HANDWRITTEN CHARACTER RECOGNITON OF TAMIL FONT

A Project Report Submitted

for the Course

MA498 Project I

by

Aditya Raj

(Roll No. 170123004)

Mayank Saharan

(Roll No. 170123033)

to the

DEPARTMENT OF MATHEMATICS

INDIAN INSTITUTE OF TECHNOLOGY GUWAHATI

GUWAHATI - 781039, INDIA

November 2020

CERTIFICATE

This is to certify that the work contained in this project report entitled "Handwritten character recognition of Tamil font" submitted by Aditya Raj (Roll No: 170123004) and Mayank Saharan (Roll No: 170123033) to the Department of Mathematics, Indian Institute of Technology Guwahati towards partial requirement of Bachelor of Technology in Mathematics and Computing has been carried out by him/her under my supervision.

It is also certified that this report is a survey work based on the references in the bibliography.

OR

It is also certified that, along with literature survey, a few new results are established/computational implementations have been carried out/simulation studies have been carried out/empirical analysis has been done by the student under the project.
Turnitin Similarity: 25 %

Guwahati - 781039 (Prof. Natesan Srinivasan)
November 2020 Project Supervisor

ABSTRACT

Handwritten text recognition has a noteworthy role in the fields of business, healthcare and preserving cultural heritage. There are lots of text available for offline handwritten character recognition for scripts all around the world. Although, not much can be found for old Indian scripts like Tamil. Tamil is not just the official language of Indian State of Tamil Nadu but also of countries Singapore and Sri Lanka. Tamil speaking diaspora can be found all over the world. Tamil is also one of the oldest languages of the world. Hence, it becomes important to develop handwritten character recognition system for such a rich, and generally used language.

We intend to use machine learning approaches to develop OCR(Optical Character Recognition System) for Tamil scipt. We propose to use CNN models of neural networks. We begin by pre-processing the input images. Our goal is to develop a CNN model for the purpose. The developed model gives an accuracy of . We then run the same model for Devnagri Script and compare the results . We also aim to experiment with different hyper-parameters and examine the changes in result.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Tamil Literature

One of the ancient Indian languages, Tamil, commonly used in the southern Indian state of Tamil Nadu, Malaysia, Srilanka. A very interesting fact about Tamil language is its having syllables to every sound pronounced. Syllable is the smallest unit of the Tamil script. Tamil script comprises of eighteen(18) consonants, twelve(12) vowels and one(1) special character(Ayudha Ezhuthu). Number of characters required to express a word in Tamil language is very minute.

Combination of these vowels & consonants (Figure 1.1) make 216 compound characters, hence a total of 247 characters(31+216). 5 characters are also borrowed from sanskrit adding another 60 compound characters and hence 307 total characters (Figure 1.2) .The representation of the complete Tamil character set is shown in Figure 1.3.

There are two forms of Handwritten Character Recognition (HCR): Offline and Online. Online method involves translation of the input given by a digital device referring certain co-ordinates using which the character recognition

can be done, while in the offline method uses images are scanned which are then translated to different characters based on the model. The major challenge of HCR is the disparity

Vowels	Consonants	Consonant + vowels	Compound Characters
அ	க்	க்+அ	க
ஆ	ங்	க்+ஆ	கா
இ	ச்	க்+இ	கி
ஈ	ஞ்	க்+ஈ	கீ
உ	ட்	க்+உ	கு
ஊ	ண்	க்+ஊ	கூ
எ	த்	க்+எ	கெ
ஏ	ந்	க்+ஏ	கே
ஐ	ப்	க்+ஐ	கை
ஒ	ம்	க்+ஒ	கொ
ஓ	ய்	க்+ஓ	கோ
ஔ	ர்	க்+ஔ	கௌ
	ல்		
	வ்		
	ழ்		
	ள்		
	ற்		
	ன்		

Figure 1.1: Combination of Consonants & Vowels of Tamil Script.

in the writing patterns of different people involved in preparing the dataset, since the handwriting of a single person can be dissimilar at different input times.

Offline HCR used traditional ML methods for a long period of time. A typical ML way of HCR is done as follows

1. Pre-processing
2. Segmentation

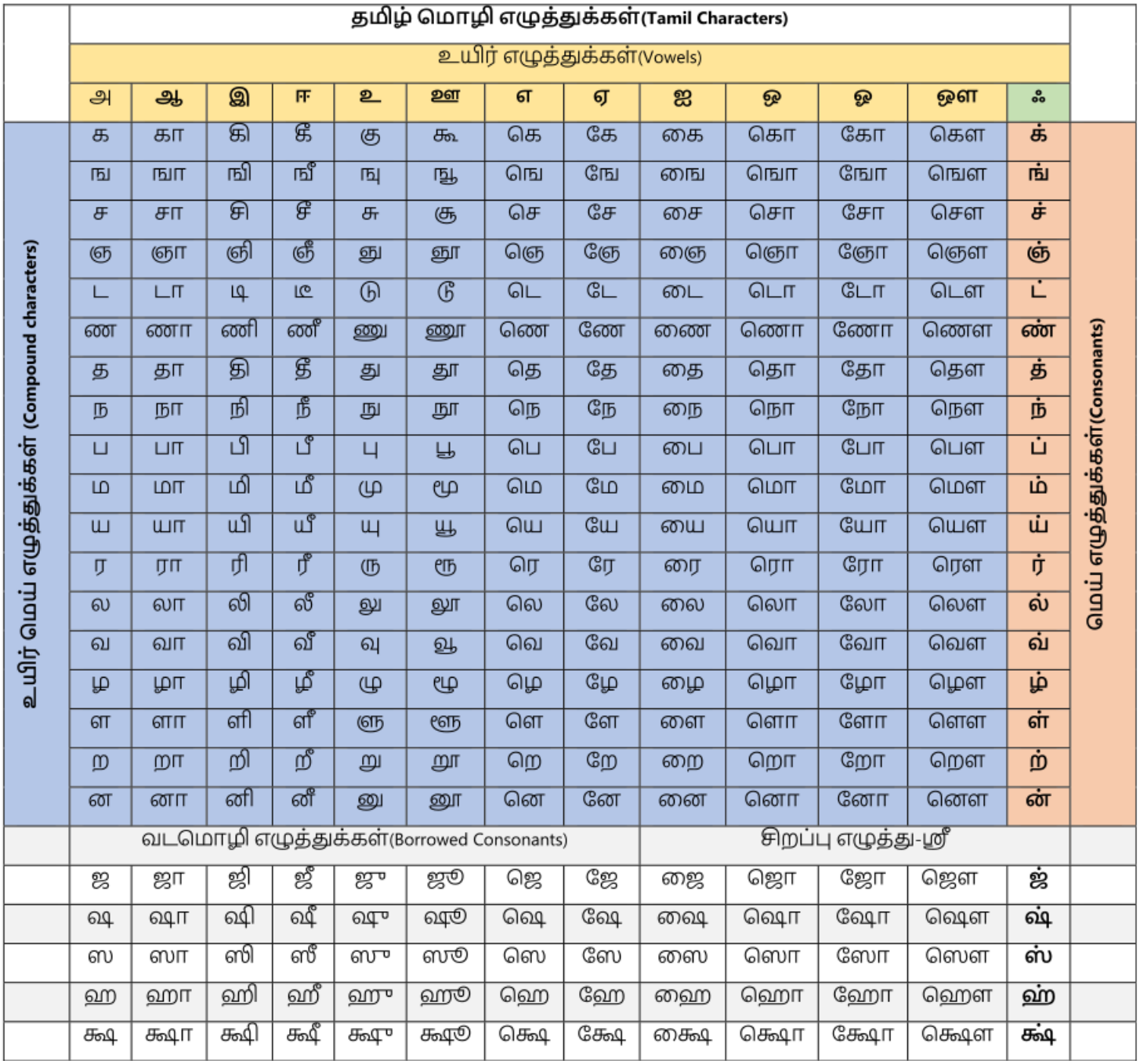

Figure 1.2: The Entire Tamil script character set.

3. Feature extraction
4. Classifying.

அ	ஆ	இ	ஈ	உ	ஊ	எ	ஏ	ஐ	ஒ	ஓ	ஔ			
0	1	2	3	4	5	6	7	8	9	10	11			
க	ங	ச	ஞ	ட	ண	த	ந	ப	ம	ய	ர	ல	வ	ழ
12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
ள	ற	ன	ஸ	ஷ	ஜ	ஹ	க்ஷ	கி	ஙி	சி	ஞி	டி	ணி	தி
27	28	29	30	31	32	33	34	35	36	37	38	39	40	41
நி	பி	மி	யி	ரி	லி	வி	ழி	ளி	றி	னி	ஸி	ஷி	ஜி	ஹி
42	43	44	45	46	47	48	49	50	51	52	53	54	55	56
க்ஷி	கீ	ஙீ	சீ	ஞீ	டீ	ணீ	தீ	நீ	பீ	மீ	யீ	ரீ	லீ	வீ
57	58	59	60	61	62	63	64	65	66	67	68	69	70	71
ழீ	ளீ	றீ	னீ	ஸீ	ஷீ	ஜீ	ஹீ	க்ஷீ	கு	ஙு	சு	ஞு	டு	ணு
72	73	74	75	76	77	78	79	80	81	82	83	84	85	86
து	நு	பு	மு	யு	ரு	லு	வு	ழு	ளு	று	னு	கூ	ஙூ	சூ
87	88	89	90	91	92	93	94	95	96	97	98	99	100	101
ஞூ	டூ	ணூ	தூ	நூ	பூ	மூ	யூ	ரூ	லூ	வூ	ழூ	ளூ	றூ	னூ
102	103	104	105	106	107	108	109	110	111	112	113	114	115	116
ா	ி	ீ	ு	ஸ்ரீ	ூ	ெ	ே	ை	க்ஷ	ஸு	ஷு	ஜு	ஹு	க்ஷு
117	118	119	120	121	122	123	124	125	126	127	128	129	130	131
க்	ங்	ச்	ஞ்	ட்	ண்	த்	ந்	ப்	ம்	ய்	ர்	ல்	வ்	ழ்
132	133	134	135	136	137	138	139	140	141	142	143	144	145	146
ள்	ற்	ன்	ஸ்	ஷ்	ஜ்	ஹ்	க்ஷ்	ஃ						
147	148	149	150	151	152	153	154	155						

Figure 1.3: Dataset 'hpl-tamil-iso-char' with labels

Firstly, an offline Handwritten character recognition system is trained by scanning images of different characters and later(testing phase) when we input a new scanned image, the HCR system should be able to associate it with some characters with certain precision. Some of the practical applications of HCR include mail sorting in post offices, check reading in banks, digitisation of handwritten form conversions and document conversions.

Digitisation of a hand-written document includes involving converting the gray-scale or coloured image to a standard binary image format, extraction of the foreground text(if present from background texture), removal of noise, line separation, segmentation of words in these separated lines, segmentation of the characters in these selected words and recognizing the selected characters.Although, presented work engages only on the off-line isolated hand-written Tamil characters, recognising the deep learning(DL) approach.

Researchers have set benchmark for languages such as Hangul, Chinese, Arabic,Japanese using corresponding standardized data-sets. The Chinese rely on National Laboratory of Pattern Recognition (NLPR) & Institute of Automation of Chinese Academy Sciences (CASIA) for the promotion of the research activities related to Character recognition of chinese script with data-sets including CASIA-HWDB & CASIA-OLHWDB for 3755 classes. Arabic use data-sets AHCD&OIHADB for 28 classes. Japanese rely on the Electrotechnical Laboratory (ETL) using the recent taken data-set ETL9B for 3036 classes. However for Tamil script, HPLabs, India developed a data-set 'hpl-tamil-iso-char' for 156 classes,which was not used by many people. In "Vijaya Raghavan"([17]), only 34 classes were used."Shanthi and Duraiswamy"[14] was only successful in recognising 34 classes using their own data.In "Bhattacharya and Ghosh"([1]) used all the classes with grouping method.Japanese & Chinese scripts have got the best results with "the state of the art CNNs". The ultimate objective of this project work is to set a benchmark for HPLabs-Tamil character dataset.

1.2 Recent Work

Most of the work done on Tamil used traditional approaches, though widely for HCR in many scripts for languages such as English, Chinese,Arabic etc. DL approaches were used. A typical approach for HTCR using traditional ML techniques would include pre- processing, character segmentation,feature extraction, classifying and then recognising the characters. in "Duraiswamy & Shanthi"[14] proposed a model with extracted features are pixel densities and Support Vector Machine(SVM) classifier used for classifying 106 classes , with 82.04 percent accuracy for 34 characters.Feature extraction us-

ing wavelet transform and Back-propagation neural network , with "89 percent" accuracy by "Jose and Wahi"[10]. "Ravichandran & Sureshkumar"[15] extracted the features from each character glyph with various attributes then classified them using SVM, Fuzzy network, RCS algorithm and Radial basis function & Self Organizing Maps (SOM). "Bhattacharya & Ghosh"[1] proposed a 2-stage recognition method using an unsupervised clustering to group the character classes in first stage, second stage using a supervised classifier for character recognition,achieving an accuracy of "89.66 percent". A work done by "Vijayaraghavan & Sra"[17], Using CNN,with an accuracy of "94.4 percent" with 35 classes. HCR of chinese script has been successful using Deep convolutional neural networks(DCNN). "Ciregan et al.(2012)"[3] tested the Chinese character dataset,Chinese Academy of Sciences,Institute of Automation(CASIA) which had 300 samples of 3755 characters as an experiment "Cirecsan & Meier"[4] worked on the same data-set(CASIA) with eight different network architectures, achieving near human accuracy using Multi Column Deep Neural Network.Many standard datasets for Chinese characters are available consisting of over a million characters ("Liu et al."[13])."Zhang et al."[18] proposed a new standard by combining CNN with normalization, direction decomposed feature map & adaptation layer on offline task with an accuracy of "97.37 percent"." Tsai"[16] used CNNs to recognize 3 types of handwritten Japanese language scripts namely kanji,katakana & hiragana with recorded overall classification accuracy "99.53 percent" for 1004 classes. A CNN model was developed by "Boufenar et al."[2] from scratch for OIHACDB-28 (Arabic character set) achieving an accuracy of "97.32 percent", also in another presented work they used transfer learning with an accuracy of "100 percent" for the model prepared from scratch. "El-Sawy et al. A CNN model developed by"[6] from scratch was shown as "94.9

percent" accurate and "Elleuch et al." [7] have using Deep Belief Network (DBN) architecture along with the regularization techniques, drop out and drop connect producing an error classification rate of 2.73 percent & 2.27 percent respectively.

1.3 OCR

1.3.1 Introduction

The problem of recognising optically processed characters is handled by Optical Character Recognition (OCR). Optical recognition is performed in online the machine recognizes the characters as they are inputed , whereas in offline recognition after the writing or printing has been completed .Handwritten & printed characters, both may be recognized, but the performance is directly proportional to the standard and condition of the documents taken for input. Adding more constraints to the input directly improves the performance of an OCR system.

1.3.2 OCR system components

There are several components of a typical OCR system. Figure 1.5 shows a common setup with corresponding steps. The first step is to digitise the analog document by an optical scanner. Each symbol is extracted , after the regions containing text are located through a segmentation process. The extracted symbols are then pre-processed, eliminating noise, to provide the feature extraction in the next step.

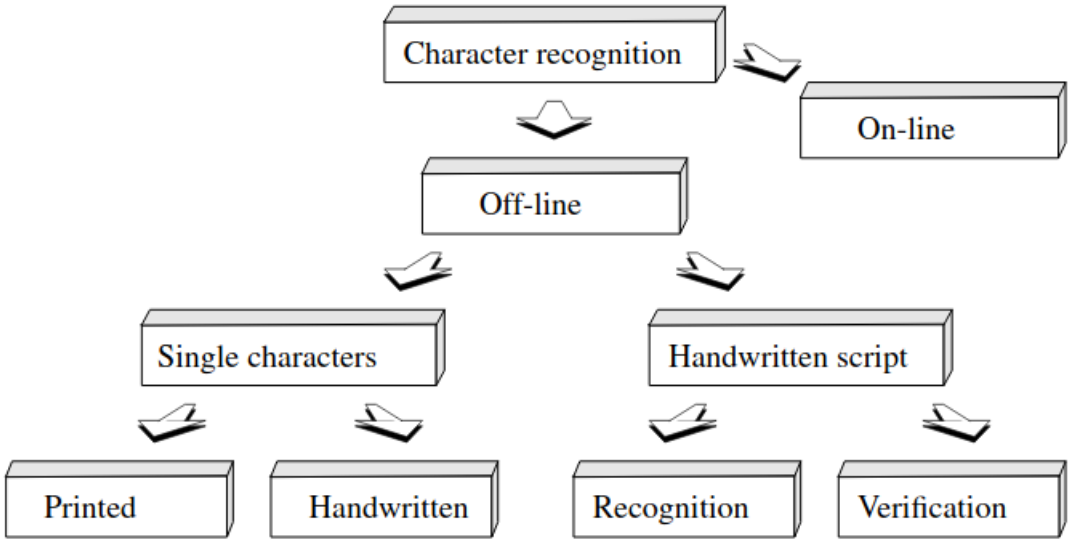

Figure 1.4: Different areas of OCR

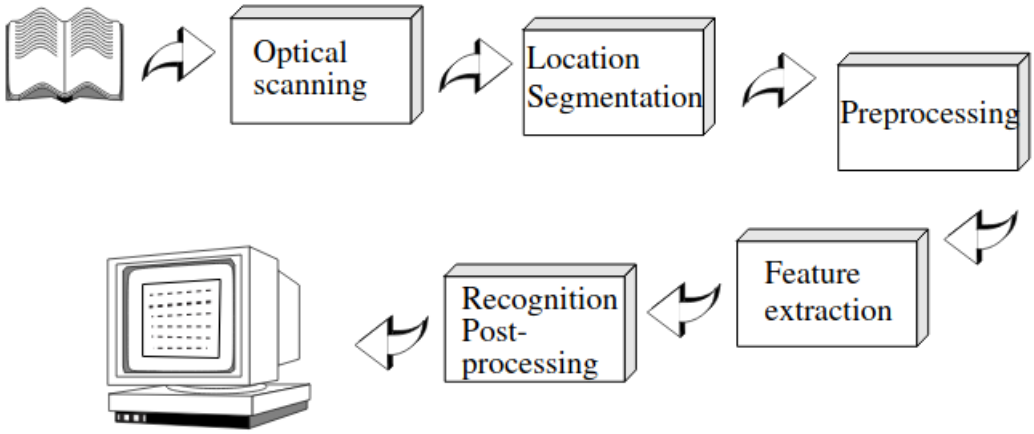

Figure 1.5: OCR system components

Chapter 2

Neural Networks

In this chapter, we discuss about the structure of neural networks. We delve deeper into the handling of the problems related to image classification and various advancements made in the field.

2.1 Artificial Neural Networks

Artificial Neural Networks are an attempt to model the information processing capabilities of human nervous system. The simplest definition of neural network is given by one of the 1st neuro-computer scientists, Dr. Robert Hecht-Nielsen 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."

2.1.1 Biological Motivation

Neurons communicate via electrical signals that are short-lived "spikes" in the voltage of the cell membrane. The inter-neuron connections are mediated

by electro-chemical junctions called **synapses**, which are located on branches of the cell referred to as **dendrites**. Each neuron is connected to thousand of neurons and simultaneously receiving and sending multitude of these *spikes*. They are summed in some way and if this sum is greater than some set threshold, then the neuron generates a voltage impulse to respond.

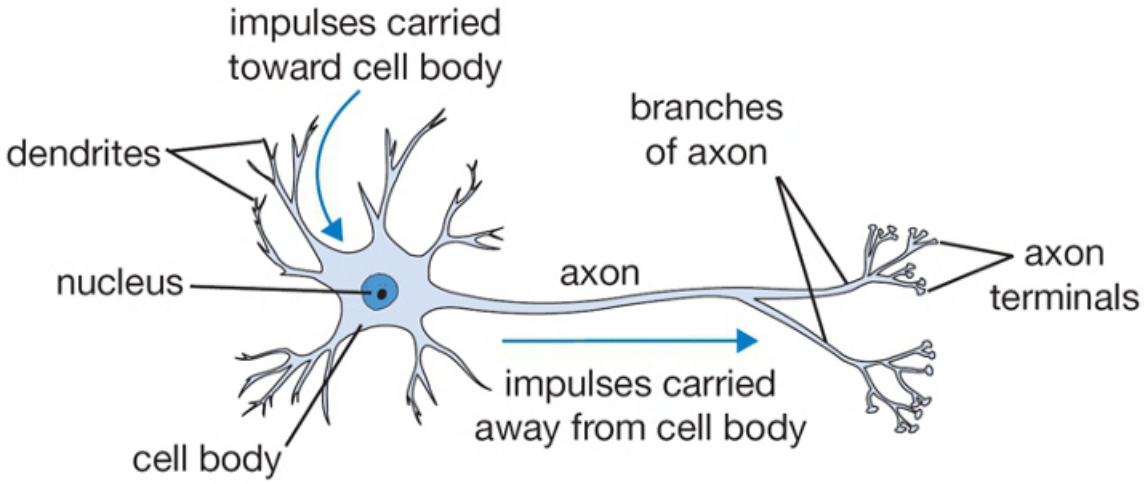

Figure 2.1: Biological Neuron.

We try to include this architecture and style of processing in neural networks. This type of system is sometimes referred to as being *connectionist* and the study of this general approach as *connectionism*. Artificial neurons or just *nodes* or *units* are the digital equivalents of biological neuron. Synapses are modelled as single numbers called **weights**. Weights can be either positive or negative. Each input is multiplied first by this weight before being sent to the equivalent of body cell. These weighted signals are summed together to get a single number called **activation** of that node. This activation value is then compared with the pre-decided threshold value; if the activation is greater than the threshold value, then the equivalent model produces a high

valued output(conventionally taken as 1), otherwise model outputs zero. The following figure explains this elementary model -

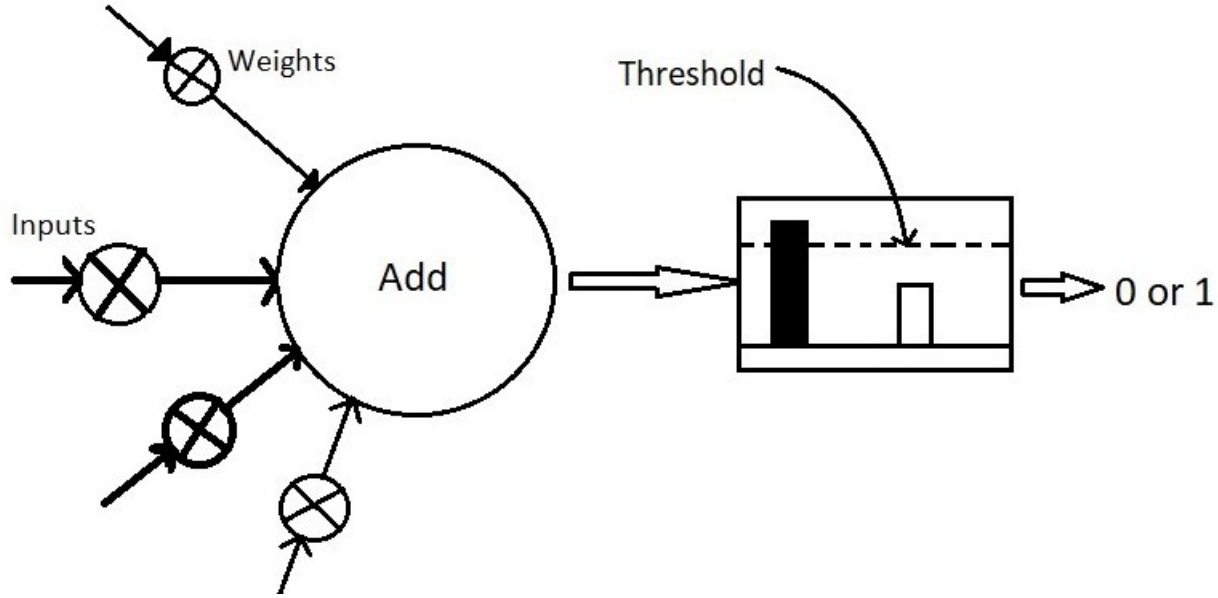

Figure 2.2: An elementary artificial neuron.

This model is called TLU - Threshold Logic Unit. Weights are represented by 'X' sign in the circles and size of the signals are represented by width of the arrows corresponding to it.

2.1.2 Neural Network Architecture

The term **network** is used to refer to any system of artificial neurons. This ranges from a single node network to network of many layers of nodes where each node in a layer is connected to all the nodes of the next layer.

Previously, we discussed about the weights can be either positive or negative. The positive weights are called **excitory** and negative ones are termed as **inhibitory**. **Activation function** is used to model the *firing rate* of the

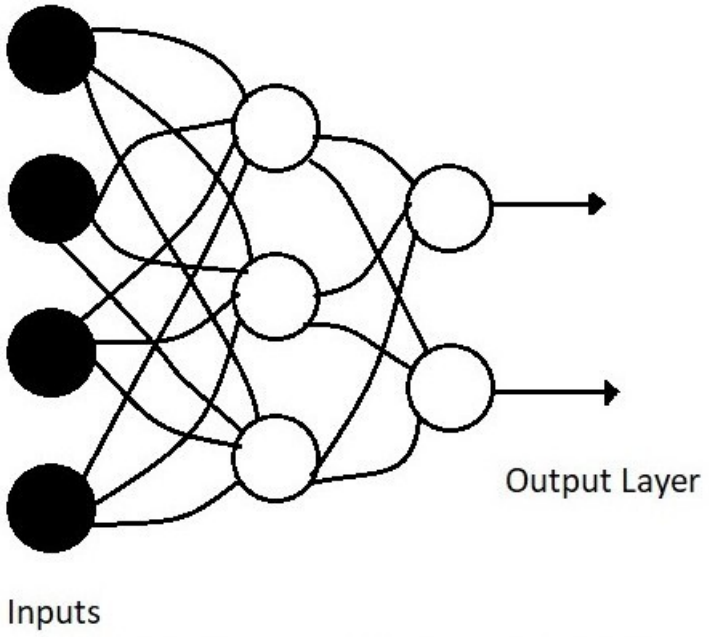

Figure 2.3: Two-Layered basic model of ANN.

neurons. A **layer** is a block of nodes arranged vertically. There are some other important features in an artificial neural network models discussed below -

- **Input Layer:** A layer of input which contains input information and pass it to the next layer. No computation is done in this layer.
- **Hidden Layer:** In Hidden layers is where computation or intermediate processing is done, computations are performed and then they transfer the weights (signals or information) from the input layer to the subsequent layer (output layer or to another hidden layer). A neural network without a hidden layer is also possible.
- **Output Layer:** We finally use an activation function mapping to our desired output format (e.g., softmax for classification, which is

discussed later).

- **Connections and Weights:** Each connection transfers the output of a *neuron* i as an input to *neuron* j. The weight assigned to this connection is W_{ij}. The subscript 'ij' represents that i is the previous layer and j is the next layer(successor layer).

- **Learning Algorithm:** Learning algorithm/rule modifies the parameters of the model in order to produce a more favored output. Learning algorithms are discussed in detail in section 2.2.6.

2.1.3 Feed-forward Neural Network

Feed-forward NN is a model in which no cycle is formed between the connections.In this network,information moves only in one direction, forward, from the input nodes, to the output nodes and through the hidden nodes (if any). There are 2 types of feed-forward neural networks -

1. **Single-Layer Perceptron :** This model doesn't contain any hidden layer and a single input neuron. Hence, it just has 2 neurons, one for input and one for output. Following illustration explains it further -

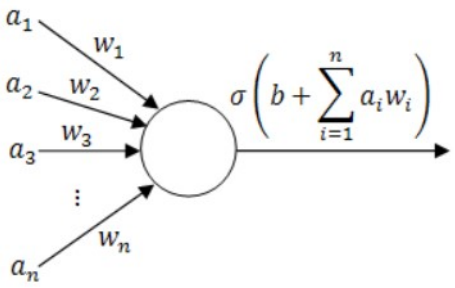

Figure 2.4: A Single-Layer Perceptron

2. **Multi-Layer Perceptron :** Network of this type contains multiple layers interconnected in a feed forward manner. Each neuron in a layer has connection with every neuron in the subsequent layer, but it strictly has no connection with any other neuron. Unit of these networks apply an activation function to calculate activation of the current layer making use of parameters and activations of the previous layer(which the current layer takes as an input). MLPs are powerful because can learn from non-linear representations. Most of the practical data is not linear in nature.

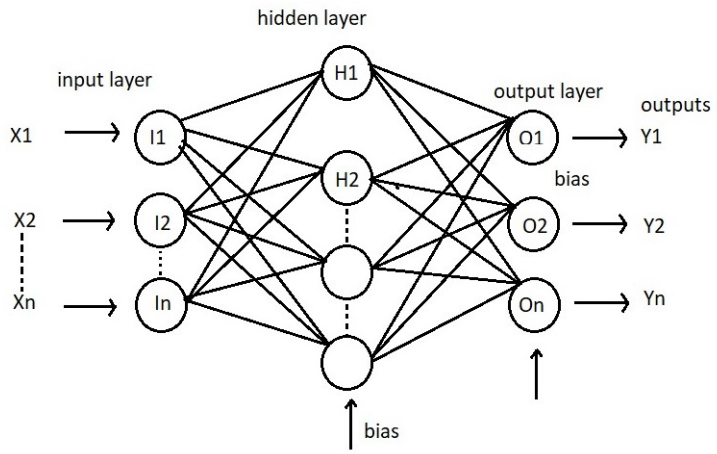

Figure 2.5: A Multi-Layer Perceptron

There are two types of learning in field of image processing - **Unsupervised Learning** & **Supervised Learning**. The learning through pre-labelled inputs,acting as *targets* is Supervised learning. For every training example there will be a set of input values (vectors) and one or more associated designated output values. Model's overall error is however reduced, through correct calculation of the output value of training example by training the model using one of the many learning algorithms.

Supervised Learning is further classified into 2 categories -

1. **Classification:** When output variable is a category like "cat", "dog", "disease", "no disease", etc.

2. **Regression:** When output variable is a real value such as "rupee", "weight", etc.

Convolutional Neural Networks comes under the *classification supervised learning*. From the input raw image vectors to the final output of the class score, the entirity of the network still expresses a single perceptive score function(the weight).The last layer contains loss functions associated with the classes, and all the theory developed for ANN apply here.

2.2 Convolutional Neural Networks

Convolutional Neural Networks, also called ConvNet or simply abbreviated as CNN are parallel to the traditional neural networks in a sense that they are also comprised of neurons which optimise themselves through self learning. CNNs are generally used in the image classification, object detection,pattern recognition etc. tasks. This field specific usage has allowed to add image-specific features in the architecture, making it more suited for such tasks.

CNN uses the concept of weight sharing, resulting in significant reduction in number of parameters. Due to reduced parameters, CNN doesn't suffer from *overfitting* and it results in improved generalization. Also, it is much easier to implement large models in CNN compared to traditional neural networks.

2.2.1 Architecture of CNN model

CNN consist of three types of layers - Convolutional Layers, Pooling Layers & Fully Connected Layers. In addition to these three, there is also an input layer where we place our inputs to the model. Stacking these layers in a particular order forms the architecture of CNN. Following is an illustration of a basic CNN model that recognises the digits from 0 to 9 -

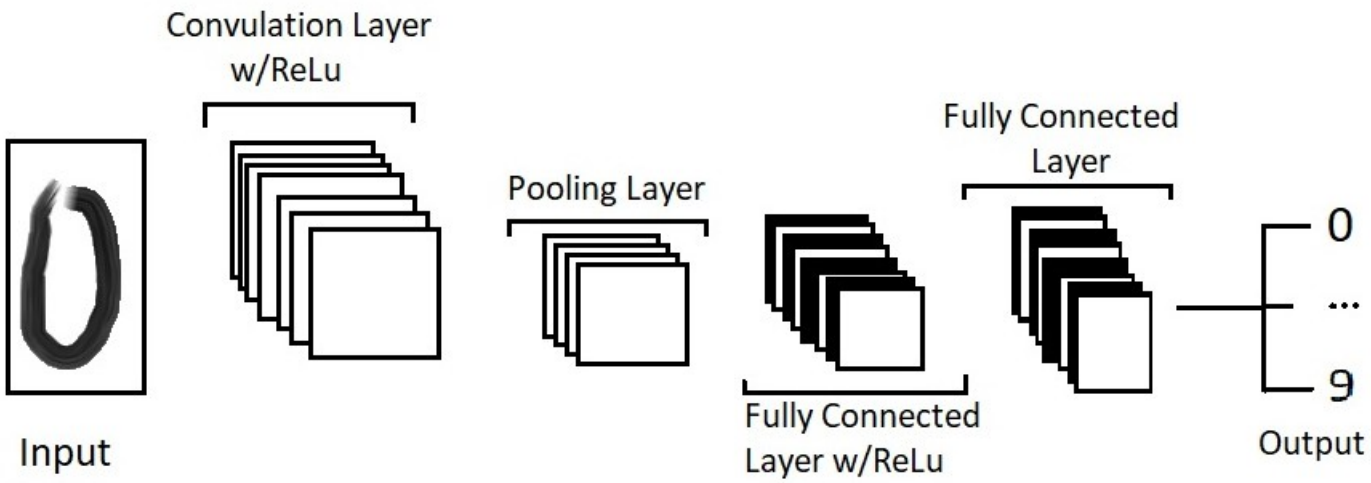

Figure 2.6: A simple CNN model with five layers.

The function of CNN can be divided into four major parts -

1. The **input** layer holds the pixel values of the image
2. The **Convolution** layer gives output of each neuron in the layer. It finds scalar product and weights and input volumes. It applies an activation function to each element which are basically activation value of previous layer.
3. The **pooling** layer further reduces the parameters within the activation. It downsamples along the spatial dimensionality.
4. **Fully Connected** Layer performs actions that are performed in the general neural network model. It produces class scores which is used for classification. To further improve the performance, we can use activation functions like *ReLu* between these layers.

2.2.2 Convolution Layer

Convolution Layer plays a key role in the operation of CNNs. An image to be classified into one of the output labels is given as an input and prediction is made based on the features extracted from the image. When the data is given to the layer, the layer convolves each filter across the spatial dimensionality of the input. This produces a 2-dimensional activation map. The weight vector which spans across all input values and is same for a particular layer is called kernel. As we move through the inputs, the scalar product is calculated for each value in the kernel. This method of sliding horizontally or vertically over values is called *convolutional* operation. These kernels form a field like structure (as seen in Figure 2.5), and hence called receptor fields.

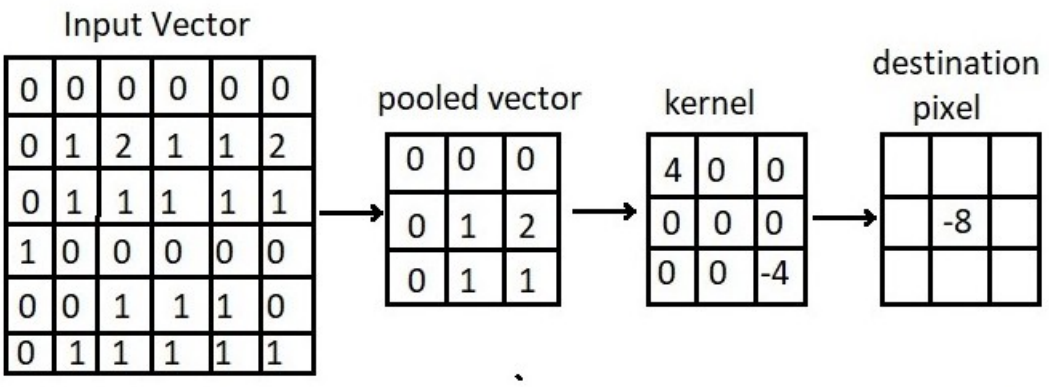

Figure 2.7: A representation of convolution layer.The centre element is placed over the input vector,then it is calculated and replaced with a weighted sum of nearby pixels and its own value.

Every kernel has an activation map which are stacked on one another, as shown in the Fig 2.5 Every neuron in the convolution layer is only connected to a small region of input. This small region as discussed earlier is called a receptor field. The magnitude of the connectivity through the depth is nearly always equal to the depth of the input.

Let a_{ij} be the activation of the location(i,j) of the next layer. It is computed after applying convolutional operation as shown below-

$$a_{ij} = \sigma((W * X)_{ij} + b)$$

Here, X is the input provided to the layer, W is the kernel which slides over the input, b is the bias and '*' denotes the convolutional operation, and σ is the function that introduces non linearity. We discuss in detail about the *convolutional operation* later in this chapter.

Convolution Layer can further optimise the output which reduces the complexity of the model. Output is optimised by changing these hyper-parameters

- **depth**, **stride** and setting **zero-padding**.

The **depth** of output volume can be manually set by changing number of neurons within the layer. Reducing depth drastically minimises the number of neurons in the model. But the downfall is that this significantly reduces the pattern recognition capabilities of the model.

Stride is the number of pixel shifts over the input layer. If we set stride to 1, we move the filters to 1 pixel at a time. This would produce large activations because of the overlappings on the receptive field. If the stride value is set to 2 then we move the filters to 2 pixels at a time. This would decrease the amount of overlapping and outputs of lower spatial dimensions.

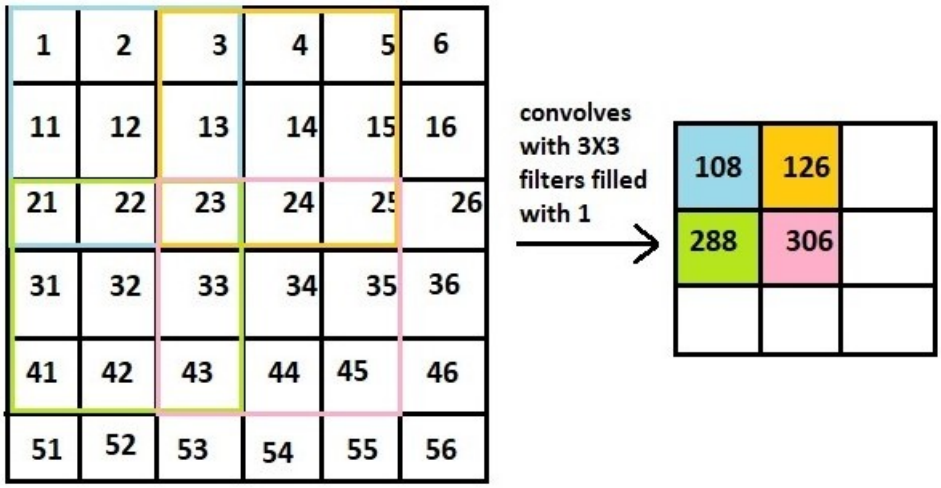

Figure 2.8: When stride is set at 2.

Zero-padding is simply the process of padding the filter with zeros so that it fits the input image. It gives further control to as to the dimensionality of the output volume.

Using the above discussed hyper-parameters - depth, stride and zero-padding, we can change the spatial dimensionality of convolution layer output. We can use the following formula -

$$\frac{(V - R) + 2Z}{S + 1}$$

Here, S is the stride, Z is the set value of zero-padding, V represents the input volume size and R is the size of receptor field. Input volume is basically height X weight X depth. Calculated result from the above formula should be an integer, otherwise the neurons will be unable to fit across the given input.

2.2.3 Pooling Layer

Pooling Layer further reduces the complexity of the model. It does so by reducing the spatial dimensionality and hence reduce the number of parameters. The exact location of a feature becomes less important once it has been detected.To perform pooling operation, a window is selected and the input elements lying in that window are passed through a pooling function Pooling layer operates on each feature map or activation map independently. It creates a new set of the same number of pooled feature maps. So, spatial pooling also called **downsampling** or subsampling reduces the spatial dimensionality of each map but retains all the important features that haven't been detected yet. Downsampling is of these three types -

1. Max Pooling

2. Average Pooling

3. Sum Pooling

The most common approach is Maximum Pooling or simply **Max Pooling**. In max pooling, we consider the kernel with maximum value in the patch selected.Generally kernels of 2×2 size are considered as a patch with stride value of 2 applied along the spatial dimensionality. This scales down the feature map to one-fourth of its original size while maintaining the depth of its actual value.

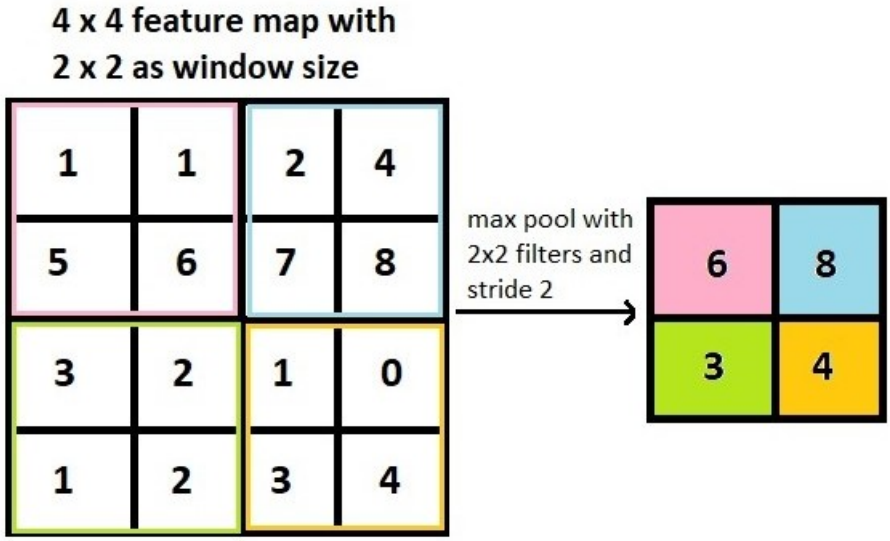

Figure 2.9: Max Pooling.

Usually stride is set to 2 and window size of kernel set to 2×2, which allow the layer to extend through the entirety of the input. When stride is set to 2, and window size 3×3, this is an example of overlapping pooling. But due to the destructive nature of pooling, with kernel size set greater than 2 generally decreases the performance of the model.

2.2.4 Fully Connected Layer

Fully connected layer is similar to conventional neural network models. The output of first two layers(convolution and pooling) is fed into the fully connected layer, and dot product of weights vector and input vector is performed to get output of this layer. But before feeding the input to fully connected layer, **flattening** of input matrix into a vector is performed.

Flattening is basically converting the matrix into a single column vector. We need to flatten the output of pooling layer into a column vector because traditional neural networks can accept vector type input only on which further dot product is performed with the weights vector. Following diagram explains flattening -

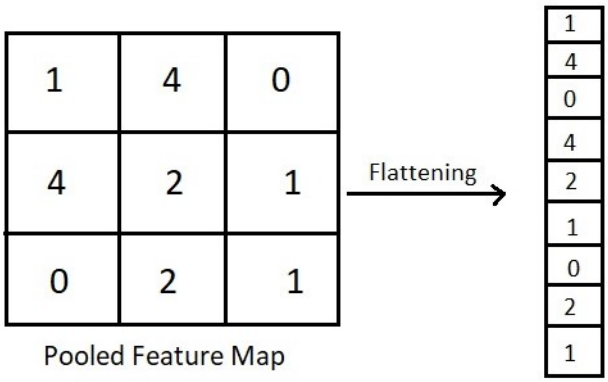

Figure 2.10: Flattening.

In fully connected layer, neurons are arranged like traditional neural networks where neurons are connected directly to neurons in the adjacent layer. **Gradient Descent** is used to reduce the cost function by estimating the cost over the whole input and update the parameters after every **epoch**. An epoch corresponds to traversing the whole data-set once. It yields global

minima but if data-set is very large, it takes lot of time, due to which this approach is now replaced by **stochastic gradient descent**. ReLu is used as an activation function between these layers to improve performance.

2.2.5 Activation Function

The purpose of activation function is to introduce non-linearity in our CNN model. This is because the real world data would want our ConvNet to learn non-negative linear values. Therefore, introduction of activation function improves performance of the model. Many non linear functions such as sigmoid, tanh, ReLu are used. Sigmoid function is the most traditional activation function. But the most used function is ReLu.

ReLu (Rectified Linear Unit) is a special implementation that combines non-linearity and rectification layers in the convolutional neural network. It is rectified because it thresholds at 0. It is a piecewise linear function defined as -

$$Y_i^{(l)} = \max(0, Y_i^{(l-1)})$$

There are three major factors why ReLu is better than *sigmoid* or *tanh* functions-

1. Calculation of partial derivative of ReLu is very easy(used in the learning algorithms).

2. If you consider training time to be one of the factors, then ReLu is faster than other traditional activation functions because it it is non-linear non-saturating.

3. ReLu doesn't allow gradients to disappear.

One of the major problem with use of ReLu is when the gradients flowing in the network are considerably large. This leads to weight updates that causes some neurons to not get activated, leading to dying ReLu problem. This can be resolved by using **Leaky ReLu**, in which if Y_i is less than 0, it is activated as ax instead of 0, where a is a small constant.

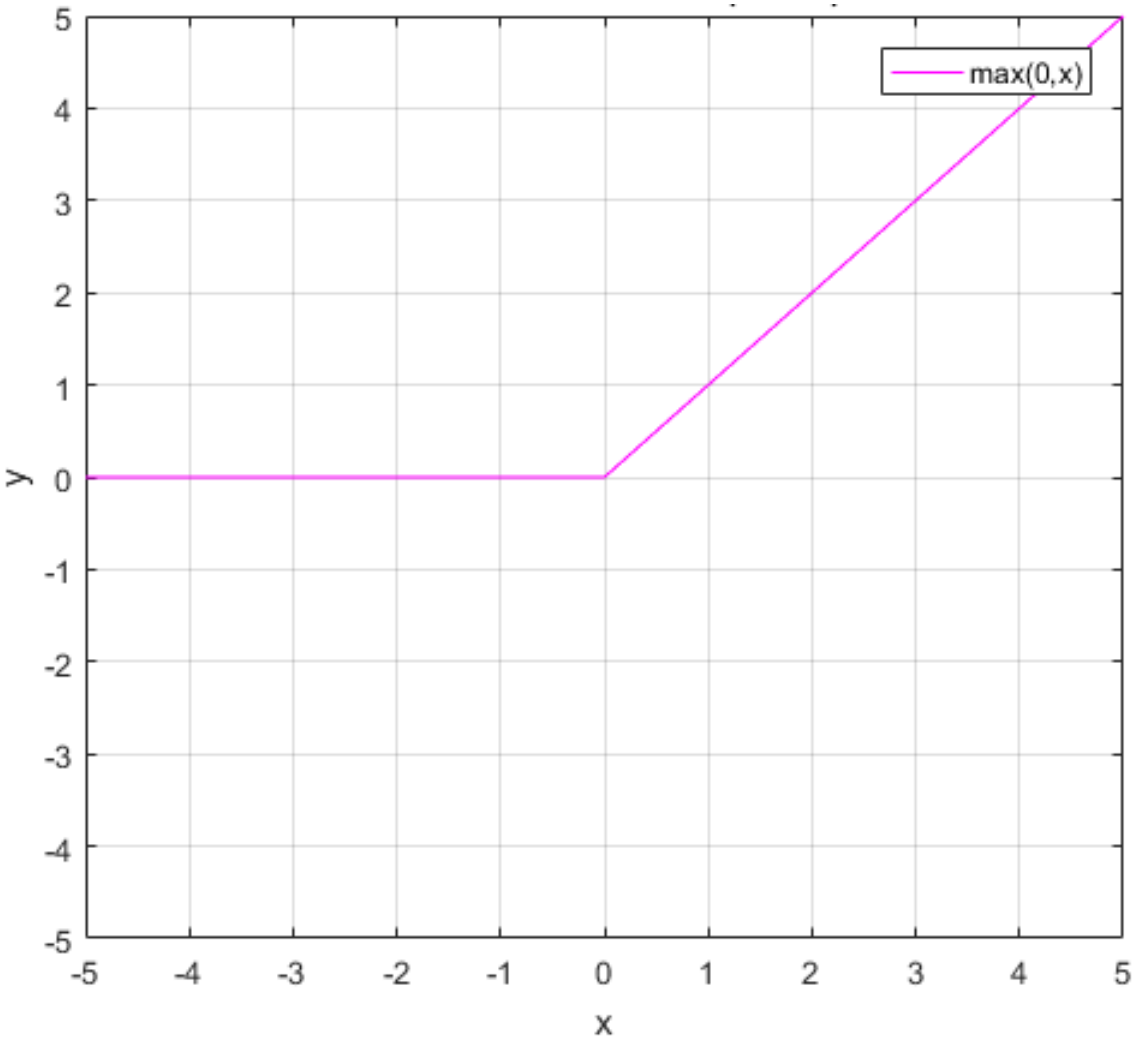

Figure 2.11: Rectified Linear Unit(ReLU).

Softmax function is special because it can handle multiple classes at once. It converts a vector of K real values to a vector of K real values that sum

to one(1). The input values can be zero,negative,positive, but the softmax converts them into values between 0 and 1,to interpret them as probabilities. We can often face with the layers in the model which output values that are not conveniently scaled and which may be difficult to work with.In such a situation, softmax function comes in handy which converts all the values to a normalized probability distribution. It's formula is given by-

$$\sigma(z)_i = \frac{e^{z_j}}{\sum_{j=1}^{K} e^{z_j}}$$

Here, K is the number of classifiers, z is the input vector to the softmax function, z_i is individual elements from z.

Since "Softmax" is a continuously differentiable function, it's possible to calculate the derivative of the loss function w.r.t. every weight in the network, for each image in the training set, allowing us to adjust the network's weights so as to reduce the loss function and make the network output closer to the desired values and improve the network's accuracy. After several iterations of training, we update the network's weights.

2.2.6 Learning Algorithms

Gradient Descent is the optimization algorithm most commonly used. It is a first-order iterative algorithm.It finds local minima in each iteration. CNN is a feed forward neural network altogether. The process of gradient descent starts by computing the outputs at every layer one by one and calculate the loss introduced on the last layer. The gradients are now back propagated to optimize the model. The gradient descent process is carried out in the following steps -

1. Perform convolution operation on the input kernels to produce feature maps. It is done using the formula-

$$C_q^l = \left(\sum_{p=1}^{n} \sum_{u=-x}^{x} \sum_{v=-x}^{x} S_p^{l-1}(i-u, j-v).k_{p,q}l(u,v) + b_q^l \right)$$

   Here, n represents number of feature maps in the previous layer, p and q are map indices for the current layer and the previous layer respectively, b is the bias and x is the filter size. S represents the sample image on which the convolutional operation is carried out. Here, we use ReLu function to introduce non-linearity.

2. After this, pooling is done. It introduces translational invariance.

$$S_q^l(i,j) = \frac{1}{4} \sum_{u=0}^{z} \sum_{v=0}^{z} C_q^l(2i-u, 2j-v)$$

   Here, z represents pool size.

3. Iterations of convolutional and pooling operations are to be performed according to the need. This causes repetition of layers.

4. The obtained outputs after these iterations are passed on to the fully connected layer. We know fully connected layers are similar to ANNs. Flattening of the output is required before passing it on to the fully connected layer for classification. We compute the output using the following equation -

$$a_{ij} = \sigma((W * X)_{ij} + b)$$

   X is the vector which is the output of previous pooling operation and W is the vector of *weights*.

5. Now, the output is passed to softmax function. $\hat{y(i)} = \frac{e^{\hat{output}}}{\sum_1^{labels} e^{\hat{output}}}$ where $\hat{output}$ is the output from fully connected layer.

6. In this step, we find loss at the final layer and calculate gradient with respect to all the learnable parameters. Loss can be calculated using **cross entropy** method. It is given by the following formula -

$$L = - \sum_{\text{no. of training parameters}} (y \log(\hat{y}) - (1 - y) \log(1 - \hat{y}))$$

   here $\hat{y}$ is the target or given output and y is the obtained output.

7. Error back propagation is done. Based on it, the parameters are updated. The first order derivative w.r.t. weight and bias are computed.

8. The whole above process is one epochs. Several epochs are executed based on the model and input.

Adam optimization (Adaptive Moment Estimation) is an optimization algorithm . It replaces the generally used stochastic gradient descent algorithm to update the weights in the network iteratively based on the training data.

Adam Optimization algorithm makes use of first and second moment of gradient to calculate learning rates for each parameter. Since, it is computationally efficient, it requires less memory and is a better algorithm in case of large data-sets. It require p_0, q_0 to be initialized to 0, where p_0 corresponds to moment vector i.e., mean,q_o corresponds to the second moment vector. Another benefit of using Adam is that the updation of parameter is completely invariant to gradient rescaling, the algorithm will converge even if objective function changes with time. But it has a disadvantage too, i.e.,

for using Adam algorithm, you need to calculate second moment vector. It increases cost of the algorithm.

Chapter 3

Proposed Approach, Experiments and Analysis

3.1 Proposed Approach

We aim to optically recognise characters of Tamil font exclusively using *Supervised Learning*. We use the approach of Convolutional Neural Networks model(CNN or simply ConvNet) to develop our model and carry out the experiments.

The data-set we have used is available online at the website of *HP Labs*. It contains around 480 samples of the 156 basic characters in Tamil font. It contains all the vowels, consonants, numerals and some but not all of *Grantha compound* characters. The Data-set available to us was in a raw form, so the first task is to filter and pre-process the data so as it can be put in our CNN model, which we made and articulated all the intricacies involved in the later sections. We also look forward to compare the results of Tamil font to that of the Devanagri Font when run on the same model and discuss about the differences in result.

3.1.1 Image Pre-processing

The input images available to us are irregular in their dimensions. Image pre-processing done in a right way can drastically improve the performance of a CNN model. We aim to implement following techniques -

1. **Image Resizing:** Image data available to us have different sizes. We aim to bring them down to a common size of 32×32. In Tamil font, the characters are very complex with lots of curves. The images are in different scale so we also need to make sure that they are brought to a common scale and while doing so, none of the curves of characters get cropped out. For this, we use the approach of **Bilinear Interpolation**.

   Bilinear Interpolation is a re-sampling method .It takes into account the distanceweighted average of the four neighbour pixel value. This weighted average is used to estimate a new pixel value. The normalization is done using bilinear interpolation.

2. **Image Smoothing:** It is done to smooth the curves of the characters. It reduces noises present in the image and produces less pixelated image. We use *PILLOW* package of python programming language, that provides us inbuilt **Box Smooth Filter** which calculates weighted mean of kernel neighbours to produce output.

3. **Colour Inversion:** All the images available to us are in gray-scale format with background set to '0'(white) and foreground to '255'(black). We invert this color scheme to check whether the accuracy of the model is affected by it or not.

The HPL dataset contains following *syllabic units*-

- isolated vowels
- isolated consonants(with inherent neutral vowel 'a')
- Consonant-Vowel combinations where the constant is modified by the vowel and is stipulated by a vowel diacritic. (e.g. - 'ku')
- Amalgamations of 2 or more consonants remoulded by a vowel like CCV, CCCV, etc.(e.g.'kshū')

The 156 characters are encoded as labels numbering from '0-155' in the same sequence in which they are encoded.

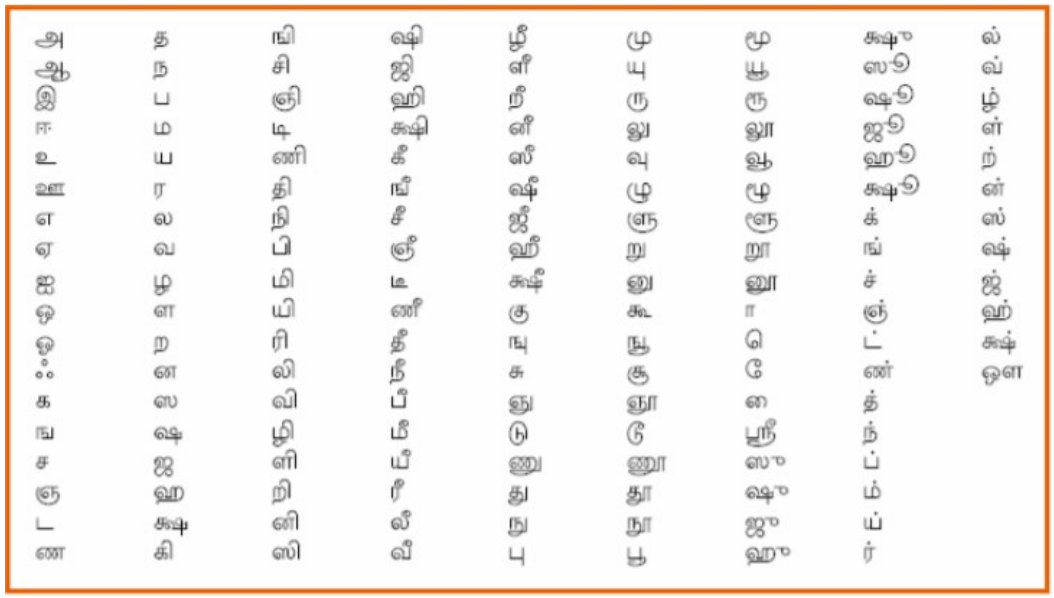

Figure 3.1: All the 156 characters for which modelling is being done. Labelling is done in a sequence across the columns.

For supervised learning approach, we are required to bifurcate the data-set into *Train* and *Test* or *Validation* data-sets. The data given is classified in terms of different users who have contributed. Hence, after pre-processing of image, we are required to split the data-set. There are total of 478 specimens of each class. We put 404 of them in the training set and remaining 74 in the validation set giving us the ratio of around 6:1. We also look forward to make changes in the ratio and discuss the changes that it causes.

3.1.2 Architecture

We aim to use CNN architecture to model our HTCR system. The programming is done on *python3* using *ananconda's Jupyter Python Notebook*. For training and testing of the model, *Keras* is used as the deep learning API which is written in python. *Tensorflow* runs in the backend. The entire training and testing of CNN model is performed on a computer machine with specifications - Windows 10- Pro 64-bit, intel i5(7300HQ)-processor with available RAM of 16 GB. Figure 3.2 shows the complete testing and training processes of the proposed HTCR (Handwritten Text Character Recognition) system.

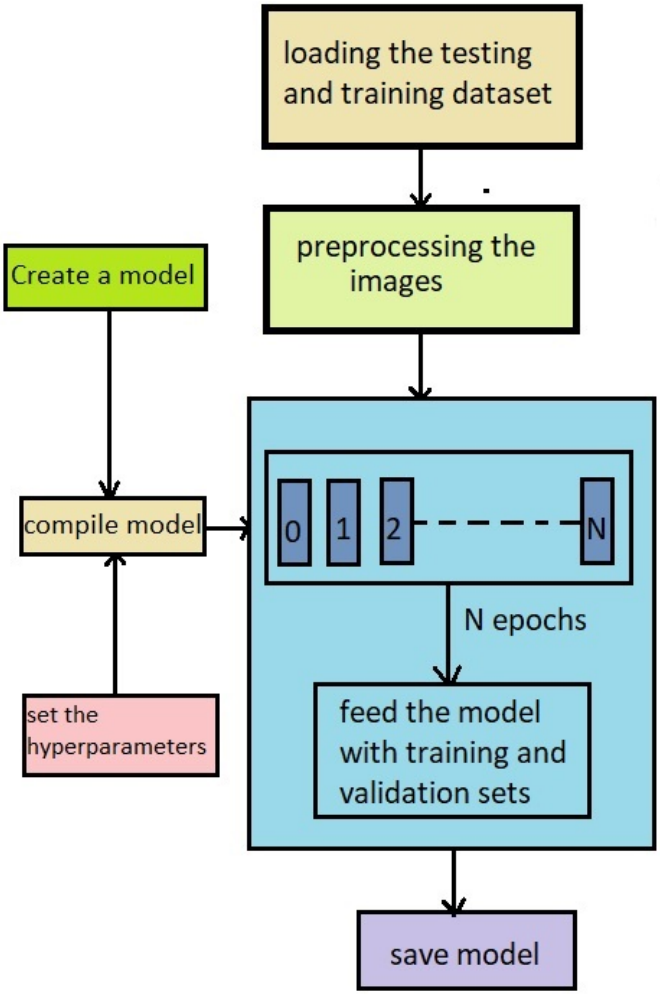

Figure 3.2: Flowchart of entire testing and training process

The architecture of proposed CNN model consists of 10 layers - 4 convolution layer, 4 max pooling layers which are after every convolution layer, 2 fully connected layers. We use batch normalization after every convolution layer. Using **batch normalization** makes the network steady in the course of

training. This requires the use of a larger learning rate, that in turn can speed up the learning process of the model.

An activation function (ReLu) is used in each convolutional layer to introduce non-linearity in the data-set. We use ReLu activation function in each of the four convolution layers. The input layer is composed of sample images of dimension 32×32 We perform another experiment with the same images but of dimension 64×64 and analyze the results. The output layer is composed of a *softmax* classifier with 156 labels.

A filter is a set of learnable weights which are learned using the back-propagation algorithm. We can think of each filter as storing a single template/pattern. The first two convolution layer have filter set to 32 and next two have filter set to 64. In all the max pooling layers, pool size and strides are both set to 2 as discussed in the section 2.2.3. Architecture of the proposed work is explained in the Figure 3.3.

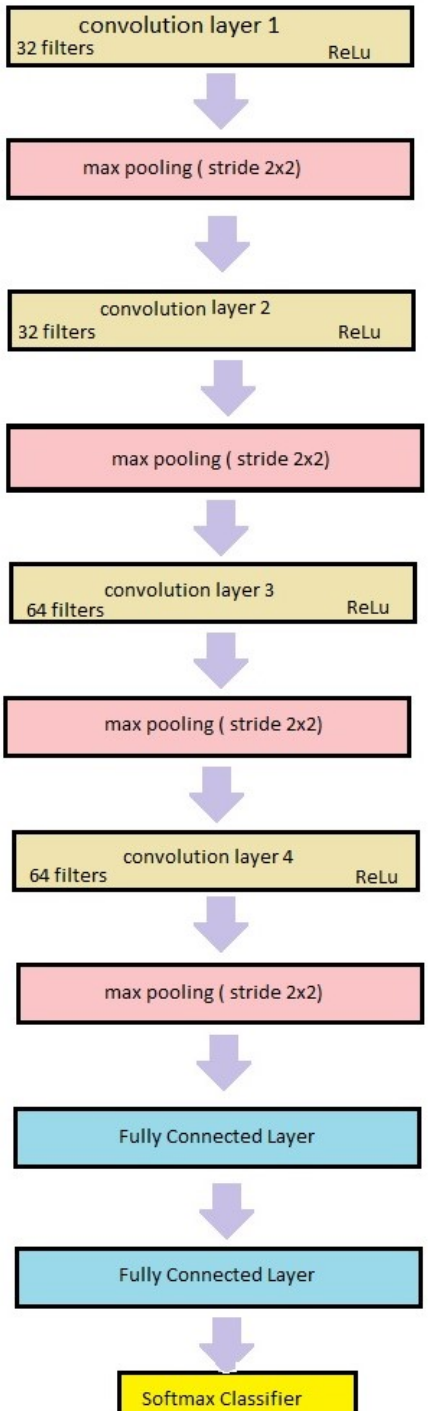

Figure 3.3: Architecture of proposed model

3.1.3 Hyper-parameters

Several hyper-parameters are required to run the model. Some of them are specific to the architecture and some are needed to increase the accuracy of the results of the network.

Various hyper-parameters are present in the layers of CNN which can be fine tuned for better results. We have set the stride value to 1. "Zero" or "same" padding is done to maintain the image size across all the outputs of every layer so that no pixels are lost around the borders of the images.

We have used Xavier initialization for initializing weights in the proposed model. The number of epochs is experimental. We tried various ranges from 15 -100. We aim to discuss the results obtained in the coming sections. Input images are of size 64x64 fed into batches of size 64. To avoid over training of the model, we use early stopping method . Adam Optimizer is used to optimize the gradient descent learning algorithm. Adam Optimizer uses **cross-entropy** to optimize the loss function. Different batch sizes can be experimented. Table 3.1 tabulates all the hyper-parameters set.

Table 3.1: Hyper-parameter set.

Hyper-parameters	Values
Batch Size	64
Initialization	Xavier
Optimizer	Adam
Epochs	100
Activation function	ReLu
Learning rate	0.001

3.2 Experiments

We use *ImageDataGenerator* function of keras to import the input images to the environment.

Firstly, we use the batch size of 32 and the sample images with half of their size ,i.e., 32×32. Various parameters have to be set up. In the training set,we have a total of 63,024 images belonging to 156 classes. In the validation set, we have a total of 11,544 images belonging to 156 classes. In the model, we set no. of epochs to be 100. *Steps per epoch* has to be less than or equal to total training samples divided by batch size. In our case, we take *steps per epoch* to be 1970. Similarily, *validation steps* has to be less than or equal to total testing samples divided by batch size. We take it to be 361. The results are given in Table 3.2

Table 3.2: Result of 1^{st} experiment

No of epochs	100
Batch Size	32
Total parameters	93,244
Trainable Parameters	92,476
Non-trainable Parameters	768
Training Loss	0.3174
Training Accuracy	0.8934
Testing Loss	1.1097
Testing Accuracy	0.8546

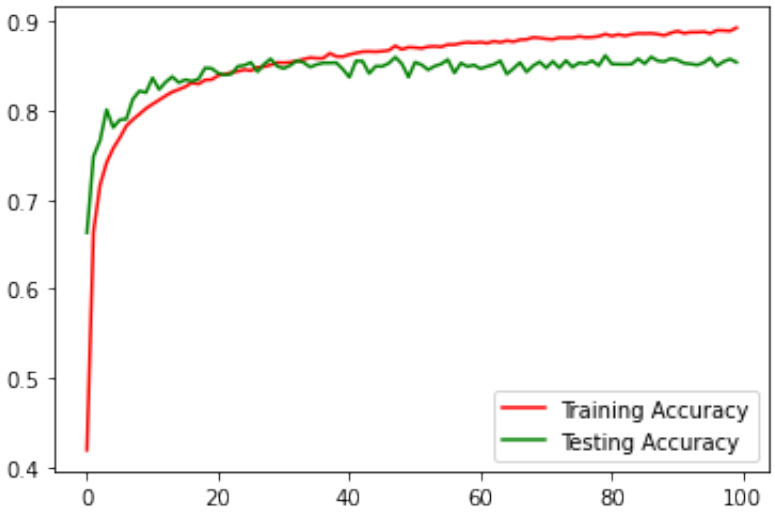

Figure 3.4: Training Accuracy vs Testing Accuracy

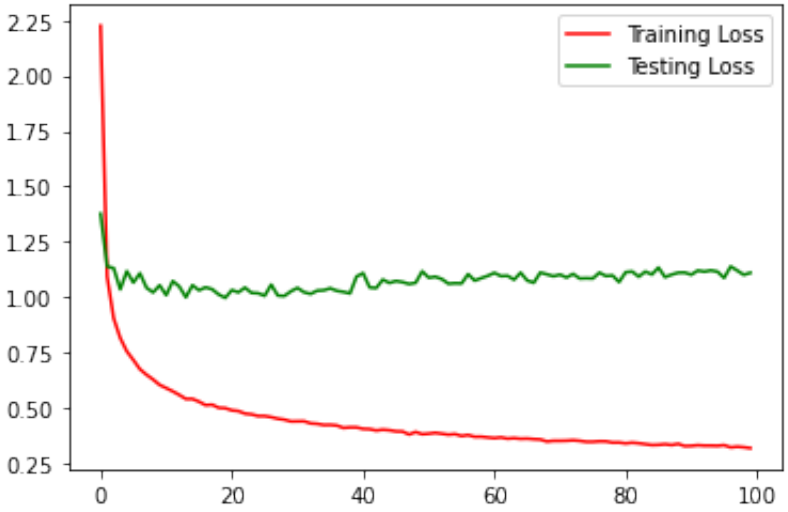

Figure 3.5: Training Loss vs Testing Loss per epoch

We plot the first filter of first layer. The first of the filters are basic ones detecting edges. As we go to high level filters, they capture high level intricacies such as objects. We have total of 32 filters in the first convolution layer and we can visualise all of them. Each filter does extraction and processing of different features in the sample image.

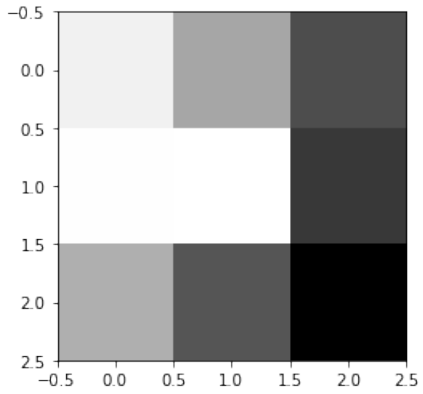

Figure 3.6: 1^{st} filter of Layer 1

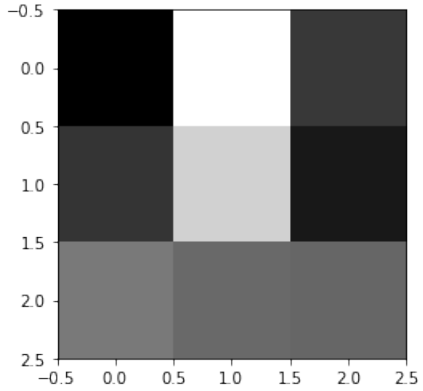

Figure 3.7: 32^{th} filter of Layer 1

We have a vast data-set of Devnagri available.The proposed model was executed on this data-set without making any changes to any in the hyper-parameters .The labels were changed to 56 as there are only 56 characters in Devnagri Font. For each character, we have 2000 sample images which is then split into 1700 images for training set and 300 for validation set. Number of epochs is set to 100 here too. Steps per epoch is 2444 and no of validation steps is set to 432. The results are tabulated in Table 3.3.

Table 3.3: Result of Devnagri

No of epochs	100
Batch Size	32
Total parameters	86,094
Trainable Parameters	85,326
Non-trainable Parameters	768
Training Loss	0.0724
Training Accuracy	0.9774
Testing Loss	0.0520
Testing Accuracy	0.9847

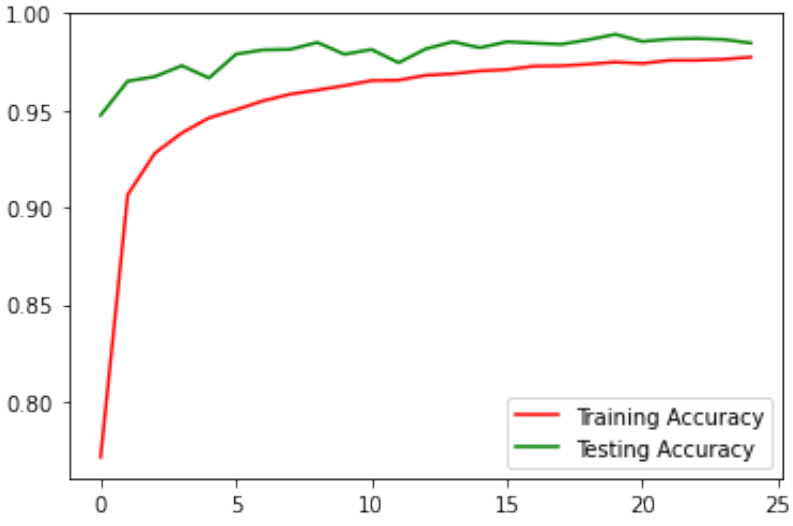

Figure 3.8: Training Accuracy vs Testing Accuracy

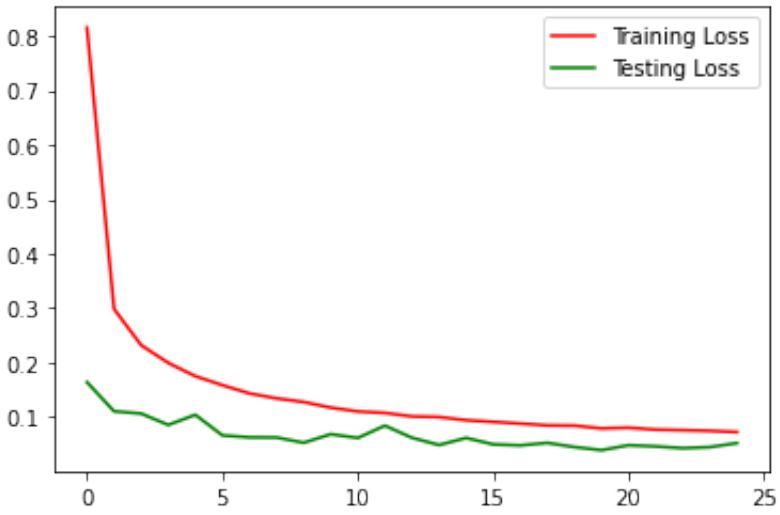

Figure 3.9: Training Loss vs Testing Loss per epoch

Next, we use the sample images of Tamil data-set of size 64×64. We also double up the batch size to 64. All other hyper-parameters are same except the number of epochs which is reduced to 50, because of computational constraints. The results are tabulated in Table 3.4.

Table 3.4: Result of 2^{nd} experiment

No of epochs	50
Batch Size	64
Total parameters	158,780
Trainable Parameters	158,012
Non-trainable Parameters	768
Training Loss	0.2569
Training Accuracy	0.9285
Testing Loss	1.0871
Testing Accuracy	0.7421

Fig.3.10 and Fig.3.11 gives us the accuracy and loss graphs.

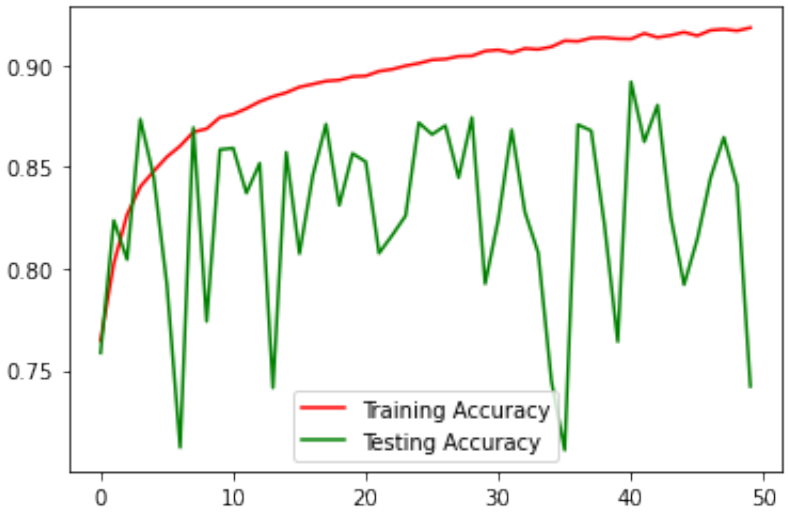

Figure 3.10: Training Accuracy vs Testing Accuracy

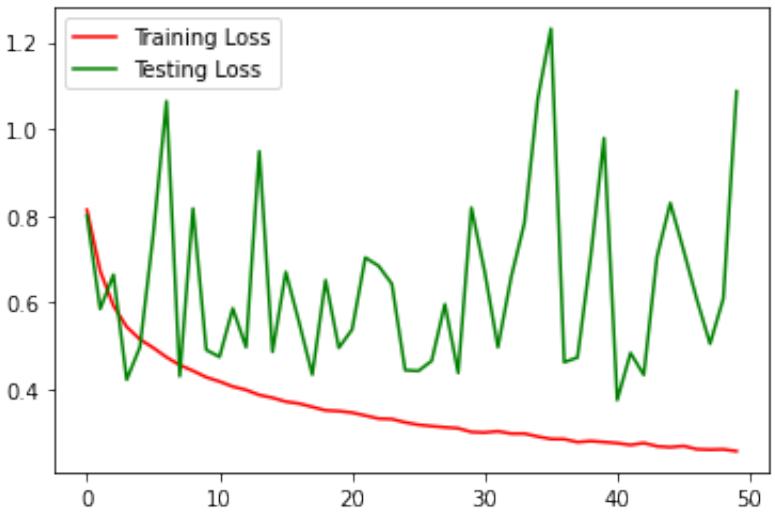

Figure 3.11: Training Loss vs Testing Loss per epoch

From the figures 3.10 and 3.11, we see the graphs of Testing Accuracy and Testing Loss has many crests and troughs. This can possibly mean our chosen learning rate is too high in the model.

Chapter 4

Conclusion and Future scope

CNN performs significantly better when compared with Feature Extraction Models. Hence, in this study, we apply CNN model to Tamil font character classification. A handwritten data-set of 156 Tamil characters comprising of more than 480 images for each classification. The data-set is obtained from publicly available HPLabs database on their website.

The model is ran for Tamil character set. The data-set after pre-processing is split into train and test sets. For each character(also called labels), 404 images are in training data-set and 74 in validation data-set. Batch sizes of 32 and 64 were ran, and their accuracies were compared. Learning rate is chosen to be 0.0001. With the batch size 32, training accuracy obtained is 89.34%, and testing accuracy is 85.46% . When the batch size is 64, training accuracy comes out to be 92.85% , and testing accuracy drops down drastically to 74.21%. We also ran the proposed model on Devnagri font, with batch size of 32, the training and testing accuracies are 97.74% and 98.47% respectively. The difference in the accuracy is because of Devnagri being a significantly less complex script with lesser number of characters. Also, in the case of devnagri, per character 2000 image samples are there compared to

480 samples per character of Tamil. We also plot various comparison graphs which help us visualise the differences.

In the future, we aim to increase the accuracy of Tamil Character Recognition System by -

1. Increase the number of samples available per character.
2. Better pre-processing of the sample images.
3. Develop automatic feature extraction methods and combine them with the proposed model.

We didn't consider all characters of Tamil font because of lack of data-set. We plan to apply the proposed model to all the characters of Tamil font.

Bibliography

[1] U. Bhattacharya, S. K. Ghosh, and S. Parui. A two stage recognition scheme for handwritten tamil characters. In *Ninth International Conference on Document Analysis and Recognition (ICDAR 2007)*, volume 1, pages 511–515, 2007.

[2] Chaouki Boufenar, Adlen Kerboua, and Mohamed Batouche. Investigation on deep learning for off-line handwritten arabic character recognition. *Cognitive Systems Research*, 50:180–195, 2018.

[3] Dan Ciregan, Ueli Meier, and Jürgen Schmidhuber. Multi-column deep neural networks for image classification. In *2012 IEEE conference on computer vision and pattern recognition*, pages 3642–3649. IEEE, 2012.

[4] Dan Cireşan and Ueli Meier. Multi-column deep neural networks for offline handwritten chinese character classification. In *2015 international joint conference on neural networks (IJCNN)*, pages 1–6. IEEE, 2015.

[5] Line Eikvil. Optical character recognition. *citeseer. ist. psu. edu/142042. html*, 1993.

[6] Ahmed El-Sawy, Mohamed Loey, and Hazem El-Bakry. Arabic handwritten characters recognition using convolutional neural network. *WSEAS Transactions on Computer Research*, 5:11–19, 2017.

[7] Mohamed Elleuch, Najiba Tagougui, and Monji Kherallah. Optimization of dbn using regularization methods applied for recognizing arabic handwritten script. *Procedia Computer Science*, 108:2292–2297, 2017.

[8] Kevin Gurney. *An introduction to neural networks*. CRC press, 1997.

[9] Sakshi Indolia, Anil Kumar Goswami, SP Mishra, and Pooja Asopa. Conceptual understanding of convolutional neural network-a deep learning approach. *Procedia computer science*, 132:679–688, 2018.

[10] Tiji M Jose and Amitabh Wahi. Recognition of tamil handwritten characters using daubechies wavelet transforms and feed-forward backpropagation network. *International Journal of Computer Applications*, 64(8), 2013.

[11] BR Kavitha and C Srimathi. Benchmarking on offline handwritten tamil character recognition using convolutional neural networks. *Journal of King Saud University-Computer and Information Sciences*, 2019.

[12] O'Shea Keiron Teilo and Ryan Nash. An introduction to convolutional neural networks. *Research Gate', 15th December*, 2015.

[13] Cheng-Lin Liu, Fei Yin, Da-Han Wang, and Qiu-Feng Wang. Casia online and offline chinese handwriting databases. In *2011 International Conference on Document Analysis and Recognition*, pages 37–41. IEEE, 2011.

[14] N Shanthi and K Duraiswamy. A novel svm-based handwritten tamil character recognition system. volume 13, pages 173–180. Springer, 2010.

[15] C Sureshkumar and T Ravichandran. Handwritten tamil character recognition and conversion using neural network. *Int J Comput Sci Eng*, 2(7):2261–67, 2010.

[16] Charlie Tsai. Recognizing handwritten japanese characters using deep convolutional neural networks. *University of Stanford in Stanford, California*, pages 405–410, 2016.

[17] P. Vijayaraghavan. Tamil recognition using a convolutional neural network. 2015.

[18] Xu-Yao Zhang, Yoshua Bengio, and Cheng-Lin Liu. Online and offline handwritten chinese character recognition: A comprehensive study and new benchmark. *Pattern Recognition*, 61:348–360, 2017.