 30000 images (20000 consonant images and 2000 vowel images shown five times in each epoch). The testing data has 1000 images, 808 consonant images and 182 vowel images. It classifies the data into 2 classes, vowel and consonant. We report an accuracy of 99.80% on the train dataset and 98% on the test dataset

Dataset	Size of Dataset	Number of Correct Classifications	Percentage Accuracy	Precision	Recall	F1 score
Train Dataset	30000	29941	99.803	0.9973	0.9883	0.9978
Test Dataset	1000	980	98	0.94	0.9879	0.9664

Table 4.1: The accuracy, precision, recall and F1 score of the first CNN which classifies into vowel or consonant.

4.1.2 N_{Vow}: Identify Vowel

Trained on a data set with 2000 vowel images. The testing data has 300 vowel images. It classifies the data into 16 classes. We report an accuracy of 99.8% on the train dataset and 96.3% on the test dataset. Given below are table of individual class accuracies of the test and training datasets.

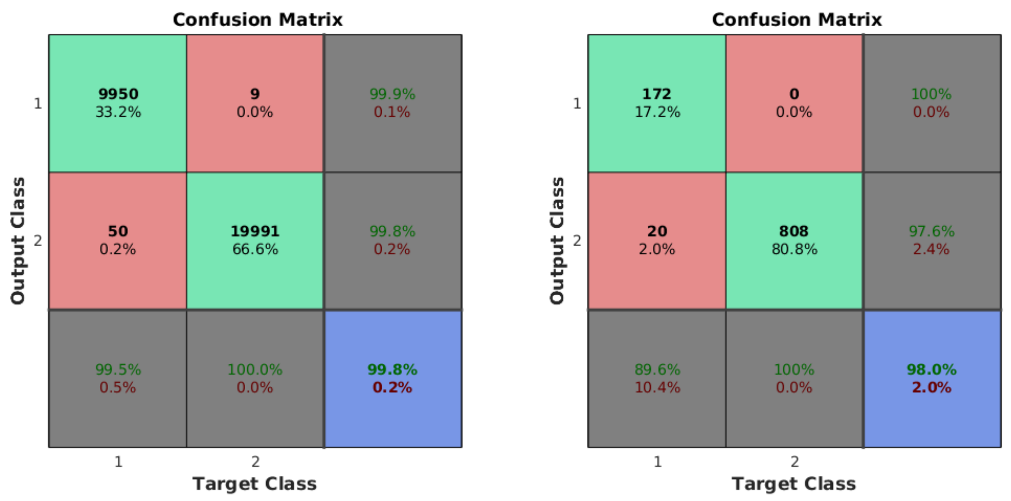

Figure 4.1: The confusion matrices of the the train and test datasets for the first CNN, which identifies whether the image contains a vowel or a consonant.

Dataset	Size of Dataset	Number of Correct Classifications	Percentage Accuracy	Precision	Recall	F1 score
Train Dataset	2000	1996	99.8	0.9982	0.9882	0.9982
Test Dataset	300	289	96.33	0.965	0.966	0.965

Table 4.2: The accuracy, precision, recall and F1 score of the second CNN which identifies the vowel.

4.1.3 N_{con1}: Identify Base Consonant

Trained on a data set with 20000 consonant images. The testing data has 1000 consonant images. It classifies the data into 36 classes shown in the table **NUMBER**. We report an accuracy of 99.7% on the train dataset and 96.2% on the test dataset.

Dataset	Size of Dataset	Number of Correct Classifications	Percentage Accuracy	Precision	Recall	F1 score
Train Dataset	20000	19948	99.74	0.9978	0.9888	0.9926
Test Dataset	1000	962	96.2	0.9608	0.9618	0.9608

Table 4.3: The accuracy, precision, recall and F1 score of the third CNN which identifies the base Consonant.

4.1.4 N_{con2}: Identify Vowel Modifier

Trained on a data set with 20000 consonant images. The testing data has 1000 consonant images. It classifies the data into 15 classes shown in the table. We report an accuracy of 98.95% accuracy on the train dataset and 92.5 % accuracy on the test dataset.

Dataset	Size of Dataset	Number of Correct Classifications	Percentage Accuracy	Precision	Recall	F1 score
Train Dataset	20000	19791	98.95	0.9881	0.9877	0.9877
Test Dataset	1000	925	92.5	0.9231	0.9292	0.9237

Table 4.4: The accuracy, precision, recall and F1 score of the fourth CNN which identifies the vowel modifier

4.2 Overall Model Performance

By testing the network run over a dataset of 1400 images the accuracy of the model was determined. The model made a total of 135 mistakes in all. Therefore the overall accuracy of the model is reported as 90.3% accuracy.

CHAPTER 5

Conclusions

We present a model which recognises Telugu handwritten characters using convolutional neural networks. We train four networks which exhibit accuracies of 98%, 96.3%, 96.2% and 92.5% on unseen test images. This overall model performs with a 90% accuracy which is comparable to other works in the same area. However, this is still very low when compared to human level performance. The reason for this that the language model itself is far too complex and the variation in writing the characters far too large.

5.1 Future Work

The work that has been done on this project can be extended to cover CCV and CCCV type characters using a similar model. These were not considered in this project for simplicity of the project. Also the same training with higher resolution images might increase the training time and load on the system, but might have a positive impact by increasing the identification accuracy.

The model can also be trained so that it continuously takes images from a line and recognizes it. This will be similar to how human read Telugu.

Although this project is done for the Telugu language model, a similar system can be extended to similar languages like Kannada and also other languages like Tamil and Malayalam. Other deep learning techniques like Recurrent Neural Networks should be tried for a more continuous mode of recognition.

APPENDIX A

Torch

The platform used to create and train convolutional neural networks is Torch7 (Collobert *et al.* (2011)). It is a computing framework with a huge number of libraries to create various type of neural networks like convolutional neural networks, recurrent neural networks, LSTMs, etc. It is different from other similar platforms by putting support for GPUs first. It is based on lua (Ierusalimschy *et al.* (1996)), a fast and easy scripting language, and C/CUDA . Torch is being widely used within several companies like Facebook, Google and Twitter.

REFERENCES

1. **Achanta, R.** and **T. Hastie** (2015). Telugu ocr framework using deep learning. *arXiv preprint arXiv:1509.05962*.

2. **Collobert, R.**, **K. Kavukcuoglu**, and **C. Farabet**, Torch7: A matlab-like environment for machine learning. *In BigLearn, NIPS Workshop*, EPFL-CONF-192376. 2011.

3. **Hinton, G. E.**, **N. Srivastava**, **A. Krizhevsky**, **I. Sutskever**, and **R. R. Salakhutdinov** (2012). Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*.

4. **Ierusalimschy, R.**, **L. H. De Figueiredo**, and **W. Celes Filho** (1996). Lua-an extensible extension language. *Softw., Pract. Exper.*, **26**(6), 635–652.

5. **Ioffe, S.** and **C. Szegedy** (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*.

6. **Nair, V.** and **G. E. Hinton**, Rectified linear units improve restricted boltzmann machines. *In Proceedings of the 27th International Conference on Machine Learning (ICML-10)*. 2010.

7. **Pal, U.**, **N. Sharma**, **T. Wakabayashi**, and **F. Kimura**, Handwritten numeral recognition of six popular indian scripts. *In Document Analysis and Recognition, 2007. ICDAR 2007. Ninth International Conference on*, volume 2. IEEE, 2007.

8. **Rajashekararadhya, S.** and **P. V. Ranjan** (2008). Efficient zone based feature extration algorithm for handwritten numeral recognition of four popular south indian scripts. *Journal of Theoretical & Applied Information Technology*, **4**(12).

9. **Rajkumar, J.**, **K. Mariraja**, **K. Kanakapriya**, **S. Nishanthini**, and **V. Chakravarthy**, Two schemas for online character recognition of telugu script based on support vector machines. *In Frontiers in Handwriting Recognition (ICFHR), 2012 International Conference on*. IEEE, 2012.

10. **Simard, P. Y.**, **D. Steinkraus**, and **J. C. Platt**, Best practices for convolutional neural networks applied to visual document analysis. *In null*. IEEE, 2003.

11. **Vikram, C.**, **C. S. Bindu**, and **C. Sasikala** (). Handwritten character recognition for telugu scripts using multi layer perceptrons (mlp).