HAND-WRITTEN BANGLA CHARACTER RECOGNITION USING DEEP CONVOLUTIONAL NEURAL NETWORK

by

Dipayan Bhadra

Master of Science in Electrical and Electronic Engineering

Department ofElectrical and Electronic Engineering

Bangladesh University of Engineering and Technology

January 2019

Candidate's Declaration of Authorship

It is hereby declared that this thesis titled, '**Hand-Written Bangla Character Recognition Using Deep Convolutional Neural Network**' or any part of it has not been submitted elsewhere for the award of any degree or diploma and that all sources are acknowledged.

Signature: Dipayan

Date: 26.01.2019

Board of Examiners

The thesis titled 'Hand-Written Bangla Character Recognition Using Deep Convolutional Neural Network' submitted by Dipayan Bhadra, Student-ID: P0411062217, Session: April 2011 has been accepted as satisfactory in partial fulfillment of the requirements for the degree of Master of Science in Electrical and Electronic Engineering on January 26, 2019.

ACKNOWLEDGEMENTS

I would like to convey my sincerest appreciations and heartfelt gratitude to my thesis supervisor, Professor Dr. S.M. Mahbubur Rahman, for his all-out support, help, inspiration and able guidance throughout the tenure of his supervision. His plentiful resources and pragmatic ideas in the field of computer vision have allowed me to work with exhaustive datasets as well asstate-of-the-art theories and techniques in this field. He has always been a great mentor for a novice like me, and his appreciation and constructive criticism in every stage of this thesis have only bolstered and instilled the confidence in me to successfully complete the dissertation. I am truly owed to have the experience of doing research work under his supervision.

I am grateful to Professor Dr. Md. Shafiqul Islam, Head of the Department of Electrical and Electronic Engineering, BUET for rendering me support and departmental assistance during my M.Sc. study in BUET.

I would also like to thank all my teachers, colleagues and friends for their support and encouragement during the work. I am indebted to my parents for their mental supportand encouragement.

And finally, I would like to thank the Almighty, the Lord of the Worlds, for blessing me with so many respected personalities around me and this auspicious achievement, in particular.

Dipayan Bhadra
January, 2019

Contents

Contents

Contents

Abstract

In recent years, there has been much interest in automatic character recognition. Between handwritten and printed forms, Handwritten Character Recognition (HCR) is more challenging. A handwritten character written by different persons is not identical but varies in both size and shape. Numerous variations in writing styles of individual character make the recognition task difficult. The similarities in distinct character shapes, the overlaps, and the inter-connections of the neighboring characters further complicate the problem. Recently, the Convolutional Neural Network (CNN) has been shown noticable success in the area of image-based recognition, video analytics, and natural language processing due to their unique characteristics of feature extraction and classification. This is mainly due to the fact that the design of a CNN is motivated by the close imitation of visual mechanism as compared to the conventional neural network. The convolution layer in a CNN performs the similar filtering function that is seen in the cells of visual cortex. As a result of replication of weight configuration of one layer to the local neighboring receptive field in the previous layer through the convolution operation, the features extracted by the CNN possess the invariance properties of scale, rotation, translation and other distortions of a pattern. A recently reported HCR technique that considers the Bangla characters uses shallow CNN by considering only two-level convolution layers and a fixed kernel size experimented on a small-size private dataset. In this thesis, a Deep CNN with three convolutional layers with different kernel sizes in different convolutional layers is used on a large dataset made of combining two datasets. Experimental result shows an accuracy in recognition that is 7% higher than that of previous work.

List of Figures

List of Tables

List of Abbreviations

BBCD	Bangla Basic Character Database
BHCR	Bangla Hand-written Character Recognition
CDR	Correct Detection Rate
CMATER	Center for Microprocessor Applications for Training Education and Research
CNN	Convolutional Neural Network
DCNN	Deep Convolutional Neural Network
DNN	Deep Neural Network
GPU	Graphic Processing Unit
HCR	Hand-written Character Recognition
HMM	Hidden Markov Model
ICA	Independent Component Analysis
LDA	Linear Discriminant Analysis
LGN	Lateral Geniculate Nucleus
MICR	Magnetic Ink Character Recognition
MLP	Multilayer Perceptron
MQDF	Modified Quadratic Discriminant Function
NAG	Nesterov's Accelerated Gradient
NN	Neural Network
OCR	Optical Character Recognition
PCA	Principle Component Analysis
ReLU	Rectified Linear Unit
ResNet	Residual Network
ROI	Region of Interest
SIFT	Scale Invariant Feature Extraction
SVM	Support Vector Machine

Chapter 1

Introduction

1.1. Introduction

Optical character recognition (OCR) is the mechanical or electronic conversion of images of typed, handwritten or printed text into machine-encoded text, whether from a scanned document, a photo of a document, a scene-photo (for example the text on signs and billboards in a landscape photo) or from subtitle text superimposed on an image (for example from a television broadcast). It is widely used as a form of information entry from printed paper data records, whether passport documents, invoices, bank statements, computerized receipts, business cards, mail, printouts of static-data, or any suitable documentation. It is a common method of digitizing printed texts so that they can be electronically edited, searched, stored more compactly, displayed on-line, and used in machine processes such as cognitive computing, machine translation, (extracted) text-to-speech, key data and text mining. OCR is a field of research in pattern recognition, artificial intelligence and computer vision.

Character Recognition techniques associate a symbolic identity with the image of a character. Character recognition system is classified into two, based on data acquisition and text type: online and offline (Figure. 1.1). The online character recognition system utilizes the digitizer which directly capture writing with the order of the strokes, speed, pen up and pen down information. Offline character recognition captures the data from paper through optical scanner or cameras. Offline character recognition is also known as optical character recognition because the image of text is converted in to a bit pattern by optically digitizing devices. In case of online handwritten character recognition, the handwriting is captured and stored in digital form via different means. Usually, a special pen is used in conjunction with an electronic surface. As the pen moves across the surface, the two- dimensional coordinates of successive points are represented as a function of time

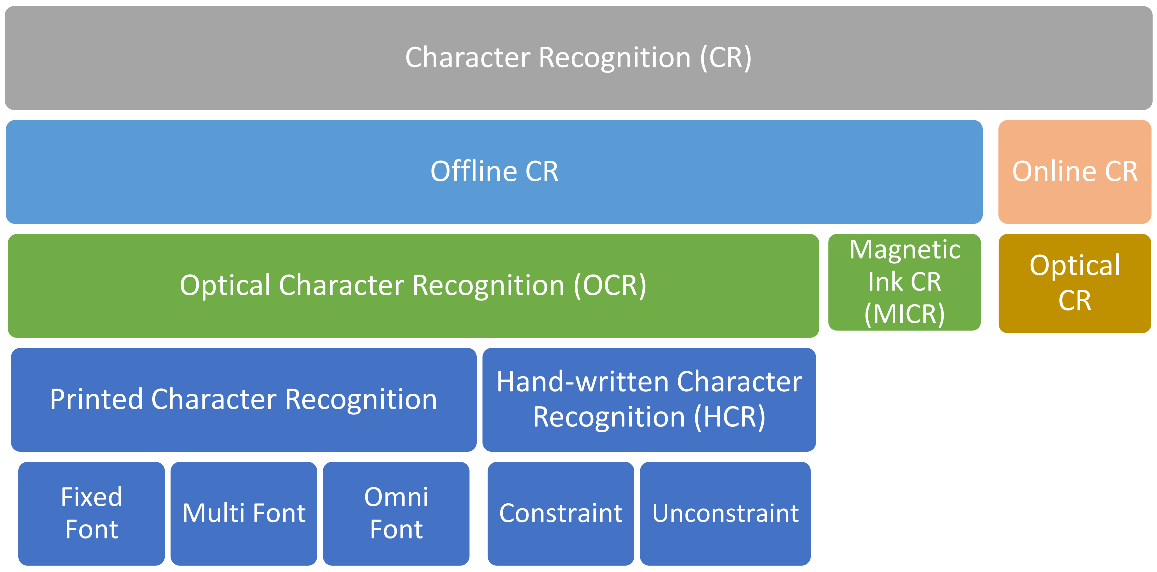

Character Recognition (CR)
Offline CR
Online CR
Optical Character Recognition (OCR)
Magnetic Ink CR (MICR)
Optical CR
Printed Character Recognition
Hand-written Character Recognition (HCR)
Fixed Font
Multi Font
Omni Font
Constraint
Unconstraint

Figure 1.1 Categories of character recognition system

and are stored in order. It is generally accepted that the on-line method of recognizing handwritten text has achieved better results than its offline counterpart. This may be attributed to the fact that more information may be captured in the on-line case such as the direction, speed and the order of strokes of the handwriting.

The offline character recognition can be further grouped into two types:

- Magnetic Ink Character Recognition (MICR)
- Optical Character Recognition (OCR)

In MICR, the characters are printed with magnetic ink. The reading device can recognize the character according to the unique magnetic field of each character. MICR is mostly used in banks for check authentication. OCR deals with the recognition of characters acquired by optical means, typically a scanner or a camera. The characters are in the form of digital images and can be either printed or handwritten, of any size, shape or orientation. The OCR can be subdivided into handwritten character recognition and printed character recognition. Handwritten character recognition is more difficult to implement than printed character recognition due to diverse human handwriting styles and customs. In printed character recognition, the images to be processed are in the forms of standard fonts like Times New Roman, Arial and Courier etc.

1.2. Handwritten Character Recognition

1.2.1. Application of Offline Handwritten Character Recognition

HCR has been successfully used in several applications. Some of the important applications of offline handwritten recognition are discussed in the following section:

- **Bank Automaion**: Offline handwritten recognition is basically used for cheque reading in banks. Cheque reading is the very important commercial application of offline handwritten recognition. Handwritten recognition system plays very important role in banks for signature verification and for recognition of amount filled by user.
- **Postal office automation:** Handwritten recognition system can be used for reading the handwritten postal address on letters. Offline handwritten recognition system used for recognition handwritten digits of postcode. HCR can be read this code and can sort mail automatically.
- **Form Processing:** HCR can be also used for form processing. Forms are normally used for collecting the public information. Replies of public information can be handwritten in the space provided.
- **Signature Verification:** HCR can also be used to identify the person by signature verification. Signature identification is the specific field of handwritten identification in which the writer is verified by some specific handwritten text. Handwritten recognition system can be used for identify the person by handwriting, because handwriting may be vary from person to person.

1.2.2. Background of HCR Systems

HCR system is developed with an objective to recognize handwritten characters from a digital image of handwritten documents. An HCR system includes steps such as image acquisition, character segmentation, pre-processing of character image, feature extraction and recognition of character class with the extracted features as well as post processing.

a) Image acquisition
   Gray-level scanning of handwritten paper documents, at an appropriate resolution typically 300-1000 dpi.

b) Preprocessing
   – Binarization (two-level thresholding).
   – Segmentation to isolate individual character.
   – Conversion to another character representation like skeleton or contour.

c) Feature Extraction
   – Extracting meaningful features.

d) Classification
   – Recognition using one or more classifier.

e) Contextual verification on post processing

Block diagram of a general character recognition system is shown in Figure 1.2. Images for HCR system might be acquired by scanning hand-written document or by capturing photograph of document or by directly writing in computer using stylus. This is also known as digitization process. Preprocessing involves series of operations performed to enhance to make it suitable for segmentation. Preprocessing step involves noise removal generated during document generation. Proper filter like mean filter, min-max filter and Gaussian filter may be applied to remove noise from document. Binarization process converts gray scale or colored image to black and white image. Binary morphological operations like opening, closing, thinning, hole filling etc may be applied to enhance image.

If document is scanned then it may not be perfectly horizontally aligned, so we need to align it by performing slant angle correction. Input document may be resized if it is too large in size to reduce dimensions to improve speed of processing. However reducing dimension below certain level may remove some useful features too. Generally document is processed in hierarchical way. At first level lines are segmented using row histogram. From each row, words are extracted using column histogram and finally characters are extracted from words. Accuracy of final result is highly depends on accuracy of segmentation.

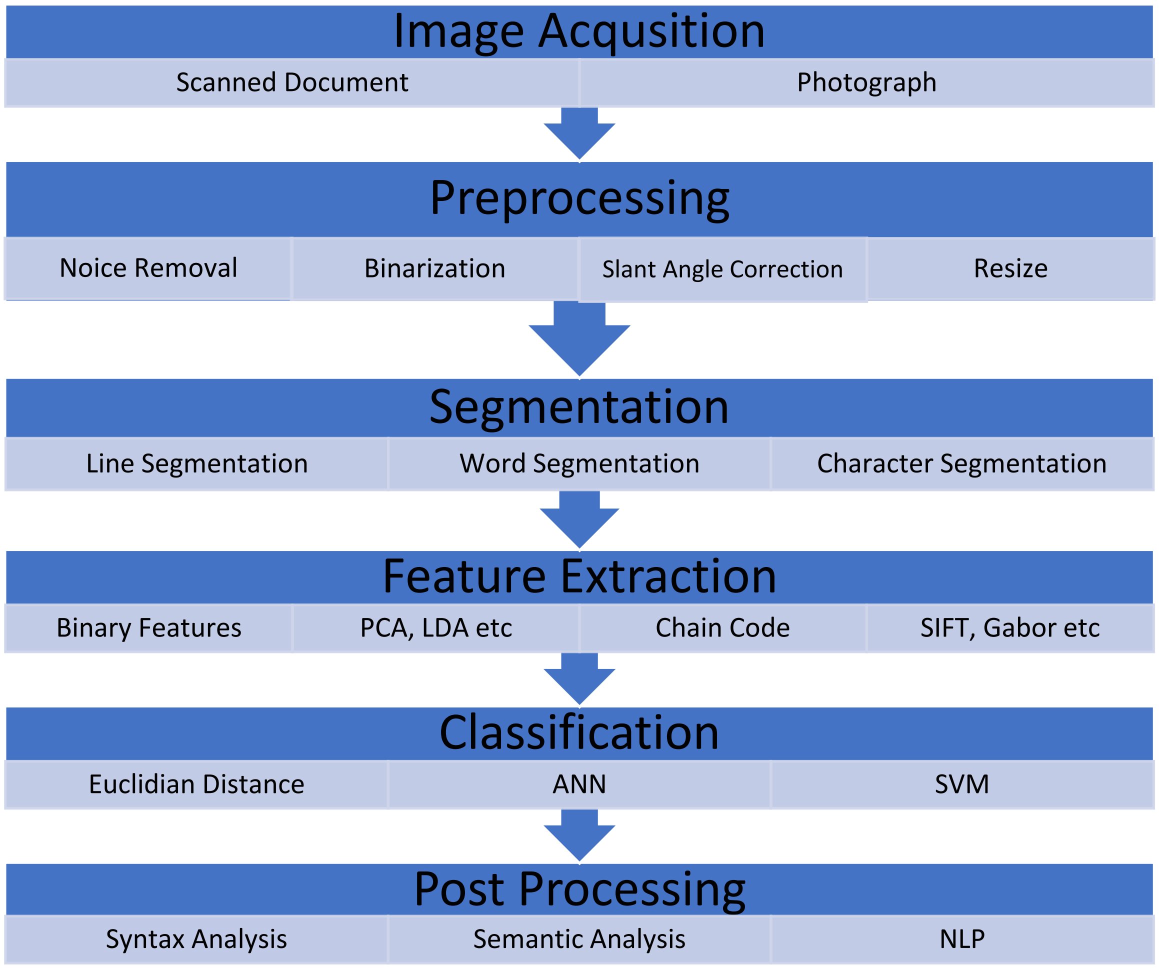


Figure 1.2: Different steps in character recognition system

Feature extraction is the heart of any character recognition system. Feature extraction techniques like Principle Component Analysis (PCA), Linear Discriminant Analysis (LDA), Independent Component Analysis (ICA), Chain Code, Scale Invariant Feature Extraction (SIFT), zoning, gradient based features and histogram are applied to extract the features of individual characters. These features are used to train classification system. When a new input image is presented to HCR system, its features are extracted and given as an input to the trained classifier like artificial neural network or support vector machine. Classifiers compare the input feature with stored pattern and find out the best matching class for input. A post processing, though not mandatory, improve the accuracy of recognition. Syntax and semantic analysis or similar higher level concepts might be applied to check the context of recognized character.

1.2.3. Challenges

Since this task of recognizing character from an image is relatively trivial for a human to perform, it is worth considering the challenges involved from the perspective of a Computer Vision algorithm. An in-exhaustive list of general challenges in image recognition task is given below:

- Viewpoint variation: A single instance of an object can be oriented in many ways with respect to the camera.

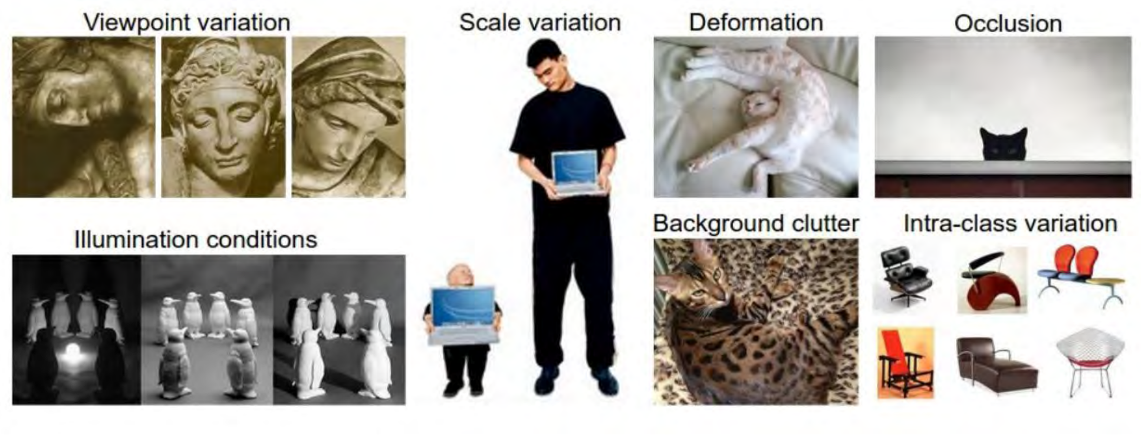


Figure 1.3: General challenges in image recognition problems

- Scale variation**:** Visual classes often exhibit variation in their size (size in the real world, not only in terms of their extent in the image).
- Deformation: Many objects of interest are not rigid bodies and can be deformed in extreme ways.
- Occlusion: The objects of interest can be occluded. Sometimes only a small portion of an object (as little as few pixels) could be visible.
- Illumination conditions: The effects of illumination are drastic on the pixel level.
- Background clutter: The objects of interest may blend into their environment, making them hard to identify.
- Intra-class variation: The classes of interest can often be relatively broad, such as chair. There are many different types of these objects, each with their own appearance.

Moreover, hand-written character recognition is more challenging than character recognition from printed form. In this particular case of hand-written character recognition task from images the complexity/challenges of recognition task extends because of numerous variations in writing styles, character shapes & sizes of different persons and similarities in character shapes, the overlaps, and interconnections of neighboring characters. HCR complexity varies among different languages due to distinct shapes, strokes and number of characters.

There are more characters in Bangla (50 characters) than in English (26 characters) and some contains additional sign up and/or below. Also compound characters are also used in Bangla frequently. Moreover, Bangla contains many similar shaped characters; in some cases a character differ from its similar one with a single dot or mark. These characteristics make difficult to achieve better performance with simple technique as well as hinders to work with Bangla HCR than English HCR.

1.3. Problem Identification

Bangla character set is divided into two categories: basic and compound characters. Basic characters are the collection of vowels and consonants. Bangla character set has 11 vowels and 39 consonants (Table 3.1). In Bangla, there are a large number of compound characters formed by combination of two or more basic characters. Most basic character shapes have a horizontal line at their upper parts, called headline or matra and three zones of such characters can be identified as shown in Fig. 3.1. Each of these characters (excepting the character 'BINDU') has a part in the middle zone while only a few of them have an additional part either in the upper or in the lower zones.

Table 1.1: Basic Bangla Characters. There are 11 vowels and 39 consonants in Bangla script.

Vowels (11 nos.)										
অ	আ	ই	ঈ	উ	ঊ	ঋ	এ	ঐ	ও	ঔ
A	AA	I	II	U	UU	R	E	AI	O	AU
Consonants (39 nos.)										
ক	খ	গ	ঘ	ঙ	চ	ছ	জ	ঝ	ঞ	ট
KA	KHA	GA	GHA	NGA	CA	CHA	JA	JHA	NYA	TTA
ঠ	ড	ঢ	ণ	ত	থ	দ	ধ	ন	প	ফ
TTHA	DDA	DDHA	NNA	TA	THA	DA	DHA	NA	PA	PHA
ব	ভ	ম	য	র	ল	শ	ষ	স	হ	ড়
BA	BHA	MA	YY	RA	LA	SHA	SSA	SA	HA	RRA
ঢ়	য়	ৎ	s	t	u					
DHRA	YYA	KHAND	ANUS	VISARG	BINDU					

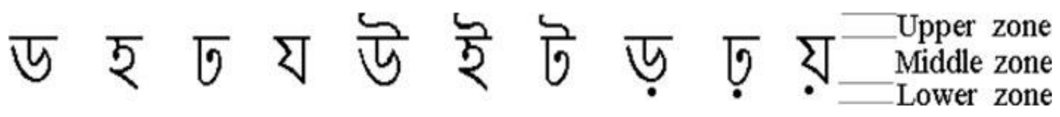


Figure 1.4: Different zones of Bangla Characters

1.4. Related Works

Offline handwriting recognition has been studied extensively during the last three decades [1–6]. Among these, recognition of isolated characters has the advantage that segmentation is usually not needed and when written in boxes, size normalization is accomplished to a large extent. So, experimental results on them provide a kind of upper bound of performance of the character recognizer in a handwriting analysis task.

Numerous techniques have been proposed in the literature for recognition of isolated handwritten characters. These include (a) template matching [7, 8], e.g., direct pixel matching, deformable template matching, relaxation based matching, structural shape matching, etc.; (b) statistical classifier, e.g., Bayes' classifier [9], hidden Markov model (HMM) [10, 11], etc.; (c) graph-based and automata-based syntactic classifier; (d) machine learning-based techniques involving neural net [12, 13], rough set [14], fuzzy set [15, 16], support vector machine (SVM) [17, 18], etc. Among these, the approaches based on HMM and SVM are popular due to their potential in recognition of unconstrained handwriting. Convolutional Neural Network [19] is also very efficient in document recognition tasks. In the overall recognition scheme, preprocessing techniques such as size normalization, smoothing, slant correction, etc., efficient feature selection and suitable post-processing methods that make use of contextual information for error correction play important roles to improve the final performance.

Most of the reported studies on handwriting recognition have been done on English [18, 20, 21] and oriental scripts like Chinese [22, 23], Korean [10, 24] and Japanese [25, 26]. The reports on Indian scripts are a few only. In the earliest such study [27], stroke-based features and a tree classifier were used for classification of handwritten Devanagari numerals. Parui et al. [28] proposed a syntactic scheme for handwritten Bangla numeral recognition while Dutta and Chaudhuri [29] used a neural net classifier to recognize isolated handwritten alphanumeric characters. Among others, Bhattacharya et al. [30] used self-organizing neural net while Bhattacharya and Chaudhuri [31] used classifier combination approach to recognition of handwritten Bangla numerals. A multistage recognition scheme for mixed numerals is reported recently [32]. For Bangla alphabetic characters, Rahman et al. [33] proposed a multistage scheme while Bhowmick et al. [34] used a neural network-based approach. HMM-based recognition of Bangla basic characters is reported in [35]. A major obstacle to effective research on off-line handwritten character recognition of Bangla and other Indian scripts is the non-existence of required benchmark databases. Previous studies were reported on the basis of small databases collected in laboratory environments. However, several standard databases such as NIST, MNIST [19], CEDAR [36], CENPARMI, etc., are available for Latin script. Khosravi and Kabir [37] presented a large dataset of handwritten Farsi digits. An Arabic handwritten database consisting of words and texts written by 100 writers was described in [38]. Su et al. [39] presented a Chinese handwriting database HIT-MW collected in an unconstrained manner. A few other databases of handwriting samples include [40, 41] and [42].

A few notable works are available for Bengali handwritten character recognition. Bhowmik et al. [52] proposed a fusion classifier using Multilayer Perceptron (MLP), RBF network and SVM. They used wavelet transform for feature extraction from character images. In classification, they considered some similar characters as a single pattern and trained the classifier for 45 classes. Basu et al. [53] proposed a hierarchical approach to segment characters from words and MLP is used for classification. In segmentation stage they used three different feature extraction techniques but they reduced character patterns into 36 classes merging similar characters in a single class. Recently, Battacharya et al. [54] considered a two-stage recognition scheme for 50 basic character classes. Feature vector for the first classifier is computed by overlaying a rectangular grid consisting of regularly spaced horizontal and vertical lines over the character bounding box. The response of this first classifier is analyzed to identify its confusion between a pair of similar shaped characters. Second stage of classification is used to resolve the confusion and feature vector is computed by overlaying another rectangular grid but consisting of irregularly spaced horizontal and vertical lines over the character bounding box. They used Modified Quadratic Discriminant Function (MQDF) classifier and MLP as classifiers in first and second stages, respectively.

Recently, Md. Mahbubar Rahman et al. [55] applied CNN scheme to Bengali HCR and reported 85.96% test accuracy. CNN with two convolution and sub-sample layers are used in this work. Kernel size considered in this work is 5×5. 6 and 12 kernels were used in 1st and 2nd convolution layer respectively to extract features. A database was created by taking samples from 30 individuals of different ages and education levels. prepared dataset size was 20000 having 400 samples for each character among which 17500 samples (350 samples for each character) were used as training set and 2500 samples (50 samples per character) were used as test set.

1.5. Motivation and Scope of Works

Convolutional neural network (CNN) has ability to recognize visual patterns directly from pixel images with minimal preprocessing. Deep CNN (DCNN) [5] has been being used successfully for image classifications, handwritten digit and character recognition in recent years. But there is no record of DCNN being used for Bangla HCR (BHCR) task. For this reason, DCNN scheme will be investigated in BHCR task and performance will be analyzed. It can be assumed easily that DCNN based BHCR will give satisfactory results in terms of recognition accuracy, time requirement for recognition and storage requirements since after training, the training data will not be needed to be stored. Only the weights and biases of the network are stored which requires very negligible storage size. Training requires much time, but testing requires very small amount of time, so it can be applied in real-time recognition and analysis.

1.6. Objectives

The specific objectives of this thesis are:

- To develop an architecture of Deep CNN (DCNN) to recognize hand-written Bangla characters
- To analyze the DCNN architecture and determine optimum number of convolutional layers and kernel-size that would provide improved recognition accuracy of fifty classes of hand-written Bangla characters.
- To evaluate the performance of the proposed DCNN based recognition scheme with that of existing methods in terms of accuracy, storage requirement, and computational complexity on publicly available dataset

The outcome of the thesis is a novel recognition scheme for hand-written Bangla characters with low-level storage requirement and processing time that would provide improved accuracy to facilitate automatic recognition.

1.7. Outline

The thesis is organized as follows:

In Chapter 2, a brief review of neural network and convolutional neural network is introduced. Then the advantages of CNN over NN are explained.

In Chapter 3, proposed DCNN architecture is explained.

Chapter 4 describes the database used in the experiment, experimental results and analyses by comparing the proposed method with the existing recognition methods.

Finally, Chapter 5 provides the conclusion along with the scopes for future work.

Chapter 2

Convolutional Neural Network: A Review

2.1. Introduction

This chapter provides a review on convolutional neural network. Since CNN is a category to neural network, hence at first a brief introduction of NN along with the structure and training method are explained. After that the basic structure of a CNN is presented. But as the training method of CNN is similar to the NN, so it is omitted. At the end of this chapter, the advantages of CNN over NN are presented.

2.2. Neural Networks

A neural network is a system of interconnected artificial "neurons" that exchange messages between each other. The connections have numeric weights that are tuned during the training process, so that a properly trained network will respond correctly when presented with an image or pattern to recognize. The network consists of multiple layers of feature-detecting "neurons". Each layer has many neurons that respond to different combinations of inputs from the previous layers. As shown in Figure 2.1, the layers are built up so that the first layer detects a set of primitive patterns in the input, the second layer detects patterns of patterns, the third layer detects patterns of those patterns, and so on. Deep neural networks typically use 2 to 10 distinct layers for pattern recognition.

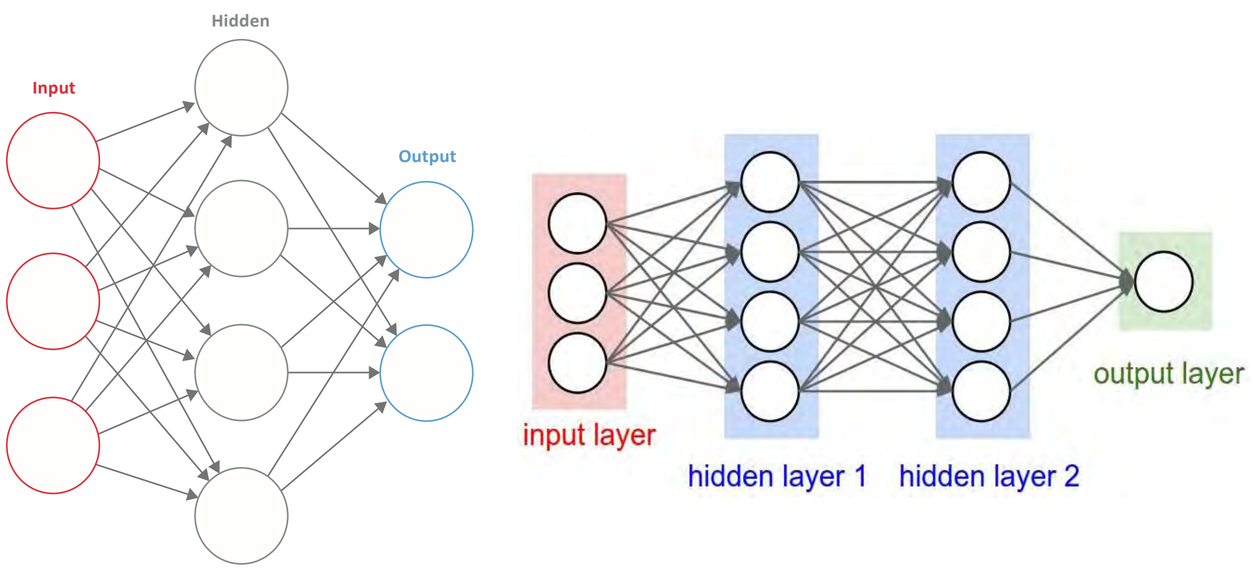


Figure 2.1: An artificial neural network

Training of a NN is performed using a “labeled” dataset of inputs in a wide assortment of representative input patterns that are tagged with their intended output response. Training uses general-purpose methods to iteratively determine the weights for intermediate and final feature neurons. Figure 2.2 demonstrates the training process at a block level.

Neural networks are inspired by biological neural systems. The basic computational unit of the brain is a neuron and they are connected with synapses. Figure 2.3 compares a biological neuron with a basic mathematical model.

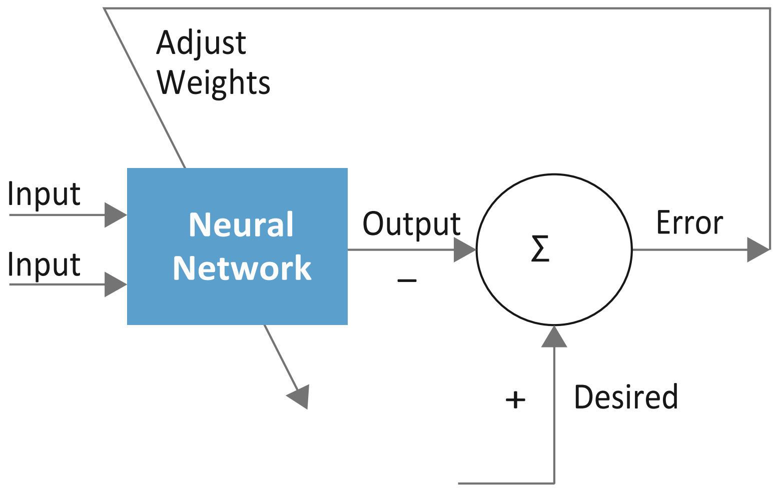
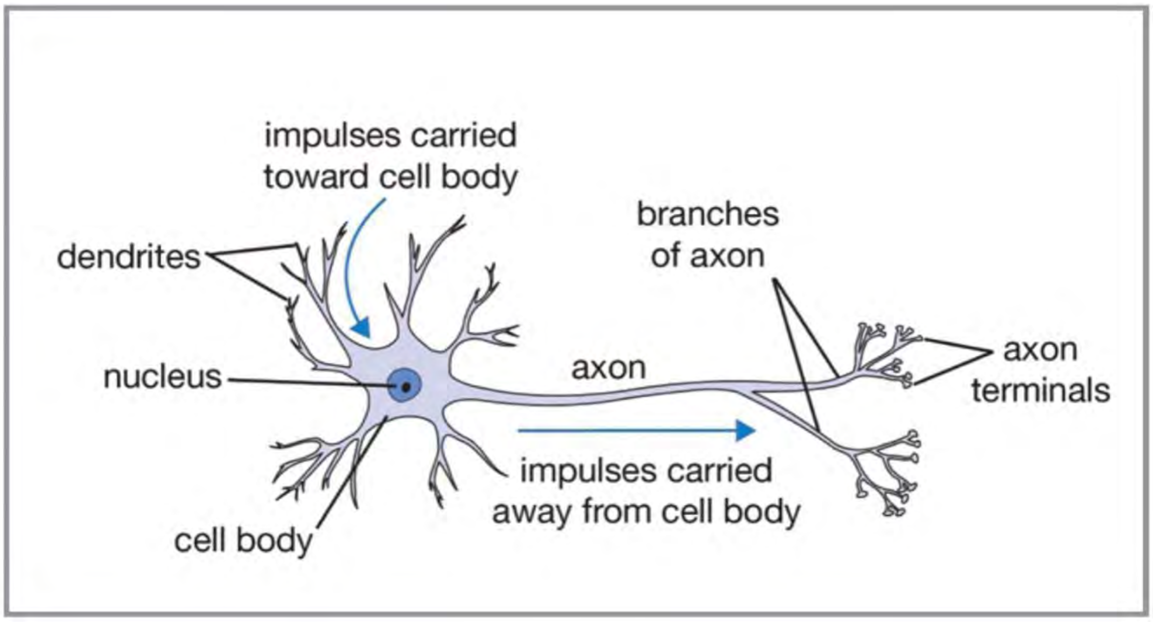


Figure 2.2: Training of Neural Networks


impulses carried toward cell body
dendrites
branches of axon
axon
axon terminals
nucleus
cell body
impulses carried away from cell body

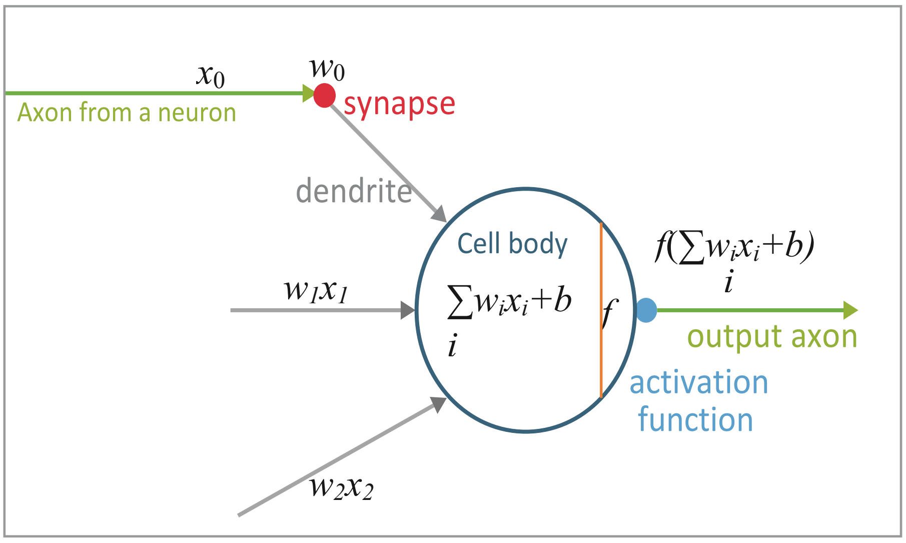


Figure 2.3: Illustration of a biological neuron (up) and its mathematical model (down).

In a real animal neural system, a neuron is perceived to be receiving input signals from its dendrites and producing output signals along its axon. The axon branches out and connects via synapses to dendrites of other neurons. When the combination of input signals reaches some threshold condition among its input dendrites, the neuron is triggered and its activation is communicated to successor neurons.

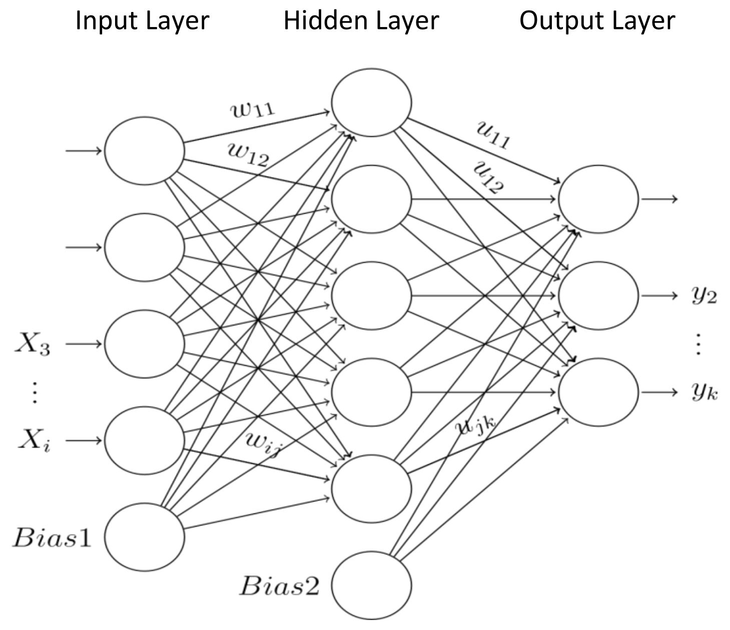


Figure 2.4: A neural network consisting of input, hidden and output layer. A neural network can contain an arbitrary number of hidden layers. Inputs of hidden layer and output layer are weighted by weights w_{ij}, u_{jk} respectively.

In the computational model of neural network, the signals that travel along the axons (e.g., x_0) interact multiplicatively (e.g., w_0x_0) with the dendrites of the other neuron based on the synaptic strength at that synapse (e.g., w_0). Synaptic weights are learnable and control the influence of one neuron or another. The dendrites carry the signal to the cell body, where they all are summed. If the final sum is above a specified threshold, the neuron fires, sending a spike along its axon. In the computational model, it is assumed that the precise timings of the firing do not matter and only the frequency of the firing communicates information. Based on the rate code interpretation, the firing rate of the neuron is modeled with an activation function ***f*** that represents the frequency of the spikes along the axon. A common choice of activation function is sigmoid. In summary, each neuron calculates the dot product of inputs and weights, adds the bias, and applies non-linearity as a trigger function (for example, following a sigmoid response function). The whole network still

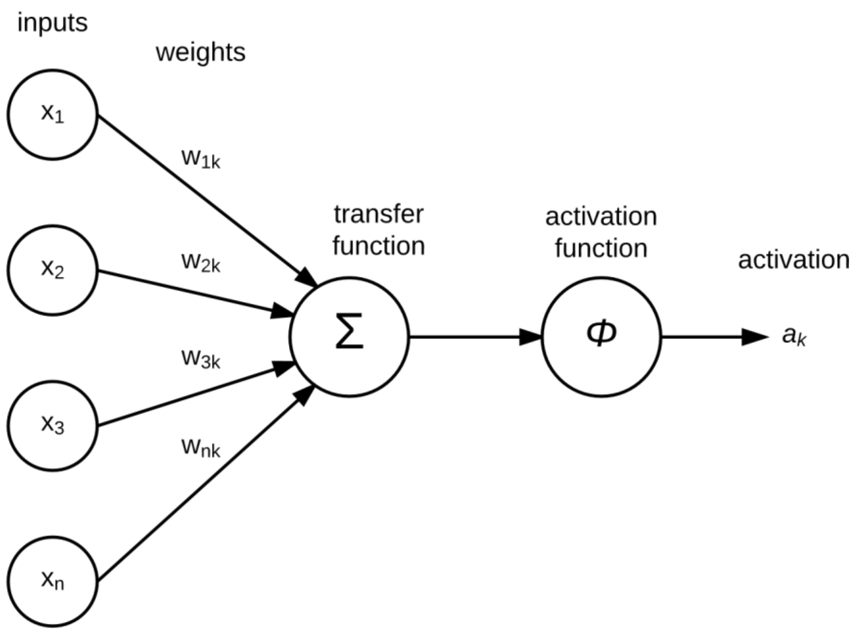


Figure 2.5: Placement of the activation function in the neural network model.

expresses a single differentiable score function: from the raw image pixels on one end to class scores at the other.

2.2.1. Activation Functions

Output of each node is produced by the node's activation function φ that takes weighted inputs of the node as parameters transformed by a transfer function (see Figure 2.5). The transfer function creates a linear combination of weighted inputs in order to feed them to the activation function. To approximate complicated functions, nonlinear activations are often used. The following sections briefly describe different nonlinear activation functions most commonly used in neural networks.

Hyperbolic tangent

One of the most popular activation functions is the hyperbolic tangent function (Equation 2.1). Input x is a weighted linear combination of the inputs of the node. This function works most effectively on inputs in range $(0,1)$, producing outputs in interval $(-1,1)$.

$$\tanh(x) = \frac{\sinh(x)}{\cosh(x)} = \frac{e^x - e^{-x}}{e^x + e^{-x}} = \frac{e^{2x} - 1}{e^{2x} + 1} \tag{2.1}$$

Sigmoid

Logistic sigmoid function (Equation 2.2) is widely used activation function biologically more plausible than hyperbolic tangent. One of the reasons the sigmoid function is broadly used is the fact, the sigmoid function is differentiable at every point.

$$f(x) = \frac{1}{1+e^{-x}} \tag{2.2}$$

ReLU

Rectified linear unit's function (Equation 2.3) is used with the purpose to increase non-linearity of the network. Rectifying neurons are considered to be biologically more plausible than logistic sigmoid or hyperbolic tangent neurons. They benefit from their simplicity, resulting in faster training and performance improvements in particular cases, and therefore often used in DNNs/CNNs. ReLU is given by the equation:

$$f(x) = \max(0,x) \tag{2.3}$$

Figure 2.6 visualizes a comparison of rectifier function and activation functions introduced in this section.

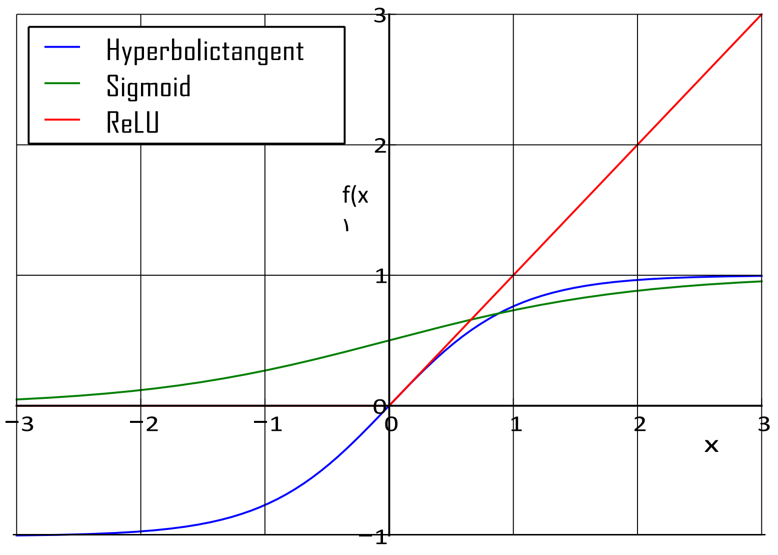


Figure 2.6: Visual comparison of the three most relevant DNNs' activation functions: hyperbolic tangent, sigmoid and rectifier.

2.2.2. Softmax

The softmax activation function (Equation 2.4) is usually used in the last network layer, converting an arbitrary real value to posterior probability of the class $c_k$in range (0,1):

$$p(c_k|x) = \frac{e^{a_k}}{\sum_{i=1}^{m} e^{a_i}} \tag{2.4}$$

where *m* corresponds the number of output nodes (classes) and a_k is the activation value of *k*-th node:

$$a_i = \sum_{j=0}^{d} w_{ij} h_j(x) \tag{2.5}$$

given *i*-th node's weights w_{ij} and the output of the previous layer $h_j(x)$.

2.2.3. Loss Function

To measure a precision of the network outcome, a loss (also cost or objective) function [33] is used. It expresses how much the prediction differs from expected value. The output of the loss function is a real value referred to as the cost or the penalty. An example of a loss function that outputs probabilities, thus often used in visual classification problems is the cross-entropy loss function (Equation 2.6):

$$L = -\sum_{i}^{m} y_i \log p_i \tag{2.6}$$

where *m* is the number of possible classes (nodes) in the output layer, *y* the target vector and *p* the aposterior probability for each class predicted by the network. Evaluated derivatives of a loss function are used in the training phase.

2.2.4. Backpropagation

Backpropagation is a neural network training algorithm. For supervised learning, target classes are essential for error calculation. The error is afterwards backpropagated to every node in previous layers. This error *e* (Equation 2.8) is obtained as a gradient of the loss function *L* with respect to each layer's weights w_{kj}given input of the node *x* and activation function

$$a_j = \phi(\sum_{k=1}^{n} w_{kj} x_k \tag{2.7}$$

$$e = \frac{\partial L}{\partial w_{kj}} = \frac{\partial L}{\partial a_j} \frac{\partial a_j}{\partial w_{kj}} \tag{2.8}$$

Gradient computation demands application of the chain rule in order to compute partial derivative of the loss function L with respect to particular weight w_{kj}. Using the error, weights are updated by an optimization algorithm such as gradient descent.

2.2.5. Gradient Descent

The most common function optimization algorithm used for neural networks is the gradient descent, a first order approximation algorithm that updates weights of the model. The algorithm approaches a local minimum in the direction of the negative gradient of the loss function with respect to the weights. The size of the step is called learning rate. It is a scalar in the range $(0, 1)$, controlling magnitude of network's parameters (weights) change. To perform one update of the weights, the whole training set has to be used. For large training sets, this method might be computationally expensive. A more time efficient gradient descent based optimization method is the stochastic gradient descent or SGD (Equation 2.9). SGD needs only one observation (or subset of the training set) to update model parameters w. As the name suggests, at each weight update a random observation is used. Furthermore, SGD does not tend to end up stuck in a local minima such as ordinary gradient descent (also called batch gradient descent). A disadvantage of SGD is a slower convergence rate than convergence rate of batch gradient descent. Due to its stochasticity, a wrong choice of starting observations may cause algorithm to move further from global minima and make converge problematic.

$$w(t+1) = w(t) - \eta \nabla_w L(w(t)) \tag{2.9}$$

Weights w are being updated by the negative of the gradient of the loss function with respect to the weights. This change is limited by the learning rate η. Root mean square prop or RMSprop is using the same concept of the exponentially weighted average of the gradients like gradient descent with momentum but the difference is the update of parameters.

$$MS\big(\omega(\eta)\big) = \gamma MS\big(\omega(\eta-1)\big) + (1-\gamma)\left(\frac{\partial D(\eta)}{\partial \omega(\eta)}\right)^2$$

$$\omega(\eta+1) = \omega(\eta) - \frac{\lambda}{\sqrt{MS\big(\omega(\eta)\big) + \epsilon}} \frac{\partial D(\eta)}{\partial \omega(\eta)} \tag{2.10}$$

2.2.6. Momentum

Numerous improvements for gradient descent were proposed. One of the most frequently used enhancements is the momentum. Momentum helps to prevent from convergence to a local minima and also speeds up the convergence process by preserving a fraction of previous weight adjustments. Previous weight adjustment is used in current update, multiplied by factor μ, the momentum (Equation 2.11).

$$w(t+1) = w(t) - \eta \nabla_w L(w(t)) + \mu \Delta w(t) \tag{2.11}$$

2.2.7.Nesterov's Accelerated Gradient

Nesterov's accelerated gradient (NAG) is an optimal algorithm for smooth convex optimization proposed by *Nesterov*, with convergence rate of $O(1/t^2)$ after t steps, compared to the one of gradient descent $O(1/t)$. However, for visual problems, optimized functions are barely convex and smooth, thus assumptions under which convergence rate holds are not preserved. Novelty of NAG is in the weight update using gradient on the weights updated by momentum (Equation 2.12).

$$\Delta w(t+1) = \mu \Delta w(t) - \eta \nabla_w L(w(t) + \mu \Delta w(t)) \tag{2.12}$$

2.2.8. Weight Decay

In the training phase, without regularization, weights use to grow to large values slowing down the convergence process. Weight decay (also called L2 regularization) is a way how to prevent weights from growing unboundedly (Equation 2.13). The weight decay parameter λ represents the portion of the weight to be subtracted.

$$w(t+1) = w(t) - \eta \nabla_w L(w(t)) - \lambda w(t) \tag{2.13}$$

2.2.9. Local Response Normalization

Efficiency of a training process is sometimes enhanced by local response normalization(LRN). It is performed over local regions of an input image, centered around point x_k (Equation 2.14). Region has size n and consists of points x_i.

$$x_k = (1 + (\frac{\alpha}{n} \sum_i x_i^2))^\beta \tag{2.14}$$

α and β are arbitrary values specified before the training starts.

2.2.10. Xavier Initialization

The background chapter has introduced issues with initialization of DNNs. If the initial weights are either too large or too small, model is unable to converge to the global minima. To face this problem, Xavier initialization is often used. Weights of the model are randomly initialized, usually taken from the Gaussian distribution with variance determined from (Equation 2.15):

$$var(W) = \frac{1}{n_{in}} \tag{2.15}$$

where W stands for the random distribution of the node to be initialized. Size of the variance depends on number of input connections (n_{in}) to the particular node. Alternative versions of Xavier initialization also exist. They often include the number of outgoing connections in the variance formula.

2.3. Convolutional Neural Networks (CNNs / ConvNets)

A CNN is a special case of the neural network described above. A CNN consists of one or more convolutional layers, often with a subsampling layer, which are followed by one or more fully connected layers as in a standard neural network. The design of a CNN is motivated by the discovery of a visual mechanism, the visual cortex, in the brain (Figure 2.7). The visual cortex contains a lot of cells that are responsible for detecting light in small, overlapping sub-regions of the visual field, which are called receptive fields. These cells act as local filters over the input space, and the more complex cells have larger receptive fields. The convolution layer in a CNN performs the function that is performed by the cells in the visual cortex.

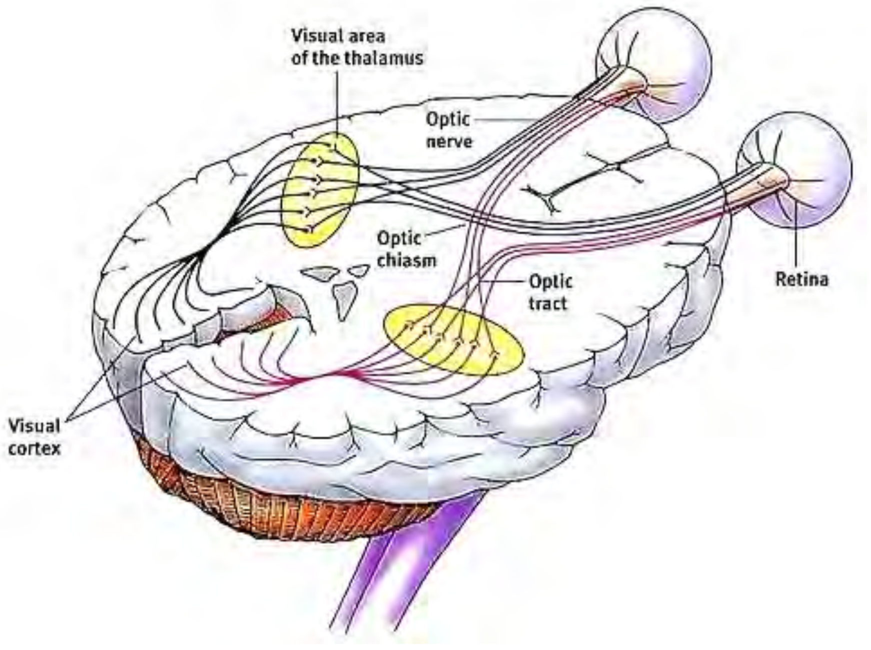
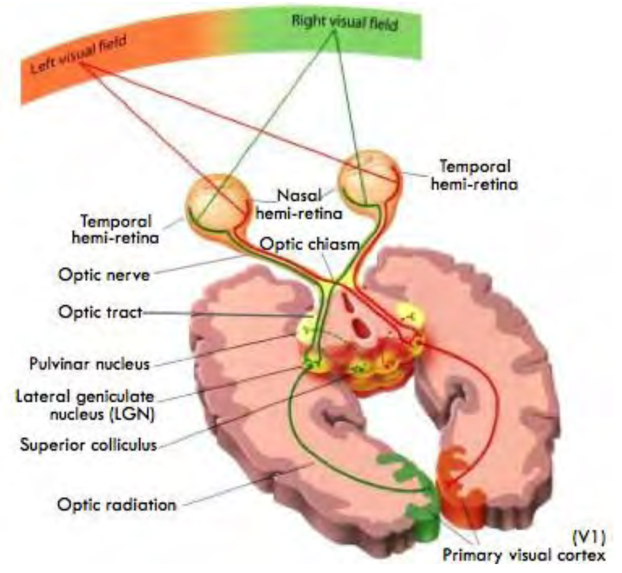
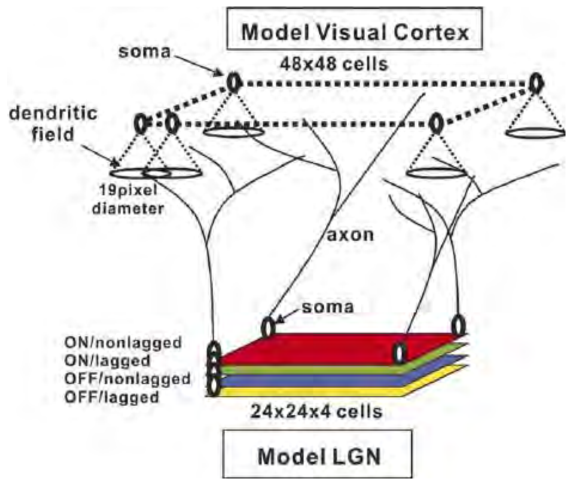


Figure 2.7: i) Visual Cortex of human brain





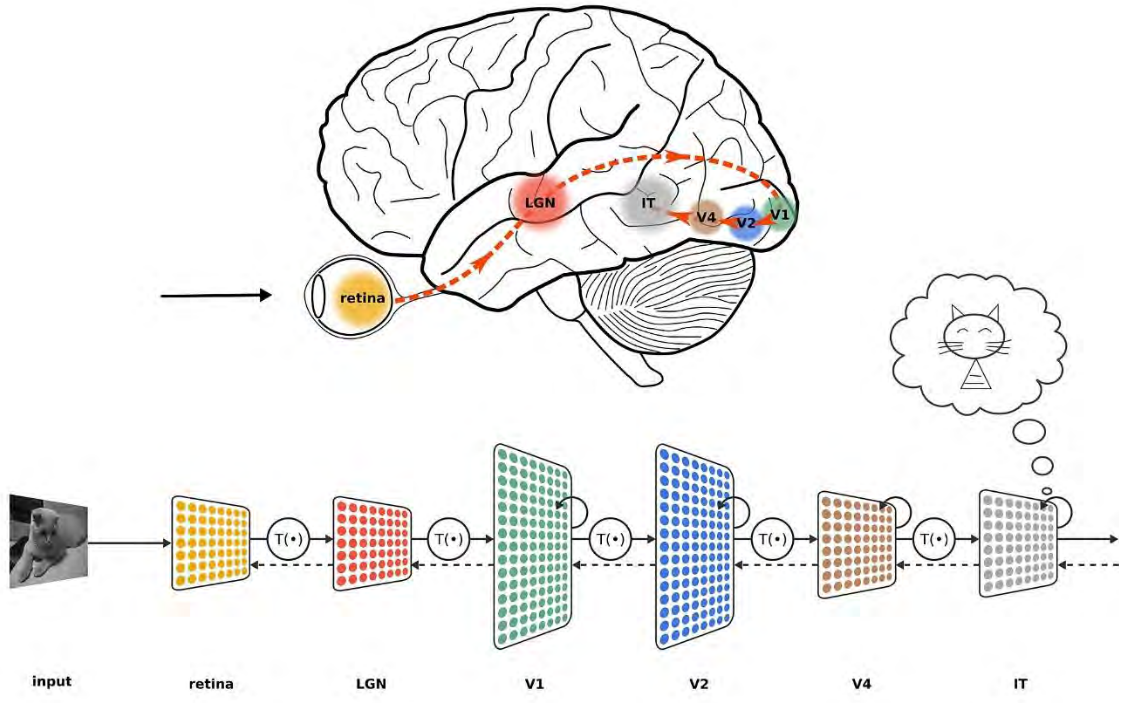


Figure 2.7: ii) A schematic diagram of model LGN and cortex. The model visual cortex is composed of 48×48 model cortical neurons, which have separate dendritic fields. The model LGN is given as four sheets of different cell types. Each sheet is composed of 24×24 model LGN cells, whose receptive field centers are arranged retinotopically.

A typical CNN is shown in Figure 2.9. Each feature of a layer receives inputs from a set of features located in a small neighborhood in the previous layer called a local receptive field. With local receptive fields, features can extract elementary visual features, such as oriented edges, end-points, corners, etc., which are then combined by the higher layers.

In the traditional model of pattern/image recognition, a hand-designed feature extractor gathers relevant information from the input and eliminates irrelevant variabilities. The extractor is followed by a trainable classifier, a standard neural network that classifies feature vectors into classes.

In a CNN, convolution layers play the role of feature extractor. But they are not hand designed. Convolution filter kernel weights are decided on as part of the training process. Convolutional layers are able to extract the local features because they restrict the receptive fields of the hidden layers to be local. For image classification, it is common to use convolutional neural networks (CNNs) as they were designed to extract information from 2D and higher order input spaces. Convolutional neural networks, thanks to their multiple levels of feature extracting layers, use a minimum of preprocessing, hence it is not necessary to consider feature extraction issues. CNN's weights are designed to form a convolutional filter that is replicated over the whole visual field. All units of the convolutional layer share the same weights within the layer, what decreases number of free parameters to learn, thus simplifies training process. The filter is used to convolve an image, each filter convolves pixels it covers. Outputs of all these filters form a feature map. Convolutional layers usually contain several feature maps for richer representation of the image content. Each feature map is produced by a different filter. Convolutional layer is typically defined by number of feature maps, kernel size (size of the filter) and by stride parameter (a size of the step over image pixels when applying filter).

CNNs are used in variety of areas, including image and pattern recognition, speech recognition, natural language processing, and video analysis. There are several reasons that convolutional neural networks are becoming important:

- In traditional models for pattern recognition, feature extractors are hand designed. In CNNs, the weights of the convolutional layer being used for feature extraction as well as the fully connected layer being used for classification are determined during the training process.
- The improved network structures of CNNs lead to savings in memory requirements and computation complexity requirements and, at the same time, give better performance for applications where the input has local correlation (e.g., image and speech).
- Large requirements of computational resources for training and evaluation of CNNs are sometimes met by graphic processing units (GPUs), DSPs, or other silicon architectures optimized for high throughput and low energy when executing the idiosyncratic patterns of CNN computation. In fact, advanced processors such as the Tensilica Vision P5 DSP for Imaging and Computer Vision from Cadence have an almost ideal set of computation and memory resources required for running CNNs at high efficiency.
- In pattern and image recognition applications, the best possible correct detection rates (CDRs) have been achieved using CNNs. For example, CNNs have achieved a CDR of 99.77% using the MNIST database of handwritten digits [59], a CDR of 97.47% with the NORB dataset of 3D objects [60], and a CDR of 97.6% on ~5600 images of more than 10 objects. CNNs not only give the best performance

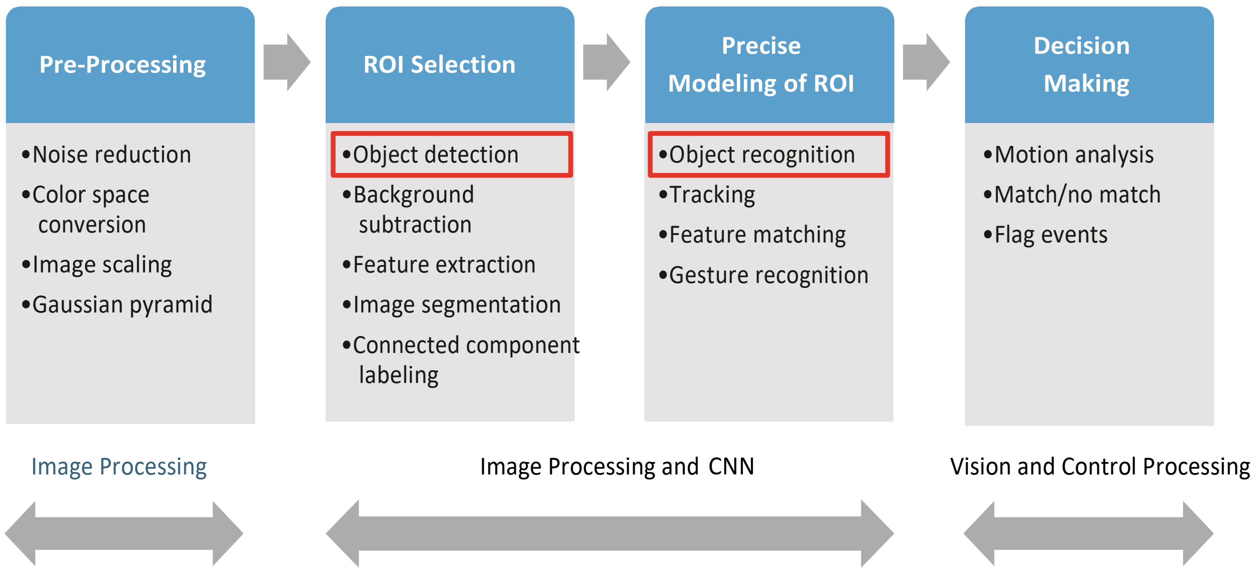


Figure 2.8: Vision algorithm

compared to other detection algorithms, they even outperform humans in cases such as classifying objects into fine-grained categories such as the particular breed of dog or species of bird [61].

- Figure 2.8 shows a typical vision algorithm pipeline, which consists of four stages: pre-processing the image, detecting regions of interest (ROI) that contain likely objects, object recognition, and vision decision making. The pre-processing step is usually dependent on the details of the input, especially the camera system, and is often implemented in a hardwired unit outside the vision subsystem. The decision making at the end of pipeline typically operates on recognized objects—It may make complex decisions, but it operates on much less data, so these decisions are not usually computationally hard or memory-intensive problems. The big challenge is in the object detection and recognition stages, where CNNs are now having a wide impact.

2.3.1 Typical CNN Structure

CNN's structure is inspired by Neocognitron, composed of alternating two types of layers. Layers typically used in convolutional neural networks are listed below:

- Input – This layer will hold the raw pixel values of the image, in this case an image of same height and width, and with three color channels R,G,B.
- Convolutional - Nodes of a convolutional layer perform convolution on a different parts of the image. This layer serves as a feature extractor. This layer will compute the output of neurons that are connected to local regions in the input, each computing a dot product between their weights and a small region they are connected to in the input volume. This may result in volume as output instead of an image.
- ReLU (Rectified Linear Unit)-layer will apply an elementwise activation function, such as the max(0, x) thresholding at zero. This leaves the size of the volume unchanged. This layer introduces non-linearity in the system.
- Pooling/Subsampling - This layer subsamples feature maps to reduce variance within local regions of the image. Pool layer will perform a down-sampling operation along the spatial dimensions (width, height), resulting in volume of reduced size. It splits the image into rectangular regions and takes out value determined by the type of pooling layer. The most popular type of pooling layer in CNNs is the max-pooling layer, which extracts maximum value of the sub-regions of the feature map.
- Fully connected - As with ordinary Neural Networks and as the name implies, each neuron in this layer will be connected to all the numbers in the previous volume and each neuron in this layer takes an input from all the previous layer's neurons. This layer will compute the class scores, resulting in volume of size $1\times1\times N$. The reasoning of the network is performed by its fully connected layers.
- Classifier - Outputs posterior probabilities for each class.

Standard convolutional neural network consists of one or more pairs of convolutional layer and subsequent max-pooling layer followed by one or more fully connected layers using rectifying activation function. The output layer is often constructed as a combination of the softmax activation function and the cross entropy loss function (Equation 2.6).

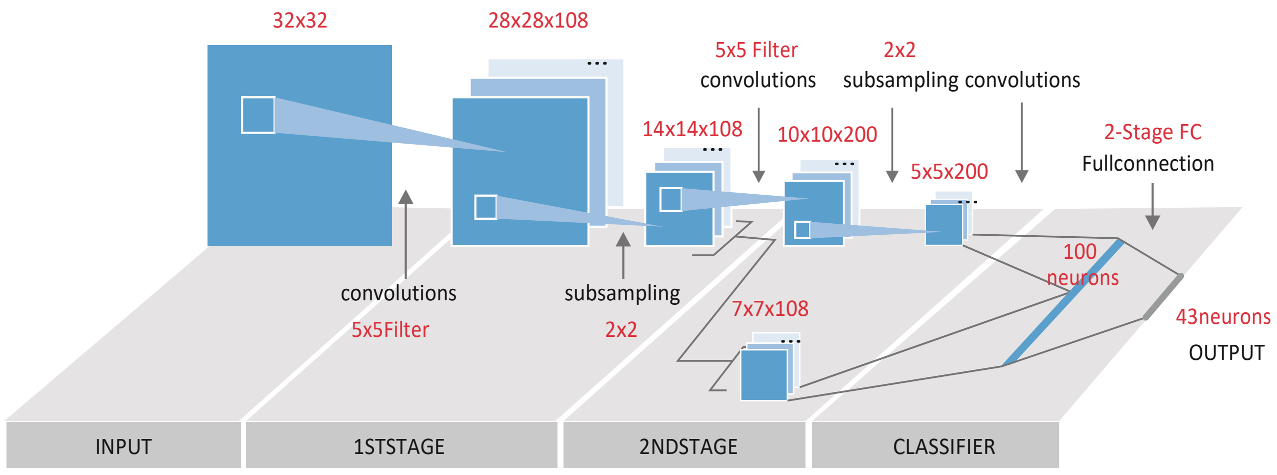


Figure 2.9: Typical block diagram of a CNN

2.3.2. Layers of CNNs

By stacking multiple and different layers in a CNN, complex architectures are built for classification problems. Four types of layers are most common: convolution layers, pooling/sub-sampling layers, non-linear layers, and fully connected layers.

Convolution Layers

The convolution operation extracts different features of the input. The first convolution layer extracts low-level features like edges, lines, and corners. Higher-level layers extract higher-level features. Figure 2.10 illustrates the process of 3D convolution used in CNNs. The input is of size N × N × D and is convolved with H kernels, each of size k × k × D separately. Convolution of an input with one kernel produces one output feature, and with H kernels independently produces H features. Starting from top-left corner of the input, each kernel is moved from left to right, one element at a time. Once the top-right corner is reached, the kernel is moved one element in a downward direction, and again the kernel is moved from left to right, one element at a time. This process is repeated until the kernel reaches the bottom-right corner. For example, when N = 32 and k = 5, there are 28 unique positions from left to right and 28 unique positions from top to bottom that the kernel can take. Corresponding to these positions, each feature in the output will contain 28×28 (i.e., (N-k+1) × (N-k+1)) elements. For each position of the kernel in a sliding window process, k × k × D elements of input and k × k × D elements of kernel are element-by-element multiplied and accumulated. So to create one element of one output feature, k × k × D multiply-accumulate operations are required.

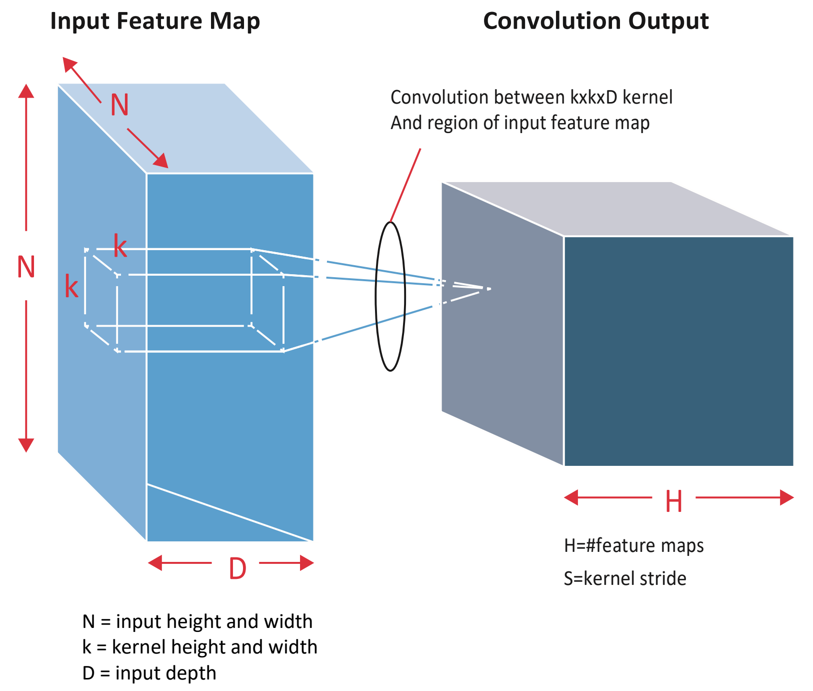

Input Feature Map
Convolution Output
N
N
k
k
D
Convolution between kxkxD kernel
And region of input feature map
H
H=#feature maps
S=kernel stride
N = input height and width
k = kernel height and width
D = input depth

Figure 2.10: A representation of convolution process

Let $\mathbf{W_i}$ be a filter set with dimension $C_i \times C_{i-1} \times N_i \times N_i$, where C_i and C_{i-1} is the number of channels of the output and input of this layer respectively, and N_i be the square-size parameter of the filters. The parameter, C_i represents the number of filters in the set $\mathbf{W_i}$. Each of the filters has a corresponding bias term, resulting a bias vector $\mathbf{b_i}$ with C_i number of elements. Hence, the output of this layer is obtained from the output of the previous layer, bias term and corresponding filter set as

$$\mathbf{X_i} = \mathbf{W_i} * \mathbf{X_{i-1}} + \mathbf{b_i} \qquad (2.16)$$

where $*$represents the linear convolution operation. This operation results in the dimension of output $\mathbf{X_i}$ to be $C_i \times M_{vi} \times M_{hi}$ from input $\mathbf{X_{i-1}}$ with shape $C_{i-1} \times Mv_{(i-1)} \times Mh_{(i-1)}$. There is a positive parameter called 'stride' which can be set to a value that will cause the spatial dimensions to change resulting in up-sampling or down-sampling. The spatial dimensions remain the same when the parameter is set to 1. If it is set to value greater than unity then the dimensions decrease. And, if it is set to a value less than unity, the spatial dimensions increase. A general tendency is to set the parameter to 1 and the dimensionality reduction, when required, is obtained using a pooling layer. In model description, convolution layer is referred to as $CN\ (C_i, N_i)$.

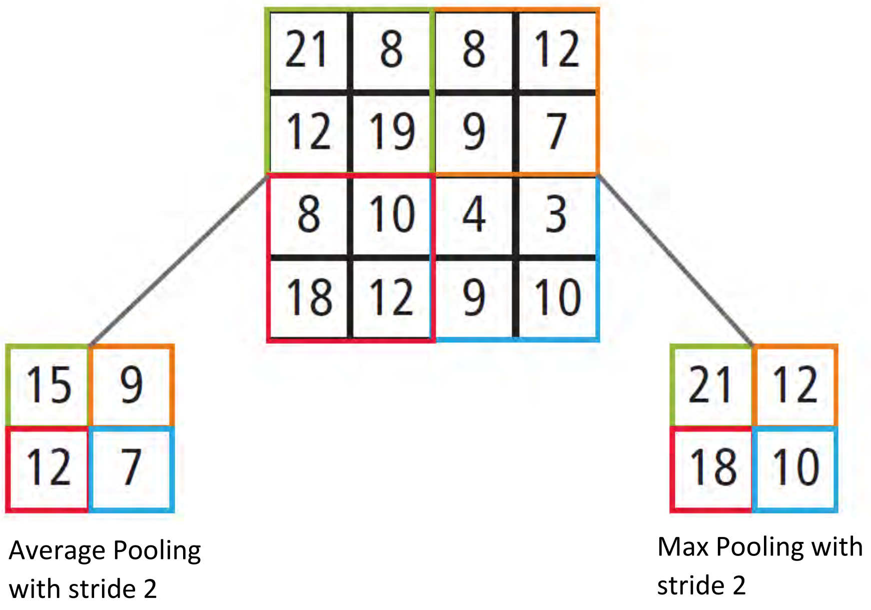


Figure 2.11: A representation of max pooling and average pooling

Pooling/Subsampling Layers

The pooling/subsampling layer reduces the resolution of the features. It makes the features robust against noise and distortion. Its function is to progressively reduce the spatial size of there presentation to reduce the amount of parameters and computation in the network, and hence to also control overfitting. There are two ways to do pooling: max pooling and average pooling. In both cases, the input is divided into non-overlapping two-dimensional spaces. For example, in Figure 2.9, layer 2 is the pooling layer. Each input feature is 28×28 and is divided into 14×14 regions of size 2×2. For average pooling, the average of the four values in the region are calculated. For max pooling, the maximum value of the four values is selected.

Figure 2.11 elaborates the pooling process further. The input is of size 4×4. For 2×2 subsampling, a 4×4 image is divided into four non-overlapping matrices of size 2×2. In the case of max pooling, the maximum value of the four values in the 2×2 matrix is the output. In case of average pooling, the average of the four values is the output. Please note that for the output with index (2,2), the result of averaging is a fraction that has been rounded to nearest integer.

Non-linear Layers

Neural networks in general and CNNs in particular rely on a non-linear “trigger” function to signal distinct identification of likely features on each hidden layer. CNNs may use a variety of specific functions —such as rectified linear units (ReLUs) and continuous trigger (non-linear) functions—to efficiently implement this non-linear triggering.

ReLU

A ReLU implements the function

$$\mathbf{X_i} = max\,(\mathbf{0}, \mathbf{X_{i-1}}) \qquad (2.17)$$

In other words, only non-negative values are kept as is and the other values are set to zero. So the input and output sizes of this layer are the same. It increases the nonlinear properties of the decision function and of the overall

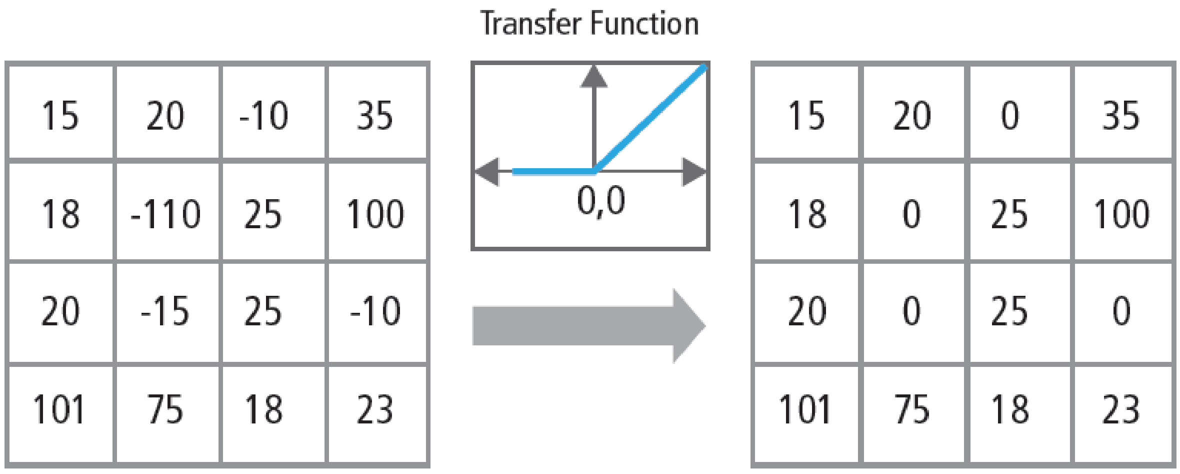


Figure 2.12: A representation of ReLU functionality

network without affecting the receptive fields of the convolution layer. In comparison to the other non-linear functions used in CNNs (e.g., hyperbolic tangent, absolute of hyperbolic tangent, and sigmoid), the advantage of a ReLU is that the network trains many times faster. In addition, the ReLU unit helps the neural network to attain a better sparse representation ([52]). It is customary for convolution layer to be followed by ReLU activation. ReLU functionality is illustrated in Figure 2.12, with its transfer function plotted above the arrow.

Continuous Trigger (Non-Linear) Function

The non-linear layer operates element by element in each feature. A continuous trigger function can be hyperbolic tangent (Figure 2.13), absolute of hyperbolic tangent (Figure 2.14), or sigmoid (Figure 2.15). Figure 2.16 demonstrates how non-linearity gets applied element by element.

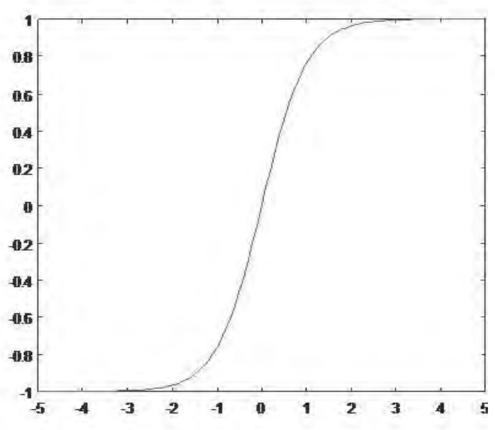


Figure 2.13: The hyperbolic tangent

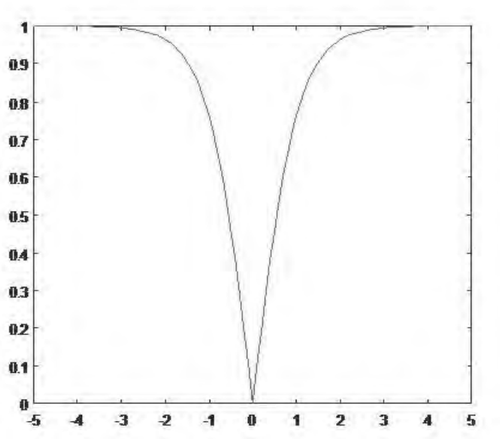


Figure 2.14: Absolute of function
hyperbolic tangent function

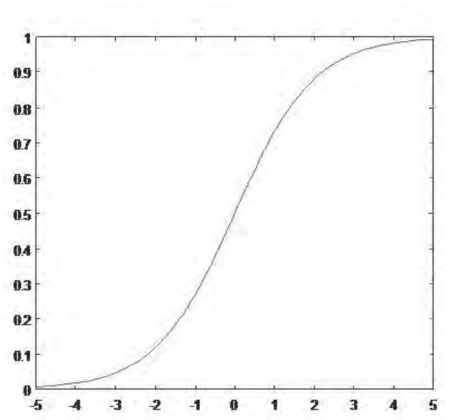


Figure 2.15: The sigmoid function

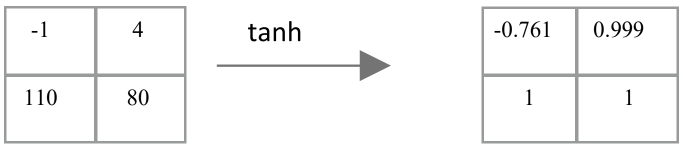


Figure 2.16: A representation of tanh processing

Fully Connected layers

Fully connected layers are often used as the final layers of a CNN. These layers mathematically sum a weighting of the previous layer of features, indicating the precise mix of “ingredients” to determine a specific target output result. In case of a fully connected layer, all the elements of all the features of the previous layer get used in the calculation of each element of each output feature.

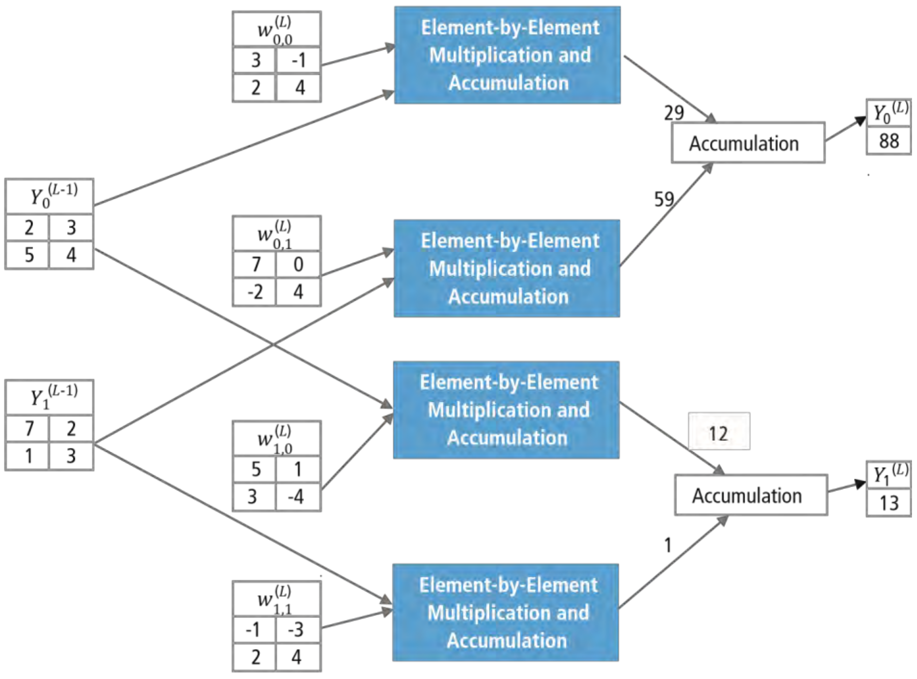


Figure 2.17: Processing of a fully connected layer

Figure 2.17 explains the fully connected layer L. Layer L-1 has two features, each of which is 2×2, i.e., has four elements. Layer L has two features, each having a single element.

2.4. Advantage of CNN over NN:

While neural networks and other pattern detection methods have been around for the past 50 years, there has been significant development in the area of convolutional neural networks in the recent past. This section covers the advantages of using CNN for image recognition.

- **Ruggedness to shifts and distortion in the image**

Detection using CNN is rugged to distortions such as change in shape due to camera lens, different lighting conditions, different poses, presence of partial occlusions, horizontal and vertical shifts, etc. However, CNNs are shift invariant since the same weight configuration is used across space. In theory, we also can achieve shift invariantness using fully connected layers. But the outcome of training in this case is multiple units with identical weight patterns at different locations of the input. To learn these weight configurations, a large number of training instances would be required to cover the space of possible variations.

- **Fewer memory requirements**

In this same hypothetical case where we use a fully connected layer to extract the features, the input image of size 32×32 and a hidden layer having 1000 features will require an order of 10^6 coefficients, a huge memory requirement. In the convolutional layer, the same coefficients are used across different locations in the space, so the memory requirement is drastically reduced.

- **Easier and better training**

Again using the standard neural network that would be equivalent to a CNN, because the number of parameters would be much higher, the training time would also increase proportionately. In a CNN, since the number of parameters is drastically reduced, training time is proportionately reduced. Also, assuming perfect training, we can design a standard neural network whose performance would be same as a CNN. But in practical training, a standard neural network equivalent to CNN would have more parameters, which would lead to more noise addition during the training process. Hence, the performance of a standard neural network equivalent to a CNN will always be poorer.

2.5. Conclusion

In this chapter a brief description of NN and CNN are presented. The reason for applying CNN in CR task is also explained. In the next chapter different models of CNN used in this experiment of Bangla CR task will be presented along with the comparison of their performances in terms of recognition accuracy rates.

Chapter 3

Proposed DCNN Architecture

3.1. Introduction

This chapter presents the structure of DCNN models used in this experiment. The performances of the different DCNN structures have been used for experiment in this thesis are shown. Number of kernels in different convolution layers, sizes of the kernels, depth of the network and number of neurons in the classifier layers have their effects on the performance of the recognizer. The architecture that gives the best output (Model no. 4) in terms of the recognition accuracy rates are presented in this chapter.

3.2. DCNN Architectures

In this work five different architectures of DCNN are used for recognition task and their performances are compared to determine the most optimized network size for better recognition accuracy. Among these five architectures, model 4 gives the best result. The descriptions of the five architectures are given below:

3.2.1. Model 1

Architecture

Model 1 consists of 3 convolutional layers and 1 affine (fully connected) layer. It takes 32×32 RGB images as input. 1st, 2nd and 3rd convolution layers contain 32, 64 and 128 numbers of receptive fields (kernels) respectively. The kernels in all the convolutional layers are of equal size: 3×3.

After 1st convolution layer ReLU is used as activation function, but no sub-sampling layer is used. So, after the 1st convolution between 32×32 input image size for each channel (RGB) and 32 nos of 3×3 kernels for each channel, the size of the feature maps become 32×32×32. Padding 1and stride 1 are used for the convolution operation.

Application of ReLU activation does not change the number of parameters. Pooling is not used in the first layer.

After each of 2nd and 3rd convolution layers, ReLU function and MaxPooling (sub-sampling) with stride 2 are used in Model 1. Padding 1 and stride 1 are used for the convolution operation. Both pooling height and width are 2. So, after the 2nd convolution between 32×32×32 feature map size and 64 nos of 3×3 kernels, the size of the feature maps becomes 32×32×64. After first pooling, the feature map size reduced to 16×16×64. After 3rd convolution layer with 128 nos of kernel, the feature size becomes 16×16×128. After second pooling, the feature map size reduced to 8×8×128. And at the end one fully-connected layer with 50 neurons are used.

Figure

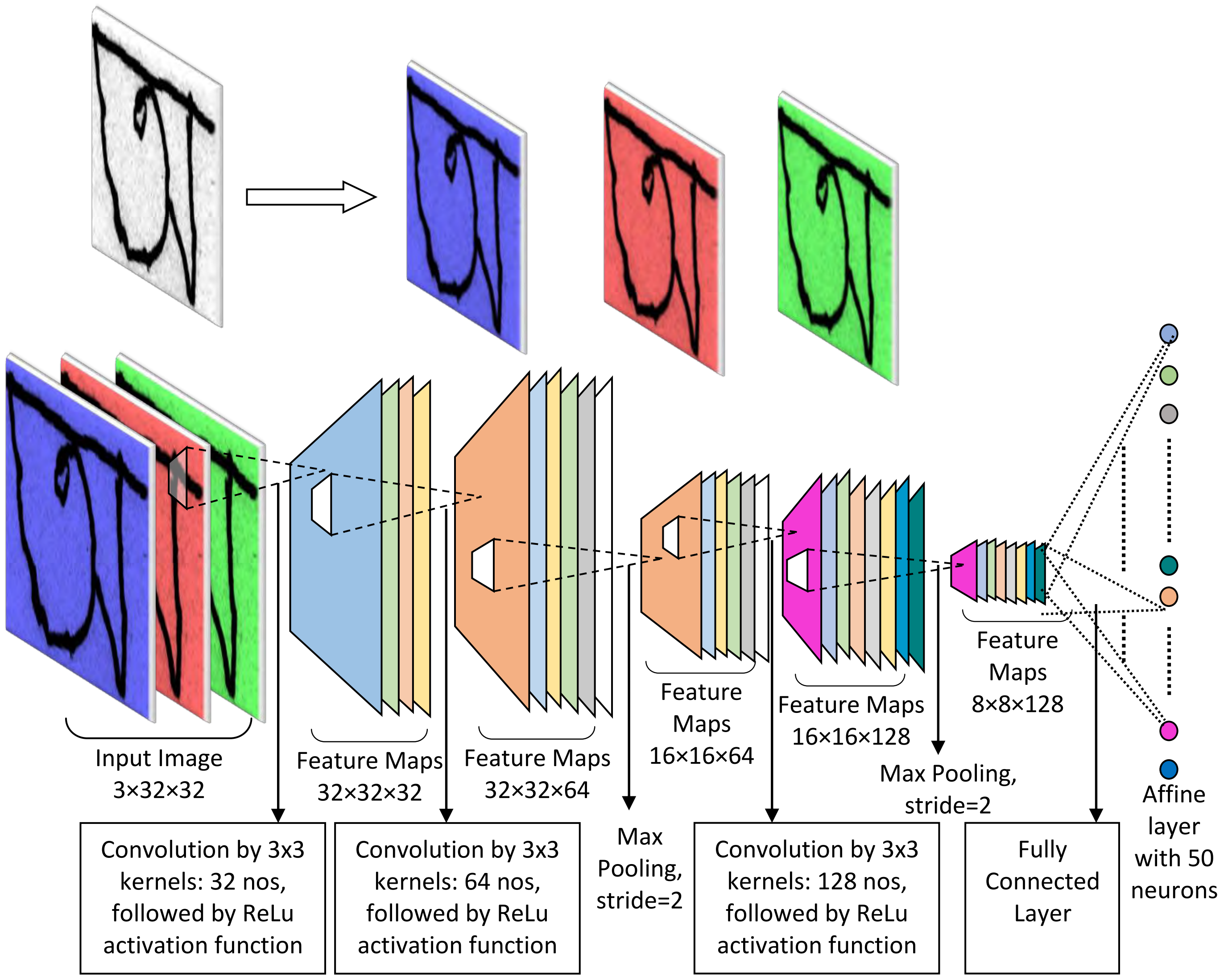

Input Image 3×32×32
Feature Maps 32×32×32
Feature Maps 32×32×64
Feature Maps 16×16×64
Feature Maps 16×16×128
Feature Maps 8×8×128
Max Pooling, stride=2
Max Pooling, stride=2
Affine layer with 50 neurons
Convolution by 3x3 kernels: 32 nos, followed by ReLu activation function
Convolution by 3x3 kernels: 64 nos, followed by ReLu activation function
Convolution by 3x3 kernels: 128 nos, followed by ReLu activation function
Fully Connected Layer

Figure 3.1 : Model 1 DCNN architecture for BHCR

Function

Let, the input image be X (3×32×32),

Layer 1 (Conv)	:	$L_1 \equiv W_1 * X + B_1$
Layer 2 (ReLU)	:	$L_2 \equiv \max(0, L_1)$
Layer 3 (Conv)	:	$L_3 \equiv W_2 * L_2 + B_2$
Layer 4 (ReLU)	:	$L_4 \equiv max(0, L_3)$
Layer 5 (Pooling)	:	$L_5 \equiv MaxPooling(L_4, \text{Size:} 2\times2, \text{Stride} = 2)$
Layer 6 (Conv)	:	$L_6 \equiv W_3 * L_5 + B_3$
Layer 7 (ReLU)	:	$L_7 \equiv max(0, L_6)$
Layer 8 (Pooling)	:	$L_8 \equiv MaxPooling(L_7, \text{Size:} 2\times2, \text{Stride} = 2)$
Layer 9 (Affine)	:	$L_9 \equiv W_4 L_8 + B_4$
Layer 10 (Softmax)	:	$L_{10} \equiv \text{SoftMax}(L_9)$

3.2.2. Model 2

Architecture

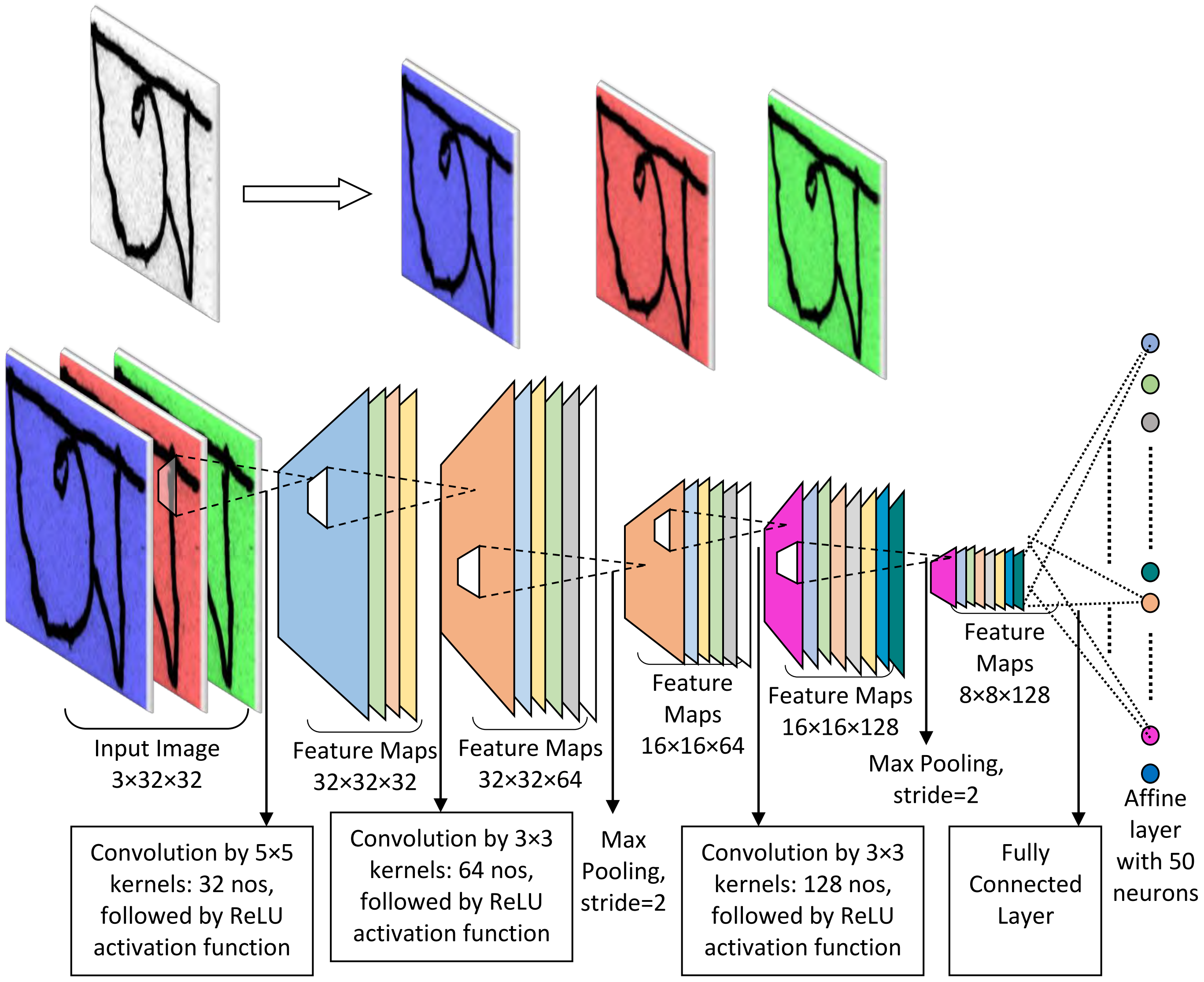
Similar to Model 1, Model 2 consists of 3 convolutional layers and 1 affine (fully connected) layer. It takes 32×32 RGB images as input. 1st, 2nd and 3rd convolution layers contain 32, 64 and 128 numbers of receptive fields (kernels) respectively. The difference between model 1 and Model 2 is: unlike model 1, the kernel size in the first convolutional layer in model 2 is 5×5, the kernel size in the 2nd and 3rdconvolutional layers are of equal size: 3×3.

After 1st convolution layer ReLU is used as activation function, but no sub-sampling layer is used. So, after the 1st convolution between 32×32 input image size for each channel (RGB) and 32 nos of 3×3 kernels for each channel, the size of the feature maps become 32×32×32. Padding 2 and stride 1 are used for the convolution operation. Application of ReLU activation does not change the number of parameters. Pooling is not used in the first layer.

After each of 2nd and 3rd convolution layers, ReLU function and MaxPooling (sub-sampling) with stride 2 are used in Model 2. Padding 1 and stride 1 are used for the convolution operation. Both pooling height and width are 2. So, after the 2nd convolution between 32×32×32 feature map size and 64 nos of 3×3 kernels, the size of the feature maps becomes 32×32×64. After first pooling, the feature map size reduced to 16×16×64. After 3rd convolution layer with 128 nos of kernel, the feature size becomes 16×16×128. After second pooling, the feature map size reduced to 8×8×128. And at the end one fully-connected layer with 50 neurons are used.

Appendix

Figure



Input Image
3×32×32
Feature Maps
32×32×32
Feature Maps
32×32×64
Feature
Maps
16×16×64
Feature Maps
16×16×128
Feature
Maps
8×8×128
Max Pooling,
stride=2
Affine
layer
with 50
neurons
Convolution by 5×5 kernels: 32 nos, followed by ReLU activation function
Convolution by 3×3 kernels: 64 nos, followed by ReLU activation function
Max
Pooling,
stride=2
Convolution by 3×3 kernels: 128 nos, followed by ReLU activation function
Fully
Connected
Layer

Figure 3.2 : Model 2 DCNN architecture for BHCR

Function

Let, the input image be X (3×32×32),

Layer 1 (Conv)	:	$L_1 \equiv W_1 * X + B_1$
Layer 2 (ReLU)	:	$L_2 \equiv \max(0, L_1)$
Layer 3 (Conv)	:	$L_3 \equiv W_2 * L_2 + B_2$
Layer 4 (ReLU)	:	$L_4 \equiv max(0, L_3)$
Layer 5 (Pooling)	:	$L_5 \equiv MaxPooling(L_4, \text{Size:}2\times2, \text{Stride} = 2)$
Layer 6 (Conv)	:	$L_6 \equiv W_3 * L_5 + B_3$
Layer 7 (ReLU)	:	$L_7 \equiv max(0, L_6)$
Layer 8 (Pooling)	:	$L_8 \equiv MaxPooling(L_7, \text{Size: } 2\times2, \text{Stride} = 2)$
Layer 9 (Affine)	:	$L_9 \equiv W_4 L_8 + B_4$
Layer 10 (Softmax)	:	$L_{10} \equiv \text{SoftMax}(L_9)$

3.2.3. Model 3

Architecture

Model 3 is similar to model 2, but unlike model 2, it has 2 affine layers at the end. Model 3 consists of 3 convolutional layers and 2 affine (fully connected) layers. It takes 32×32 RGB images as input. 1st, 2nd and 3rd convolution layers contain 32, 64 and 128 numbers of receptive fields (kernels) respectively. The kernel size in the first convolutional layer in model 3 is 5×5, the kernel size in the 2nd and 3rd convolutional layers are of equal size: 3×3.

After 1st convolution layer ReLU is used as activation function, but no sub-sampling layer is used. So, after the 1st convolution between 32×32 input image size for each channel (RGB) and 32 nos of 3×3 kernels for each channel, the size of the feature maps become 32×32×32. Padding 2 and stride 1 are used for the convolution operation. Application of ReLU activation does not change the number of parameters. Pooling is not used in the first layer.

After each of 2nd and 3rd convolution layers, ReLU function and MaxPooling (sub-sampling) with stride 2 are used in Model 3. Padding 1 and stride 1 are used for the convolution operation. Both pooling height and width are 2. So, after the 2nd convolution between 32×32×32 feature map size and 64 nos of 3×3 kernels, the size of the feature maps becomes 32×32×64. After first pooling, the feature map size reduced to 16×16×64. After 3rd convolution layer with 128 nos of kernel, the feature size becomes 16×16×128. After second pooling, the feature map size reduced to 8×8×128. And at the end two fully-connected layers with 3000 and 50 neurons respectively are used as classifier.

Figure

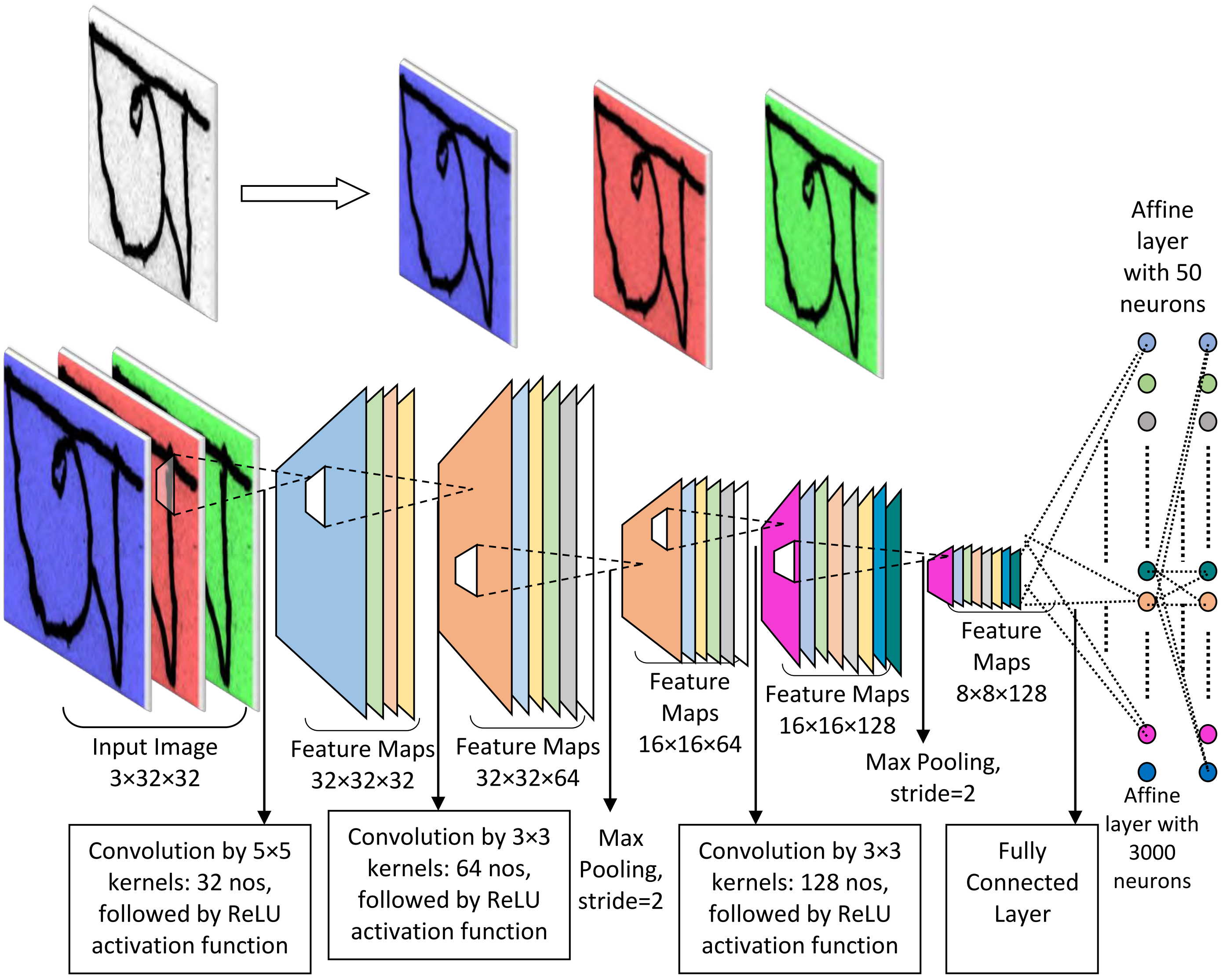

Affine layer with 50 neurons
Input Image 3×32×32
Feature Maps 32×32×32
Feature Maps 32×32×64
Feature Maps 16×16×64
Feature Maps 16×16×128
Feature Maps 8×8×128
Max Pooling, stride=2
Affine layer with 3000 neurons
Convolution by 5×5 kernels: 32 nos, followed by ReLU activation function
Convolution by 3×3 kernels: 64 nos, followed by ReLU activation function
Max Pooling, stride=2
Convolution by 3×3 kernels: 128 nos, followed by ReLU activation function
Fully Connected Layer

Figure 3.3 : Model 3 DCNN architecture for BHCR

Function

Let, the input image be X (3×32×32),

Layer 1 (Conv)	:	$L_1 \equiv W_1 * X + B_1$
Layer 2 (ReLU)	:	$L_2 \equiv \max(0, L_1)$
Layer 3 (Conv)	:	$L_3 \equiv W_2 * L_2 + B_2$
Layer 4 (ReLU)	:	$L_4 \equiv max(0, L_3)$
Layer 5 (Pooling)	:	$L_5 \equiv MaxPooling(L_4, \text{Size:}2\times2, \text{Stride} = 2)$
Layer 6 (Conv)	:	$L_6 \equiv W_3 * L_5 + B_3$
Layer 7 (ReLU)	:	$L_7 \equiv max(0, L_6)$
Layer 8 (Pooling)	:	$L_8 \equiv MaxPooling(L_7, \text{Size: } 2\times2, \text{Stride} = 2)$
Layer 9 (Affine)	:	$L_9 \equiv W_4 L_8 + B_4$
Layer 10 (Affine)	:	$L_{10} \equiv W_5 L_9 + B_5$
Layer 11 (Softmax)	:	$L_{11} \equiv \text{SoftMax}(L_{10})$

3.2.4. Model 4 (Proposed DCNN)

Architecture

Among the different models used in this experiment, the best DCNN architecture is model 4. Model 4 is almost same to model 3, the only difference is in the number of neurons used in the first affine layer.

Proposed DCNN architecture consists of 3 convolutional layers and 2 affine (fully connected) layers. It takes 32×32 RGB images as input. 1st, 2nd and 3rd convolution layers contain 32, 64 and 128 numbers of receptive fields (kernels) respectively. The kernel size in the first convolutional layer in model 4 is 5×5, the kernel sizes in the 2nd and 3rd convolutional layers are of equal size: 3×3. After 1st convolution layer ReLU is used as activation function, but no sub-sampling layer is used. So, after the 1st convolution between 32×32 input image size for each channel (RGB) and 32 nos of 5×5 kernels for each channel, the size of the feature maps become 32×32×32. Padding 2 and stride 1 are used for the convolution operation. Application of ReLU activation does not change the number of parameters. Pooling is not used in the first layer.

After each of 2nd and 3rd convolution layers, ReLU function and MaxPooling (sub-sampling) with stride 2 are used in Model 4. Padding 1 and stride 1 are used for the convolution operation. Both pooling height and width are 2. So, after the 2nd convolution between 32×32×32 feature map size and 64 nos of 3×3 kernels, the size of the feature maps becomes 32×32×64. After first pooling, the feature map size reduced to 16×16×64. After 3rd convolution layer with 128 nos of kernel, the feature size becomes 16×16×128. After second pooling, the feature map size reduced to 7×7×128. And at the end two fully-connected layers with 3500 and 50 neurons respectively are used as classifier.

There are 2,400 parameters (for each of 3 channels of inputs, 32 numbers of 5×5 sized kernels) as weights and 32 parameters as bias in the first layer of CNN. Layer 2 contains 18,432 weights and 64 bias parameters. There are 73728 no weight parameters and 128 bias parameters in layer 3. Layer four has 21,952,000 weight parameters and 3,500 bias parameters. And final layer (layer 5) contains 175,000 weight and 50 bias parameters. The network contains a total of 22,225,334 no of parameters for weights and biases.

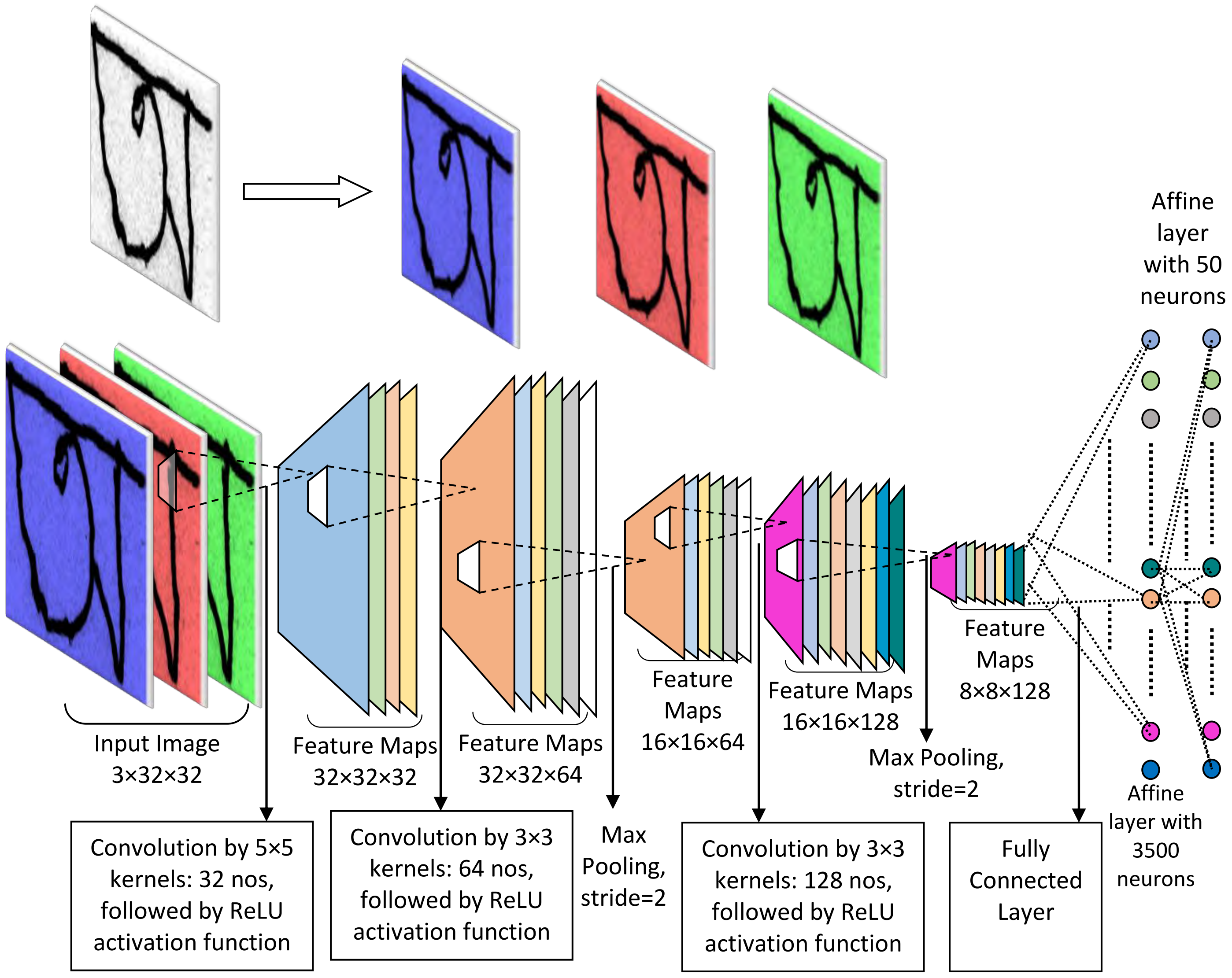
Table 3.1: Parameters setup for DCNN

Layer	Operation of Layer	Number of Feature maps	Size of feature maps	Size of kernel	Number of parameters
X	Input Layer	3	32×32	-	-
C1	Convolution	32	32×32	5×5	3×32×5×5+32 =2,432
RL1	ReLU	32	32×32	-	-
C2	Convolution	64	32×32	3×3	32×64×3×3+64 =18,496
RL2	ReLu	64	32×32	-	-
S2	Max-pooling	64	16×16	2×2	-
C3	Convolution	128	16×16	3×3	64×128×3×3+128 =73,856
RL3	ReLU	128	16×16	-	-
S3	Max-pooling	128	8×8	2×2	-
FC1	Affine	3500	1×1	-	128×7×7×3500 +3500=21955500
FC2	Affine	50	1×1	-	3500×50+50 =175050
				Total:	22,225,334

Let, the input image be X (3×32×32),

Layer 1 (Conv)	:	$L_1 \equiv W_1 * X + B_1$
Layer 2 (ReLU)	:	$L_2 \equiv \max(0, L_1)$
Layer 3 (Conv)	:	$L_3 \equiv W_2 * L_2 + B_2$
Layer 4 (ReLU)	:	$L_4 \equiv max(0, L_3)$
Layer 5 (Pooling)	:	$L_5 \equiv MaxPooling(L_4, \text{Size:}2\times2, \text{Stride} = 2)$
Layer 6 (Conv)	:	$L_6 \equiv W_3 * L_5 + B_3$
Layer 7 (ReLU)	:	$L_7 \equiv max(0, L_6)$
Layer 8 (Pooling)	:	$L_8 \equiv MaxPooling(L_7, \text{Size: } 2\times2, \text{Stride} = 2)$
Layer 9 (Affine)	:	$L_9 \equiv W_4 L_8 + B_4$
Layer 10 (Affine)	:	$L_{10} \equiv W_5 L_9 + B_5$
Layer 11 (Softmax)	:	$L_{11} \equiv SoftMax(L_{10})$

Figure



Affine layer with 50 neurons
Input Image 3×32×32
Feature Maps 32×32×32
Feature Maps 32×32×64
Feature Maps 16×16×64
Feature Maps 16×16×128
Feature Maps 8×8×128
Max Pooling, stride=2
Affine layer with 3500 neurons
Convolution by 5×5 kernels: 32 nos, followed by ReLU activation function
Convolution by 3×3 kernels: 64 nos, followed by ReLU activation function
Max Pooling, stride=2
Convolution by 3×3 kernels: 128 nos, followed by ReLU activation function
Fully Connected Layer

Figure 3.4 : Proposed DCNN architecture (Model 4) for BHCR

Function

Let, the input image be X (3×32×32),

Layer 1 (Conv)	:	$L_1 \equiv W_1 * X + B_1$
Layer 2 (ReLU)	:	$L_2 \equiv \max(0, L_1)$
Layer 3 (Conv)	:	$L_3 \equiv W_2 * L_2 + B_2$
Layer 4 (ReLU)	:	$L_4 \equiv max(0, L_3)$
Layer 5 (Pooling)	:	$L_5 \equiv MaxPooling(L_4, \text{Size:}2\times2, \text{Stride} = 2)$
Layer 6 (Conv)	:	$L_6 \equiv W_3 * L_5 + B_3$
Layer 7 (ReLU)	:	$L_7 \equiv max(0, L_6)$
Layer 8 (Pooling)	:	$L_8 \equiv MaxPooling(L_7, \text{Size: } 2\times2, \text{Stride} = 2)$
Layer 9 (Affine)	:	$L_9 \equiv W_4 L_8 + B_4$
Layer 10 (Affine)	:	$L_{10} \equiv W_5 L_9 + B_5$
Layer 11 (Softmax)	:	$L_{11} \equiv SoftMax(L_{10})$

3.2.5. Model 5

Architecture

Model 5 is the most deep network in this study. Model 5 consists of 4 convolutional layers and 2 affine (fully connected) layers. It takes 68×68 RGB images as input. 1st, 2nd, 3rd and 4th convolution layers contain 32, 48, 64 and 96 numbers of receptive fields (kernels) respectively. The kernel sizes in the convolutional layers in model 5are7×7, 5×5, 3×3and 3×3 respectively.

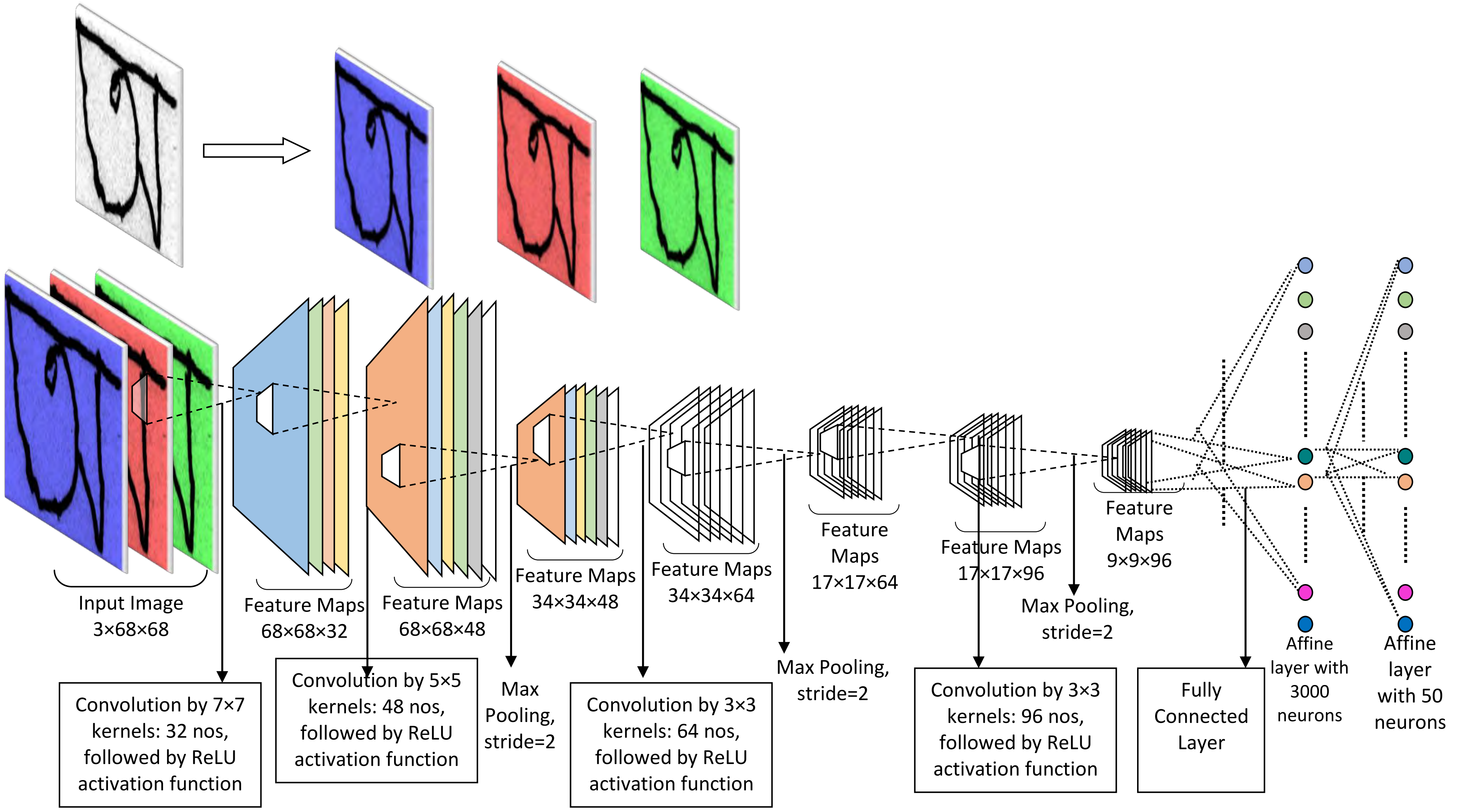
After 1st convolution layer ReLU is used as activation function, but no sub-sampling layer is used. So, after the 1st convolution between 68×68 input image size for each channel (RGB) and 32 nos of 7×7 kernels for each channel, the size of the feature maps become 68×68×32. Padding 3 and stride 1 are used for the convolution operation. Application of ReLU activation does not change the number of parameters. Pooling is not used in the first layer.

After each of 2nd convolution layer, ReLU function and MaxPooling (sub-sampling) with stride 2 are used in Model 5. Padding 2 and stride 1 are used for the convolution operation. Both pooling height and width are 2. So, after the 2nd convolution between 68×68×32 feature map size and 48nos of 5×5 kernels, the size of the feature maps becomes 68×68×48. After first pooling operation, the feature map size reduced to 34×34×46.

After each of 3rd and 4th convolution layers, ReLU function and MaxPooling (sub-sampling) with stride 2 are used in Model 5. After 3rdconvolution layer with 64nos of kernel, the feature size becomes 34×34×64. After second pooling with2×2 size and 2 stride, the feature map size reduced to 17×17×64. After 4th convolution layer with 96 nos of kernel, the feature size becomes 17×17×96. After third pooling with 2×2 size and 2 stride, the feature map size reduced to 9×9×96.And at the end two fully-connected layers with 3000 and 50 neurons respectively are used as classifier.

Appendix

Figure



Input Image 3×68×68
Feature Maps 68×68×32
Feature Maps 68×68×48
Feature Maps 34×34×48
Feature Maps 34×34×64
Feature Maps 17×17×64
Feature Maps 17×17×96
Feature Maps 9×9×96
Convolution by 7×7 kernels: 32 nos, followed by ReLU activation function
Convolution by 5×5 kernels: 48 nos, followed by ReLU activation function
Max Pooling, stride=2
Convolution by 3×3 kernels: 64 nos, followed by ReLU activation function
Max Pooling, stride=2
Convolution by 3×3 kernels: 96 nos, followed by ReLU activation function
Max Pooling, stride=2
Fully Connected Layer
Affine layer with 3000 neurons
Affine layer with 50 neurons

Appendix

Function

Let, the input image be X (3×32×32),

Layer 1 (Conv)	:	$L_1 \equiv W_1 * X + B_1$
Layer 2 (ReLU)	:	$L_2 \equiv \max(0, L_1)$
Layer 3 (Conv)	:	$L_3 \equiv W_2 * L_2 + B_2$
Layer 4 (ReLU)	:	$L_4 \equiv max(0, L_3)$
Layer 5 (Pooling)	:	$L_5 \equiv MaxPooling(L_4, \text{Size:} 2\times2, \text{Stride} = 2)$
Layer 6 (Conv)	:	$L_6 \equiv W_3 * L_5 + B_3$
Layer 7 (ReLU)	:	$L_7 \equiv max(0, L_6)$
Layer 8 (Pooling)	:	$L_8 \equiv MaxPooling(L_7, \text{Size: } 2\times2, \text{Stride} = 2)$
Layer 9 (Conv)	:	$L_9 \equiv W_4 * L_8 + B_4$
Layer 10 (ReLU)	:	$L_{10} \equiv max(0, L_9)$
Layer 11 (Pooling)	:	$L_{11} \equiv MaxPooling(L_{10}, \text{Size: } 2\times2, \text{Stride} = 2)$
Layer 12 (Affine)	:	$L_{12} \equiv W_5 L_{11} + B_5$
Layer 13 (Affine)	:	$L_{13} \equiv W_6 L_{12} + B_6$
Layer 14 (Softmax)	:	$L_{14} \equiv \text{SoftMax}(L_{13})$

Table 3.2: Comparison between deep-CNN models

	Database	Network Architecture				Input Size	Training Parameters					Validation Accuracy rate
		Conv. Layers	**Kernel Size**	**Activation function & Pool size, Stride**	**Affine Layers**		**Regularization factor**	**Learning Rate**	**Learning rate decay**	**Batch size**	**No. of Epochs**	
Model 1	CMATERdb 3.1.2	3 nos. 1st Layer: 32 Kernels	1st Layer: Kernel size 3×3	1st Layer: Conv-ReLU	1 layer: 50 neurons	32×32	0.001	0.0001	0.95	50	50	86.79%
		2nd Layer: 64 Kernels	2nd Layer: Kernel size: 3×3	2nd Layer: Conv-ReLU-Pool, 2×2, 2								
		3rd Layer: 128 Kernels	3rd Layer: Kernel size: 3×3	3rd Layer: Conv-ReLU-Pool, 2×2, 2								
Model 2	CMATERdb 3.1.2	3 nos. 1st Layer: 32 Kernels	1st Layer: Kernel size 5×5	1st Layer: Conv-ReLU	1 layer: 50 neurons	32×32	0.001	0.0001	0.95	50	75	88.42%
		2nd Layer: 64 Kernels	2nd Layer: Kernel size: 3×3	2nd Layer: Conv-ReLU-Pool, 2×2, 2								
		3rd Layer: 128 Kernels	3rd Layer: Kernel size: 3×3	3rd Layer: Conv-ReLU-Pool, 2×2, 2								
Model 3	Combined Dataset	3 nos. 1st Layer: 32 Kernels	1st Layer: Kernel size 5×5	1st Layer: Conv-ReLU	2 layers 1st Layer: 3000	32×32	0.001	0.0001	0.95	100	50	89.96%
		2nd Layer: 64 Kernels	2nd Layer: Kernel size: 3×3	2nd Layer: Conv-ReLU-Pool, 2×2, 2	2nd Layer: 50							
		3rd Layer: 128 Kernels	3rd Layer: Kernel size: 3×3	3rd Layer: Conv-ReLU-Pool, 2×2, 2								
Model 3	Combined Dataset	3 nos. 1st Layer: 32 Kernels	1st Layer: Kernel size 5×5	1st Layer: Conv-ReLU	2 layers 1st Layer: 3000	32×32	0.001	0.0001	0.95	100	150	92.19%
		2nd Layer: 64 Kernels	2nd Layer: Kernel size: 3×3	2nd Layer: Conv-ReLU-Pool, 2×2, 2	2nd Layer: 50							
		3rd Layer:	3rd Layer:	3rd Layer:								

	Database	Network Architecture				Input Size	Training Parameters					Validation Accuracy rate
		Conv. Layers	**Kernel Size**	**Activation function & Pool size, Stride**	**Affine Layers**		**Regularization factor**	**Learning Rate**	**Learning rate decay**	**Batch size**	**No. of Epochs**	
		128 Kernels	Kernel size: 3×3	Conv-ReLU-Pool, 2×2, 2								
Model 4 (proposed model)	Combined Dataset	3 nos. 1st Layer: 32 Kernels	1st Layer: Kernel size 5×5	1st Layer: Conv-ReLU	2 layers 1st Layer: 3500	32×32	0.001	0.0001	0.95	100	150	92.20%
		2nd Layer: 64 Kernels	2nd Layer: Kernel size: 3×3	2nd Layer: Conv-ReLU-Pool, 2×2, 2	2nd Layer: 50							
		3rd Layer: 128 Kernels	3rd Layer: Kernel size: 3×3	3rd Layer: Conv-ReLU-Pool, 2×2, 2								
Model 5	Combined Dataset	4 nos. 1st Layer: 32 Kernels	1st Layer: Kernel size 7×7	1st Layer: Conv-ReLU	2 layers 1st Layer: 3000	68×68	0.001	0.0001	0.95	100	150	85.26%
		2nd Layer: 48 Kernels	2nd Layer: Kernel size: 5×5	2nd Layer: Conv-ReLU-Pool, 2×2, 2								
		3rd Layer: 64 Kernels	3rd Layer: Kernel size: 3×3	3rd Layer: Conv-ReLU-Pool, 2×2, 2	2nd Layer: 50							
		4th Layer: 96 Kernels	4th Layer: Kernel size: 3×3	4th Layer: Conv-ReLU-Pool, 2×2, 2								

3.3. Conclusion

In this chapter a brief description of proposed DCNN model are presented. We can see that there is a significant change in accuracy level for different model architectures. So, number of kernels in different convolution layers, sizes of the kernels, depth of the network and number of neurons in the classifier layers have their effects on the performance of the recognizer. Among the different models used in this experiment, the best DCNN architecture is model 4, which has 3 convolutional layers and 2 affine (fully connected) layers. 1^{st}, 2^{nd} and 3^{rd} convolution layers contain 32, 64 and 128 numbers of receptive fields (kernels) respectively. The kernel size in the first convolutional layer in model 4 is 5×5, the kernel sizes in the 2^{nd} and 3^{rd} convolutional layers are of equal size: 3×3.

In the next chapter, description of the database and the experimental platform will be presented. After that, the characteristics of the learning process and performance of the proposed model with respect to the other techniques of BHCR will be analyzed.

Chapter 4

Experimental Results

4.1. Introduction

This chapter describes the database and experimental results of this study. At first the problem is defined. After that experimental platform in terms of hardware and software are mentioned. Then the database information, source and sample data are presented. At the end of the chapter, the performance of the proposed models are analyzed and evaluated.

4.2. Experimental Platform

The experiment has been conducted on desktop machine (CPU: Intel Core i7-6700K @ 4 GHz, RAM: 16.00 GB, GPU: GeForce GTX 970, Hard Disk Drive: Transcend 128 GB Solid State Drive) in Ubuntu 16.04LTS 64-bit OS (Linux) environment.

The algorithm ran on Anaconda 4.2.0 64-bit platform with Jupyter Notebook version 4.2.3. The DCNN algorithm is implemented in Python 2.7.12. List of major library and packages used in the implementation of the algorithm are given in table 4.1.

Table 4.1: Major library and packages used to implement the algorithm

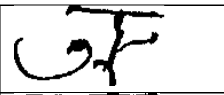
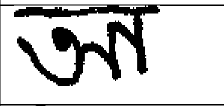
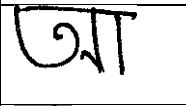
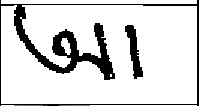
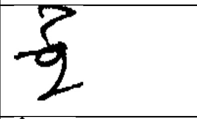
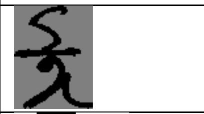
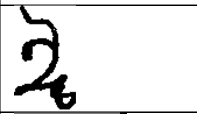
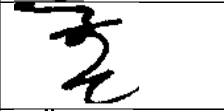
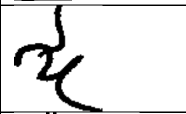
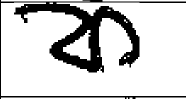
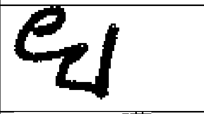
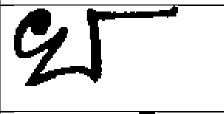
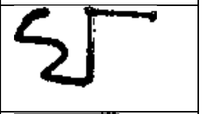
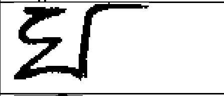
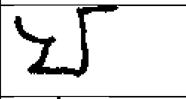
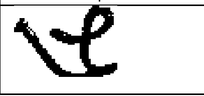
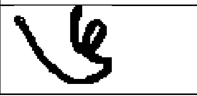
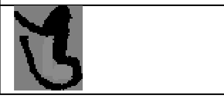
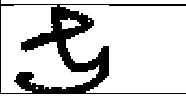
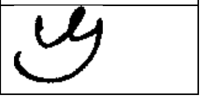
Package/ Library Name	Version number
numpy	1.11.1
nose	1.3.7
cython	0.24.1
matplotlib	1.5.3
pandas	0.18.1
scipy	0.18.1
six	1.10.0
sympy	1.0

4.3. Database

4.3.1. Database CMATERdb 3.1.2

There are two databases used in this experiment. One is CMATERdb 3.1.2 [56] containing 12000 train and 3000 test samples equally distributed among 50 classes of hand-written Bangla characters. CMATERdb is the pattern recognition database repository created at the 'Center for Microprocessor Applications for Training Education and Research' (CMATER) research laboratory, Jadavpur University, Kolkata 700032, India. Sample images are given in table 4.2.

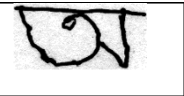
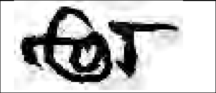
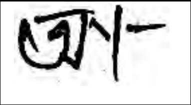
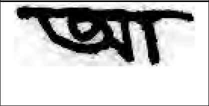
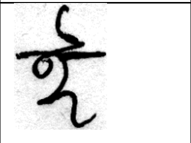
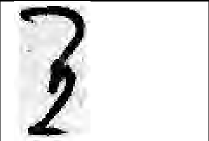
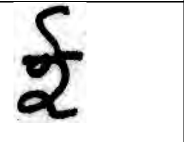
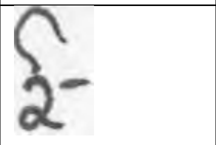
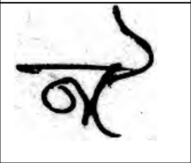
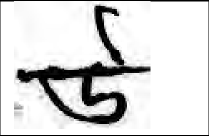
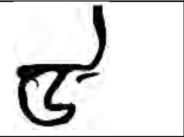
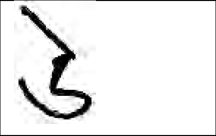
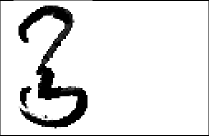
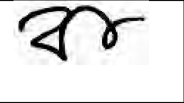
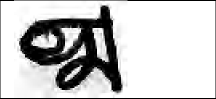
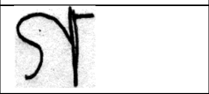
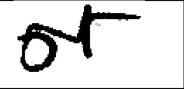
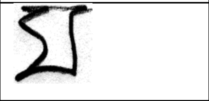
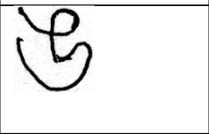
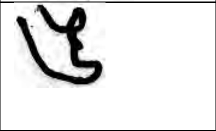
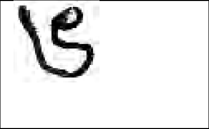
Table 4.2: Sample images of CMATERdb 3.1.2 database. More sample images are given in Appendix A.

Character	Sample Images				
অ					
আ					
ই					
ঈ					
ক					
খ					
গ					
ঘ					
ঙ					

4.3.2. Database BBCD

Another database is referred to as the "Bangla Basic Character Database (BBCD)" [54], the database of 37,858 samples were randomly subdivided into training and test sets. Samples of this database were collected using three different types of form documents, viz., railway reservation form, job application form, and a tabular form specially designed for data collection. Handwritten samples of various basic characters collected from the name and address parts of the first two types of forms vary widely in number with only a few samples for rarely occurring Bangla basic characters. Some sample images are given in table 4.3.

Table 4.3: Sample images of BBCD database. More sample images are given in Appendix A.

Character	Sample Images				
অ					
আ					
ই					
ঈ					
উ					
ক					
খ					
গ					
ঘ					
ঙ					

4.3.3. Combined Database

The both datasets (Database CMATERdb 3.1.2 and BBCD) are combined to form larger dataset containing a total of 52,788 samples subdivided into 28,529 (54.04%) training images, 8,400 (15.91%) validation samples and 15,859 (30.04%) test samples of similar sizes. The dataset contains wide variation of distinct characters because of different peoples' writing styles. Some of these character images are very complex shaped and closely correlated with others. This is the largest dataset among all reported BHCR works.

4.4. Training of the DCNN

There is no significant preprocessing of the input database. Since the input images are of different sizes, hence to feed the images as the inputs of the DCNN, all the input images are resized into 32×32 images. The images of letters are black in white background, so to reduce computational overhead, images are converted through foreground character black to white and background changed to black. The input images are considered as RGB images containing 3 channels and 8 bit depth per pixel. The images are then normalized to get a zero mean over the complete dataset. For the training of DCNN following factors are used:

• Regularization factor	: 0.001
• Learning rate	: 0.0001
• Learning rate decay factor	: 0.95
• Batch size	: 100
• No of epochs	: 150
• Back-propagation method	: RMS propagation with SGD and decay rate = 0.99
• Cost Function	: SoftMax Loss function.

All weights and bias parameters are initialized randomly using zero mean and unit variance gaussian distribution.

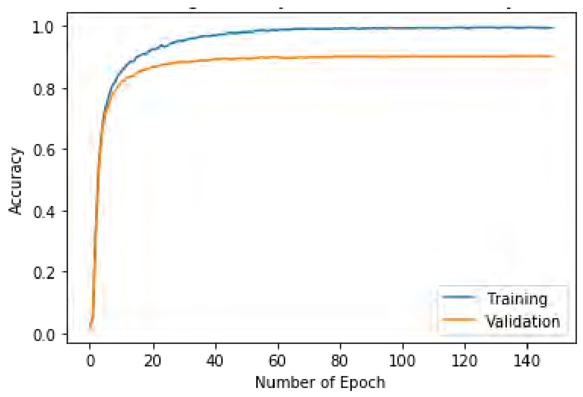


Figure 4.1: Training and Validation accuracy curves versus number of Epoch

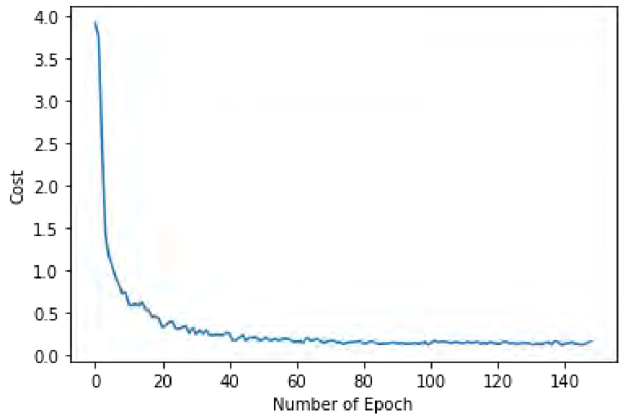


Figure 4.2: Cost function versus number of Epoch

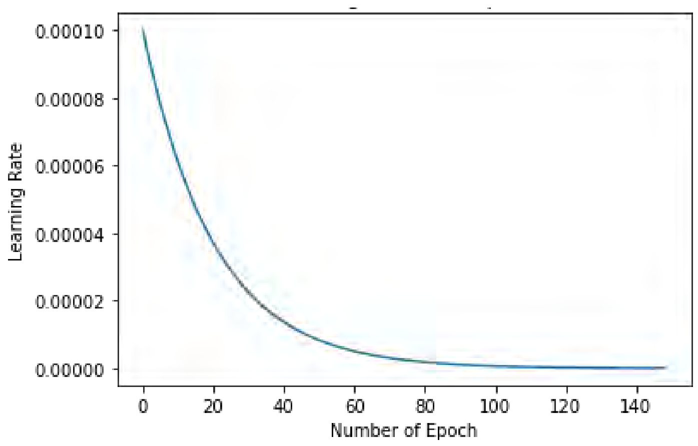


Figure 4.3: Learning rate versus number of Epoch

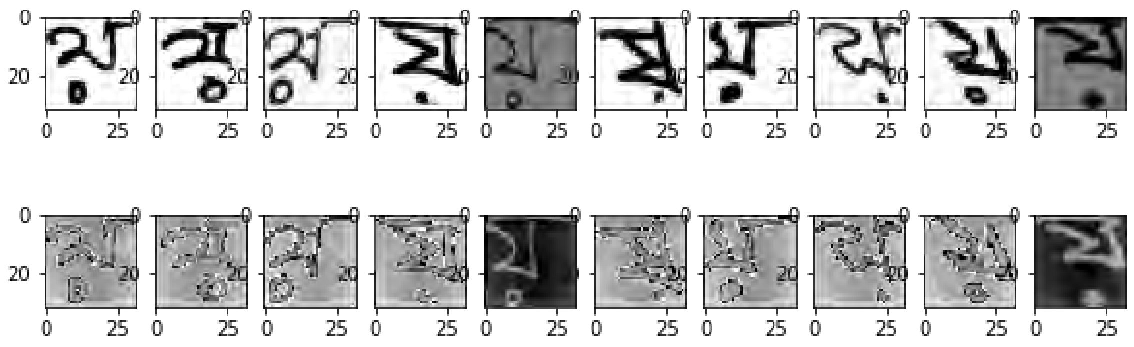


Figure 4.4: Input images in database (above) and the same images after normalization (below)

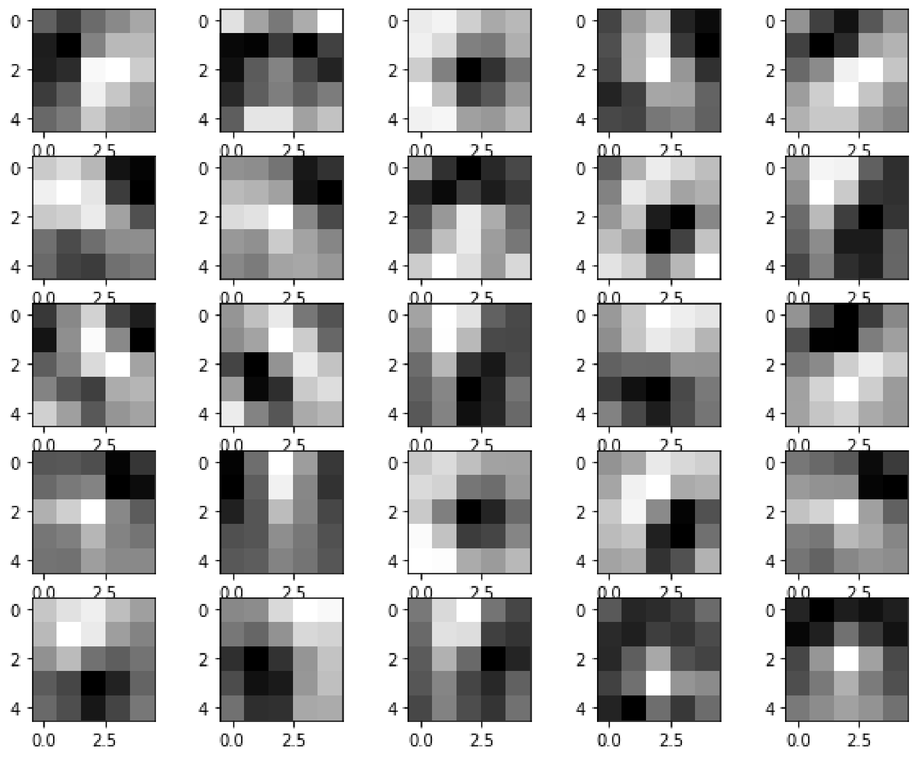


Figure 4.5 : Sample Kernels of the first convolution layer

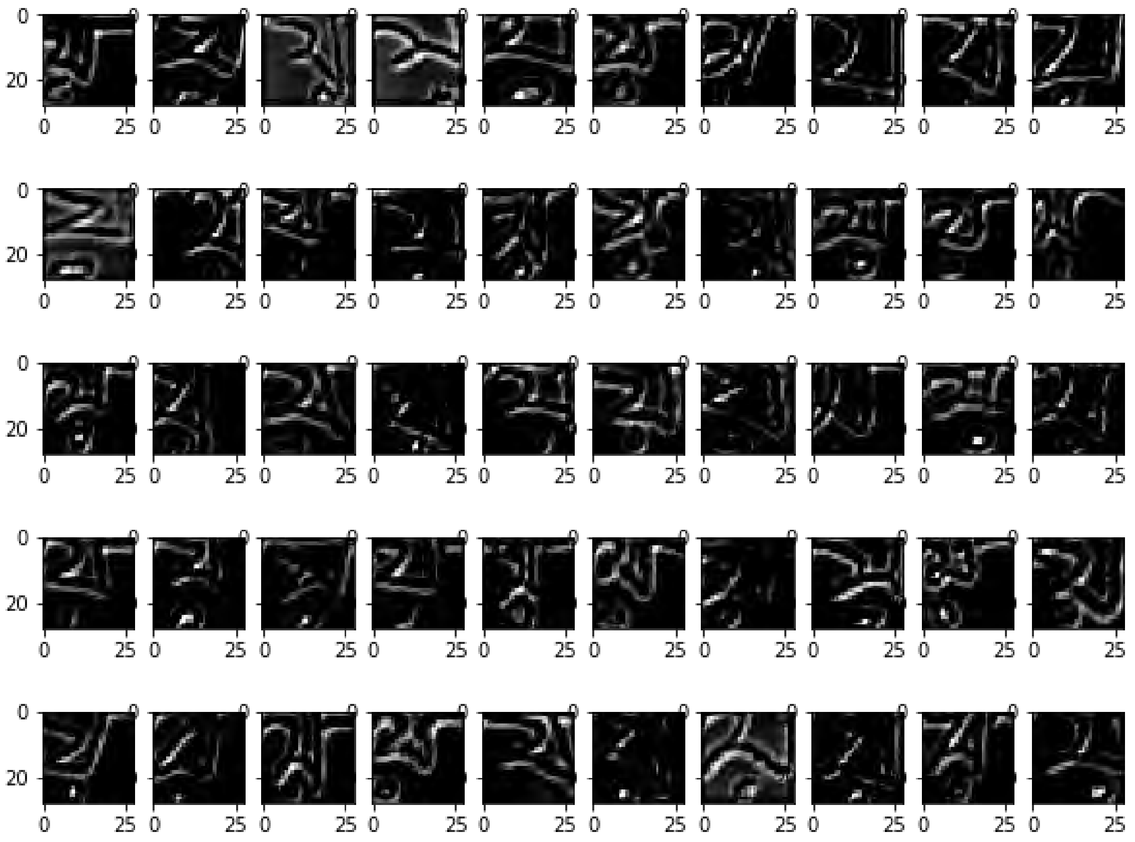


Figure 4.6: Feature Maps after the first convolution layer

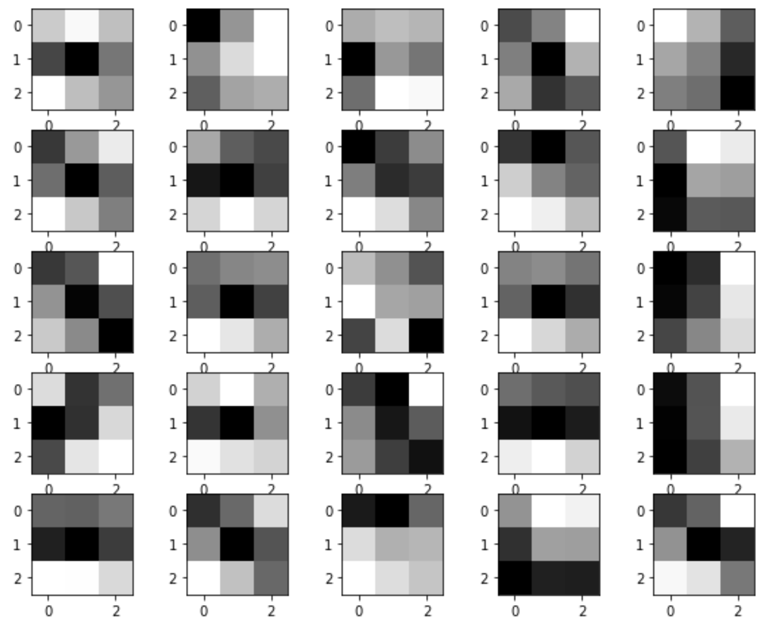


Figure 4.7 : Sample Kernels of the second convolution layer

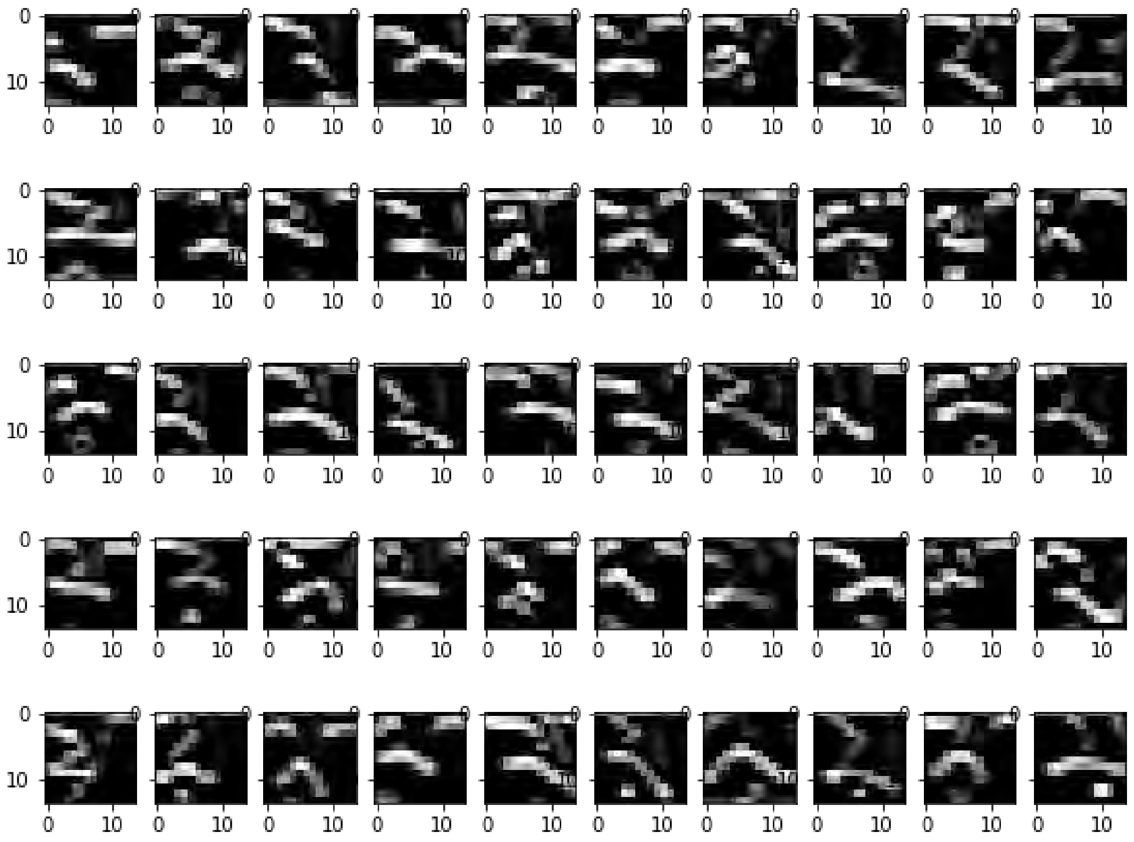


Figure 4.8: Feature Maps after the second convolution layer

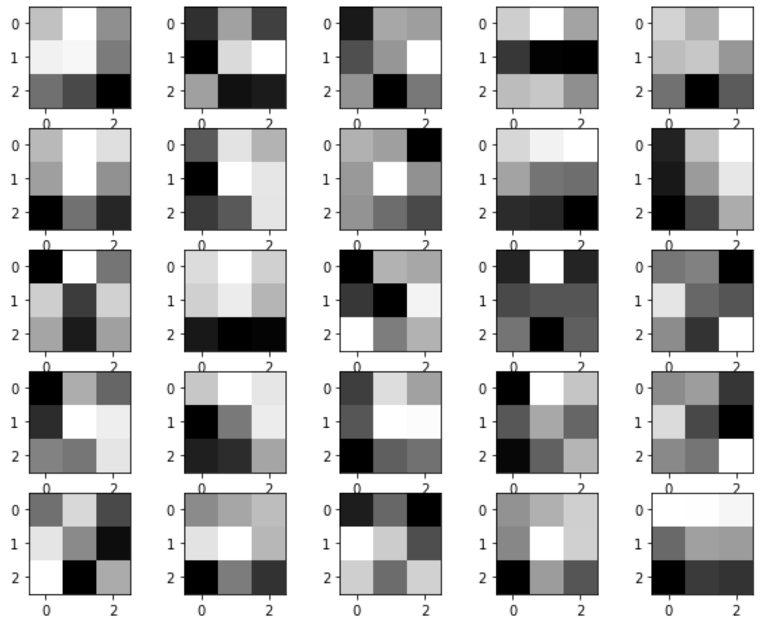


Figure 4.9: Sample kernels of 3rd convolution layer

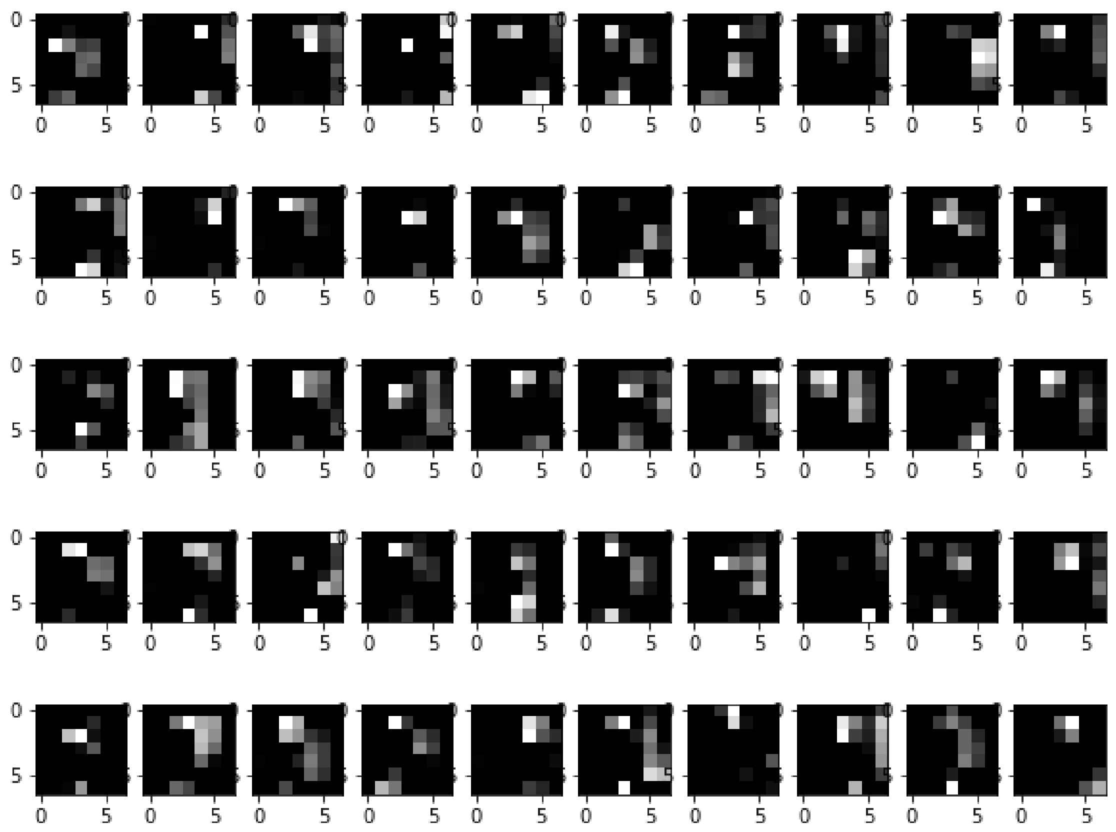


Figure 4.10: Feature Maps after 3rd convolution layer

4.5. Performance Evaluation

After 150 epochs of training, the accuracy of the DCNN for BHCR is presented in table 4.4. After 150 epochs proposed DCNN achieves 99.43% recognition accuracy on training dataset, 92.10% recognition accuracy on validation dataset and 91.25% recognition accuracy on test dataset. The confusion matrix of the test samples is given in Table 4.5. From the table number of samples and recognition accuracy for each class can be seen. From the table, it can be seen that the proposed method performs worst to recognize the character “খ (KHA)”. Among 240 samples, it truly recognizes 187 cases (77.92%). In 26 cases (10.83%) the character has been classified as “ঘ (GHA)” and in 7 cases (2.92%) it has been classified as “থ (THA)” that looks similar even printed form and more difficult in handwritten form.

Table 4.4: Accuracy of the DCNN for BHCR

No of Epoch	Training Accuracy	Validation Accuracy	Test Accuracy
150	99.43%	92.10%	91.25%

Similarly among 316 samples of "ঘ (GHA)" the model truly recognizes 254 cases (80.38%) and in 33 cases (10.44%) it is classified as "খ (KHA)", in 7 cases (2.22%) it is classified as "ম (MA)" and in 6 cases (1.90%) it is classified as "য (YY)". The proposed method has shown best performance for "s (ANUS)". Among 157 samples of "s (ANUS)" the model truly recognizes 156 cases (99.36%) and in 1 case (0.64%) it is classified as "V (TTHA)". Due to large variation in writing styles, such character images are difficult to classify even by human. Finally, the proposed DCNN misclassifies 1,388 cases out of 15,859 test cases and achieves accuracy 91.25% on test dataset.

Table 4.6 and 4.7 present the confusion matrix of the training and validation datasets respectively. It shows 99.43% recognition accuracy on training dataset of 28,529 samples and 92.10% recognition accuracy on validation dataset of 8,400 samples.

Table 4.5: Confusion Matrix produced for test dataset (15,859 samples) from DCNN of BHCR

O U T P U T   C L A S S E S - - - - - - >

T A R G E T   C L A S S E S   I I I I I I V

		অ	আ	ই	ঈ	উ	ঊ	ঋ	এ	ঐ	ও	ঔ	ক	খ	গ	ঘ	ঙ	চ	ছ	জ	ঝ	ঞ	ট	ঠ	ড	ঢ	ণ	ত	থ	দ	ধ	ন	প	ফ	ব	ভ	ম	য	র	ল	শ	ষ	স	হ	ড়	ঢ়	য়	ৎ	ং	ঃ	ঁ	No of Samples per Class	Recognition Rate
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50		
অ	1	392	27	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	3	4	2	0	0	0	0	0	0	0	0	2	0	0	3	1	0	0	0	0	0	0	1	0	3	0	0	0	0	0	0	0	0	440	89.09%
আ	2	8	413	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	3	0	0	0	0	0	0	0	0	0	0	429	96.27%
ই	3	0	0	120	3	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	128	93.75%
ঈ	4	0	0	2	138	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	146	94.52%
উ	5	0	0	6	1	404	15	0	0	0	5	0	0	0	0	0	1	0	0	1	0	0	7	8	1	2	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	453	89.18%
ঊ	6	0	0	0	1	5	132	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	140	94.29%
ঋ	7	1	0	0	0	0	0	242	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	249	97.19%
এ	8	0	0	0	0	0	0	2	313	1	3	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	321	97.51%
ঐ	9	0	0	0	0	0	0	2	0	188	0	0	0	0	0	1	2	0	0	1	0	0	0	0	0	0	0	3	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	199	94.47%
ও	10	0	0	3	0	0	1	0	3	1	128	0	0	1	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	2	0	0	0	0	0	142	90.14%
ঔ	11	0	1	0	0	0	0	0	0	0	0	114	0	5	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	123	92.68%
ক	12	0	1	0	1	0	0	0	0	0	0	1	801	0	1	1	0	0	1	2	24	0	0	3	0	0	4	0	1	1	1	0	2	20	5	0	2	0	0	0	9	2	1	0	0	0	0	0	0	0	0	884	90.61%
খ	13	0	1	0	0	0	0	0	0	0	0	3	1	187	1	26	0	0	0	0	0	0	0	0	0	0	0	0	7	0	1	0	2	3	0	2	1	0	0	1	1	2	1	0	0	0	0	0	0	0	0	240	77.92%
গ	14	4	0	0	0	0	0	0	0	0	0	0	0	1	241	3	0	1	1	3	0	0	1	0	0	0	11	0	0	3	0	0	4	0	0	0	1	0	0	0	1	0	7	0	0	0	0	0	0	0	0	282	85.46%
ঘ	15	0	0	0	0	0	0	0	1	0	0	0	0	33	0	254	0	0	0	1	0	0	0	0	0	0	0	0	1	0	1	0	0	3	0	0	7	6	0	0	1	3	3	1	1	0	0	0	0	0	0	316	80.38%
ঙ	16	0	0	0	0	4	0	0	0	0	2	0	1	0	0	0	182	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	198	91.92%
চ	17	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	188	0	0	0	0	1	0	1	1	1	0	0	0	0	0	0	0	2	1	0	0	0	0	0	2	0	0	0	1	0	0	0	0	0	199	94.47%
ছ	18	0	0	3	0	0	0	0	0	0	0	0	0	0	0	1	0	0	181	1	0	0	0	0	0	1	0	0	0	2	1	0	1	0	0	0	1	0	0	0	0	0	0	5	1	1	0	0	0	0	0	199	90.95%
জ	19	3	1	0	1	0	0	0	0	0	0	0	0	0	1	0	0	1	6	238	0	1	0	0	0	0	0	0	0	1	1	0	0	0	1	0	1	0	0	0	0	0	1	1	0	0	0	0	0	0	1	259	91.89%
ঝ	20	0	0	0	0	0	0	0	0	0	0	5	1	0	0	0	0	0	0	0	203	0	0	0	0	0	0	0	1	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	213	95.31%
ঞ	21	0	0	0	0	0	0	0	1	0	2	0	2	0	0	0	2	0	0	1	0	201	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	209	96.17%
ট	22	0	0	0	1	4	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	175	1	0	4	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	190	92.11%
ঠ	23	0	0	2	0	2	0	1	0	0	0	0	0	0	0	0	0	3	1	0	0	0	2	127	0	0	0	1	0	0	5	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	146	86.99%
ড	24	0	0	0	0	2	0	0	0	5	0	0	0	0	0	1	1	0	0	0	0	0	0	0	209	1	0	0	0	1	0	0	0	0	0	8	0	0	0	0	0	1	0	0	0	0	0	0	0	2	0	231	90.48%
ঢ	25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	4	0	0	0	0	1	0	0	120	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	126	95.24%
ণ	26	0	0	0	0	0	0	0	0	0	0	0	0	0	5	0	0	1	0	0	0	0	0	0	0	0	166	0	0	0	0	24	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	198	83.84%
ত	27	0	0	0	0	1	0	0	0	3	0	0	0	0	0	0	2	0	0	0	0	0	0	0	2	2	0	431	0	0	0	0	0	0	0	9	0	0	0	0	0	0	0	0	0	0	0	0	0	3	2	455	94.73%
থ	28	0	0	0	0	0	0	0	0	0	0	0	1	9	0	0	0	0	0	0	0	0	0	0	0	1	0	0	153	0	2	0	1	1	0	0	0	7	0	1	1	0	0	2	0	0	0	0	0	0	0	179	85.47%
দ	29	0	0	0	1	0	0	2	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	1	0	2	1	0	257	0	9	1	0	1	0	0	0	2	1	0	0	1	1	0	0	1	0	0	0	0	283	90.81%
ধ	30	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	4	0	0	0	0	0	0	118	0	1	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	129	91.47%
ন	31	1	0	0	0	1	0	2	1	0	0	0	1	0	10	1	0	7	1	0	0	0	1	2	0	3	55	0	0	20	0	690	1	2	0	0	11	2	1	10	1	0	4	0	0	2	0	0	0	0	0	830	83.13%
প	32	1	0	0	0	0	0	2	0	0	0	0	2	0	7	0	0	0	0	2	0	0	1	0	0	0	13	0	3	4	1	1	570	2	2	0	1	1	0	4	8	1	17	0	0	0	1	0	0	0	1	645	88.37%
ফ	33	0	0	0	0	0	0	0	0	0	0	1	3	0	1	0	0	0	0	0	1	0	1	1	0	0	0	0	1	0	0	0	0	190	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	200	95.00%
ব	34	0	0	0	0	0	0	1	1	0	0	0	10	0	0	0	0	8	0	0	4	0	1	1	0	0	2	0	0	0	5	0	1	0	768	0	0	10	6	0	0	5	1	0	0	0	0	0	0	0	0	824	93.20%
ভ	35	0	0	0	0	2	0	1	0	1	1	0	0	0	1	0	5	0	0	1	0	0	4	4	13	3	0	4	0	0	0	0	0	0	0	257	0	0	0	0	0	0	0	1	0	0	0	0	0	3	1	302	85.10%
ম	36	0	0	0	0	0	0	0	0	0	0	0	0	2	0	1	0	1	0	0	1	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	232	4	0	0	2	0	2	1	0	0	0	0	0	0	0	248	93.55%
য	37	0	0	0	0	0	0	0	0	0	0	0	0	1	0	4	0	0	0	0	0	0	1	2	0	1	0	1	13	0	0	0	1	1	0	0	1	152	0	0	0	2	0	0	0	0	2	0	0	0	0	182	83.52%
র	38	0	0	0	0	0	0	1	2	2	0	0	1	0	3	0	0	0	1	2	0	0	3	0	0	0	1	2	0	1	1	0	0	0	4	0	12	0	1037	0	0	2	4	2	0	1	12	0	0	1	3	1098	94.44%
ল	39	1	0	0	0	0	0	0	0	0	0	1	1	0	3	0	0	0	0	0	0	0	0	0	0	0	5	0	0	0	0	27	4	4	0	0	0	1	0	761	7	1	5	0	0	0	0	0	0	0	0	821	92.69%
শ	40	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	3	0	0	0	2	0	0	1	185	0	0	0	0	0	0	0	0	0	0	194	95.36%
ষ	41	0	0	0	1	0	0	0	0	0	0	0	0	3	0	1	1	0	1	0	0	3	0	1	0	0	0	0	0	1	0	0	2	1	0	0	2	3	0	0	2	272	0	0	0	0	1	0	0	0	0	295	92.20%
স	42	2	0	0	1	0	0	0	0	0	0	0	0	3	3	5	0	0	0	3	0	0	0	1	0	0	0	0	1	2	0	1	5	0	0	0	6	5	0	0	1	0	254	0	0	0	0	0	0	0	0	293	86.69%
হ	43	1	0	14	2	0	0	3	3	0	1	0	0	0	0	0	0	1	10	2	0	0	0	0	1	0	0	0	0	0	7	0	0	0	0	0	2	1	0	0	0	0	0	321	0	0	0	1	0	0	6	376	85.37%
ড়	44	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	242	2	1	1	0	0	2	252	96.03%
ঢ়	45	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	256	1	0	0	0	0	263	97.34%
য়	46	0	0	0	0	0	0	0	0	1	0	0	0	1	1	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	3	4	2	0	4	0	4	0	1	5	482	0	0	0	0	510	94.51%
ৎ	47	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	207	3	2	4	221	93.67%
ং	48	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	156	0	0	157	99.36%
ঃ	49	1	0	0	0	0	0	1	0	1	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	178	0	188	94.68%
ঁ	50	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	2	0	0	0	0	0	0	2	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	245	254	96.46%
		415	444	152	151	427	149	262	325	205	144	126	827	248	279	301	203	215	214	266	237	205	204	160	229	141	263	450	183	296	149	752	603	232	785	286	286	197	1049	782	228	297	310	336	245	271	501	213	160	189	267	15859	91.25%

Table 4.6: Confusion Matrix produced for training dataset (28,529 samples) from DCNN of BHCR

O U T P U T   C L A S S E S - - - - - - - >

T A R G E T   C L A S S E S

		অ	আ	ই	ঈ	উ	ঊ	ঋ	এ	ঐ	ও	ঔ	ক	খ	গ	ঘ	ঙ	চ	ছ	জ	ঝ	ঞ	ট	ঠ	ড	ঢ	ণ	ত	থ	দ	ধ	ন	প	ফ	ব	ভ	ম	য	র	ল	শ	ষ	স	হ	ড়	ঢ়	য়	ৎ	ং	ঃ	ঁ	No of Samples per Class	Recognition Rate
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50		
অ	1	496	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	501	99.00%
আ	2	11	559	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	572	97.73%
ই	3	0	0	568	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	572	99.30%
ঈ	4	0	0	2	569	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	572	99.48%
উ	5	0	0	0	0	568	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	572	99.30%
ঊ	6	0	0	0	0	3	568	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	572	99.30%
ঋ	7	0	0	0	0	0	0	572	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	572	100.00%
এ	8	0	0	0	0	0	0	0	572	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	572	100.00%
ঐ	9	0	0	0	0	0	0	0	0	572	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	572	100.00%
ও	10	0	0	0	0	0	0	0	0	0	572	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	572	100.00%
ঔ	11	0	0	0	0	0	0	0	0	0	0	570	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	572	99.65%
ক	12	0	0	0	0	0	0	0	0	0	0	0	572	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	572	100.00%
খ	13	0	0	0	0	0	0	0	0	0	0	0	0	561	0	7	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	572	98.08%
গ	14	0	0	0	0	0	0	0	0	0	0	0	0	0	572	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	572	100.00%
ঘ	15	0	0	0	0	0	0	0	0	0	0	0	0	5	0	565	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	572	98.78%
ঙ	16	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	571	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	572	99.83%
চ	17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	572	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	572	100.00%
ছ	18	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	1	566	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	572	98.95%
জ	19	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	2	567	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	572	99.13%
ঝ	20	0	0	0	0	0	0	0	1	0	0	2	0	0	0	0	0	0	0	0	569	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	572	99.48%
ঞ	21	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	572	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	572	100.00%
ট	22	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	568	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	572	99.30%
ঠ	23	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	572	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	572	100.00%
ড	24	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	567	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	572	99.13%
ঢ	25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	569	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	572	99.48%
ণ	26	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	562	0	0	0	0	8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	572	98.25%
ত	27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	570	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	572	99.65%
থ	28	0	0	0	0	0	0	0	0	0	0	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	562	0	0	1	0	1	0	0	1	4	0	0	0	0	0	0	0	0	0	0	0	0	0	572	98.25%
দ	29	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	571	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	572	99.83%
ধ	30	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	571	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	572	99.83%
ন	31	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	1	0	0	0	12	0	0	0	0	557	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	572	97.38%
প	32	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	572	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	572	100.00%
ফ	33	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	569	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	572	99.48%
ব	34	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	572	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	572	100.00%
ভ	35	0	0	0	0	0	1	0	0	1	0	0	1	0	0	0	0	1	0	0	0	0	0	0	1	0	0	3	0	0	0	0	0	0	0	562	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	572	98.25%
ম	36	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	570	1	0	0	0	0	0	0	0	0	0	0	0	0	0	572	99.65%
য	37	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	8	0	0	1	0	2	0	0	1	558	0	0	0	0	0	0	0	0	0	0	0	0	0	572	97.55%
র	38	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	572	0	0	0	0	0	0	0	0	0	0	0	0	572	100.00%
ল	39	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	569	0	0	0	0	0	0	0	0	0	0	0	572	99.48%
শ	40	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	572	0	0	0	0	0	0	0	0	0	0	572	100.00%
ষ	41	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	570	0	0	0	0	0	0	0	0	0	572	99.65%
স	42	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	570	0	0	0	0	0	0	0	0	572	99.65%
হ	43	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	572	0	0	0	0	0	0	0	572	100.00%
ড়	44	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	569	0	0	0	0	0	0	572	99.48%
ঢ়	45	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	569	0	0	0	0	0	572	99.48%
য়	46	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	572	0	0	0	0	572	100.00%
ৎ	47	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	571	0	0	0	572	99.83%
ং	48	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	571	0	0	572	99.83%
ঃ	49	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	571	0	572	99.83%
ঁ	50	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	572	572	100.00%
		508	563	571	572	573	571	572	573	573	573	575	573	569	579	574	571	577	568	570	572	572	572	574	572	576	574	573	574	573	573	569	572	573	572	565	572	567	572	569	572	571	571	574	571	569	572	572	571	573	572	28529	99.43%

Table 4.7: Confusion Matrix produced for validation dataset (8,400 samples) from DCNN of BHCR

O U T P U T   C L A S S E S - - - - - - >

T A R G E T   C L A S S E S | | | | | | V

		অ	আ	ই	ঈ	উ	ঊ	ঋ	এ	ঐ	ও	ঔ	ক	খ	গ	ঘ	ঙ	চ	ছ	জ	ঝ	ঞ	ট	ঠ	ড	ঢ	ণ	ত	থ	দ	ধ	ন	প	ফ	ব	ভ	ম	য	র	ল	শ	ষ	স	হ	ড়	ঢ়	য়	ৎ	ং	ঃ	ঁ	No of Samples per Class	Recognition Rate
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50		
অ	1	143	20	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	168	85.12%
আ	2	1	164	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	168	97.62%
ই	3	0	0	158	0	0	3	0	1	0	1	0	0	0	0	0	0	0	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	168	94.05%
ঈ	4	0	0	5	160	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	168	95.24%
উ	5	0	0	0	0	146	7	0	0	0	3	0	0	0	0	0	1	0	0	1	0	0	3	4	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	168	86.90%
ঊ	6	0	0	1	2	11	150	0	1	0	1	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	168	89.29%
ঋ	7	0	0	0	0	0	0	160	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	4	0	0	0	1	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	168	95.24%
এ	8	0	0	2	0	1	0	0	162	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	168	96.43%
ঐ	9	0	0	0	0	0	0	0	0	159	2	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	168	94.64%
ও	10	0	0	3	1	2	0	0	1	2	155	0	0	0	0	0	1	0	0	0	0	0	2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	168	92.26%
ঔ	11	0	0	0	1	0	0	0	0	0	0	159	0	3	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	168	94.64%
ক	12	0	0	0	0	0	0	0	0	0	0	1	151	1	0	0	0	0	0	0	5	0	0	0	0	0	0	0	2	0	0	0	0	3	0	0	0	0	0	1	3	0	0	1	0	0	0	0	0	0	0	168	89.88%
খ	13	0	0	0	0	0	0	0	0	0	0	1	0	134	0	24	0	0	0	0	1	0	0	0	0	0	0	0	6	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	168	79.76%
গ	14	0	0	0	0	0	0	0	0	0	0	0	0	0	155	0	0	0	1	0	0	0	0	0	0	0	1	0	0	3	0	3	3	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	168	92.26%
ঘ	15	0	0	0	0	0	0	0	0	0	0	0	0	13	0	150	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	1	0	0	0	0	1	0	0	0	0	0	0	0	0	168	89.29%
ঙ	16	0	0	0	0	4	0	0	0	1	1	0	0	0	0	0	152	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	7	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	168	90.48%
চ	17	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	158	0	0	0	0	2	2	0	2	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	168	94.05%
ছ	18	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	155	3	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	5	0	1	0	1	0	0	0	168	92.26%
জ	19	0	2	0	0	0	1	1	0	0	0	0	0	1	0	0	1	0	0	154	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	2	1	2	0	0	0	0	0	1	168	91.67%
ঝ	20	0	0	0	1	0	0	0	0	0	0	3	1	0	0	0	0	0	0	0	155	0	0	0	0	0	0	0	1	0	0	0	1	1	0	0	1	0	0	0	1	2	1	0	0	0	0	0	0	0	0	168	92.26%
ঞ	21	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	166	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	168	98.81%
ট	22	0	0	0	2	3	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	155	2	0	2	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	168	92.26%
ঠ	23	0	0	2	0	4	0	0	0	1	0	0	0	0	0	0	1	0	1	0	0	0	6	146	0	0	0	1	0	1	3	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	168	86.90%
ড	24	0	0	0	0	2	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	153	0	0	2	0	0	0	0	0	0	0	7	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	168	91.07%
ঢ	25	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	3	0	0	0	0	1	0	0	161	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	168	95.83%
ণ	26	0	0	1	0	0	0	0	0	0	0	0	0	0	3	0	0	1	0	0	0	0	1	1	0	0	143	0	0	2	0	11	1	0	1	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	1	168	85.12%
ত	27	1	0	0	0	0	0	2	0	2	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	2	0	152	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	1	0	0	0	0	0	3	0	168	90.48%
থ	28	0	0	0	0	0	0	0	0	0	0	0	0	6	0	3	0	0	0	0	0	0	0	0	0	0	0	0	149	0	0	1	0	3	0	0	0	6	0	0	0	0	0	0	0	0	0	0	0	0	0	168	88.69%
দ	29	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	155	0	4	0	0	0	0	0	0	3	3	0	0	0	0	0	0	0	1	0	0	1	168	92.26%
ধ	30	0	0	0	0	0	0	0	0	0	0	0	1	3	0	0	0	0	1	0	0	0	1	2	0	0	0	0	0	0	156	0	1	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	168	92.86%
ন	31	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	3	0	1	0	0	0	0	0	0	7	0	0	3	0	150	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	168	89.29%
প	32	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	158	0	0	0	0	0	0	2	4	0	0	0	0	0	0	0	0	0	0	168	94.05%
ফ	33	0	0	0	0	0	0	0	0	0	0	1	5	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	7	0	0	1	1	148	0	0	0	2	0	1	1	0	0	0	0	0	0	0	0	0	0	168	88.10%
ব	34	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	161	0	0	1	2	0	0	0	0	0	0	0	0	0	0	0	0	168	95.83%
ভ	35	0	0	0	0	2	0	0	0	3	1	0	0	0	0	0	3	0	0	0	0	0	0	0	0	7	0	0	0	0	0	0	0	0	0	150	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	168	89.29%
ম	36	0	0	0	0	0	0	0	0	0	0	1	0	2	0	1	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	151	3	0	0	2	3	3	0	0	0	0	0	0	0	0	168	89.88%
য	37	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	19	0	0	1	1	0	1	0	2	141	0	0	1	0	0	0	0	0	0	0	0	0	0	168	83.93%
র	38	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	159	0	0	1	0	0	0	1	3	0	0	0	1	168	94.64%
ল	39	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	4	0	0	0	0	0	0	0	159	1	0	1	0	0	0	0	0	0	0	0	168	94.64%
শ	40	0	0	0	0	0	0	0	0	0	0	0	1	2	2	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	4	1	0	0	0	0	0	2	154	0	1	0	0	0	0	0	0	0	0	168	91.67%
ষ	41	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	3	0	0	0	0	0	0	0	0	161	0	0	0	0	0	0	0	0	0	168	95.83%
স	42	0	0	0	0	0	0	1	0	0	0	0	0	0	3	3	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	4	1	0	0	1	4	0	0	1	0	148	0	0	0	0	0	0	0	0	168	88.10%
হ	43	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	2	0	0	0	0	0	0	0	0	0	0	0	0	161	0	0	0	2	0	0	0	168	95.83%
ড়	44	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	159	3	3	0	0	1	1	168	94.64%
ঢ়	45	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	164	1	0	0	0	1	168	97.62%
য়	46	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	3	1	0	0	0	0	0	6	156	0	1	0	0	168	92.86%
ৎ	47	0	0	0	0	0	0	2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	155	2	3	3	168	92.26%
ং	48	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	165	0	0	168	98.21%
ঃ	49	0	0	0	0	0	0	2	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	160	0	168	95.24%
ঁ	50	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	1	2	0	160	168	95.24%
		146	186	173	169	175	161	169	168	173	167	168	163	167	167	186	166	167	165	161	166	168	173	162	163	170	154	163	189	168	165	177	180	159	165	170	158	158	168	172	174	170	161	173	163	176	163	163	170	169	173	8400	92.10%

Table 4.8: Experimental results showing comparison between proposed DCNN with some state-of-art methods of BHCR in terms of Accuracy and Variance on the same test Dataset of Combined Database and same experimental setup in terms of hardware and software.

Serial no	**Classification Methods**	**Test Accuracy**	**Variance**
1	kNN	64.878%	0.011354
2	Wavelet (Daubechies) based feature extraction [52] and then kNN classifier	65.439%	0.010489
3	Shallow CNN [55]	78.315%	0.003316
4	AlexNet [59] with last customized layer	80.04%	0.003234
5	DCNN (proposed method)	91.248%	0.001042

Experiments have been carried out on the combined dataset mentioned in article 4.4.3. Experimental results showing comparison between proposed DCNN with some state-of-art methods of BHCR in terms of test accuracy and variance on the same test Dataset of 15,859 samples of Combined Database are presented in table 4.8. The table shows that proposed DCNN method for BHCR outperforms other techniques in terms of both accuracy and variance. Moreover, since no feature extraction or significant preprocessing are needed, computational time required to get result for test dataset is very low compared to some other techniques of the table. It is to be noted that in proposed DCNN method, test accuracy (91.25%) is very close to the validation accuracy (92.10%) during training. It represents good generalization of learning of the network.

Table 4.9 represents a comparison of reported results of some prominent works with proposed DCNN on BHCR. Here, we can see that proposed method has been tested over the largest dataset to get result among the state-of-art methods.

In this experiment two separate databases are merged together to form a large dataset and many samples of this combined dataset are challenging to detect. It is notable that proposed method does not employ any feature selection technique whereas many existing methods use single or two stages feature selections. Though, the methods in Refs. [52] and [53] consider 45 and 36 classes respectively by merging or excluding some confusing character, still the table shows proposed method outperforms all other techniques except methods of Ref. [54].

The recognition techniques that uses Ref. [54] is much complex than others; it uses two recognition stages each one consists of individual feature selection and classification techniques. Besides this, the proposed method without feature selection is very simple. Also, in Ref. [54], significant preprocessing was done database used. As a result, once training is completed, proposed method recognizes the test samples very quickly compared to those which use computationally expensive feature selection stage. Moreover, the dataset used for training, validation and test in the work of Ref. [54] are a portion (database BBCD) of the combined database prepared for the experiment under this work.

Table 4.9: Comparison of reported test accuracies of some state-of-art methods with proposed DCNN on BHCR.

The work reference	Total Classes	Database	Size of test set	Feature Selection	Classification	Recog. Accuracy
Basu et al. [53]	36	-	-	Longest run, Modified Shadow, Octant-centroid	MLP	80.58 %
Bhowmick et al. [52]	45	Total: 27,000 samples, training samples: 18,000, Validation Samples: 4,500	4,500	Wavelet Transformation	MLP	84.33 %
Rahman et al. [33]	49	-	Not available	Multi-stage framework	Multiple Experts	88.38%
Bhattacharya et al. [43]	50	Total: 20,187 samples, training samples: 10,000	10,187	Chain code histogram feature	MLP classifier	88.95%
Bhattacharya et al. [35]	50	Total: 24,481 samples, training samples: 15,000.	9,481	Two-stage framework HMM	MLP classifier	90.42%
Bhattacharya et al. [54]	50	BBCD database containing 37,858 samples. Training samples: 20,000 and Validation Samples: 5000.	12,858	Regular and Irregular Grid based Selection	MQDF, MLP	95.84 %

The work reference	Total Classes	Database	Size of test set	Feature Selection	Classification	Recog. Accuracy
BHCR-CNN [55]	50	Prepared dataset of 20,000 samples. Training samples: 17,500	2,500	No	Shallow CNN	85.96 %
Proposed BHCR-DCNN	50	Combined dataset of CMATERdb 3.1.2 and BBCD. Total samples: 52,788. Training samples: 28,529. Validation samples: 8,400.	15,859	No	Deep CNN	91.25 %

4.6. Conclusion

The chapter touched several achievements of the proposed DCNN architecture by highlighting the results from different aspects. Different state-of-the-art performance metrics are used for evaluating its effectiveness. The proposed DCNN has been trained on the largest database among all reported works on BHCR so far. From all the results and illustrations, it is clearly seen that the proposed methodology has the capacity to outperform many of the existing BHCR recognition approaches for Bangla Characters.

Chapter 5

Conclusion

5.1. Summary of the work

Inspired by human visual cortex (visual cognition functions of human brain) CNN has the ability to recognize visual patterns directly from pixel images with minimal preprocessing. Therefore, in this thesis CNN structure is investigated without any feature selection for Bangla handwritten pattern classification. Proposed CNN structure has more depth compared to previous studies for Bangla Hand-written character recognition task. In this work, two large databases are merged together to form one larger database for the recognition task. The outcome has been compared with existing state-of-art methods for Bangla HCR. The proposed method has shown outstanding performance with respect to the exiting methods on the basis of generalized recognition capacity, test set accuracy and robustness in recognition. Since Bangla character set has 50 characters and many of them are similar and the CNN architecture proposed in this thesis is not dependent on specific features linked to character shapes of Bangla language, hence it has more generalized capacity of recognition and robustness in recognition task. So the proposed CNN architecture can also be used for HCR in other languages. Some other state-of-art techniques show good recognition accuracy but they use features that can be applicable to Bangla character set.

So, the proposed deep CNN architecture is efficient as well as robust in Bangla HCR.

5.2. Future Scope:

There are tremendous scopes of future extension of this work. Some of the scopes are listed out below:

- Multiple CNN channels (CNN ensemble) may be used to get majority based decision. Expected error from ensemble is always smaller than the expected error from a single predictor.

- Dropout layer may be introduced in the deep CNN model used in this work. Dropout is a regularization technique for reducing over-fitting in neural networks by preventing complex co-adaptations on training data.

- Inception module (i.e. different kernel sizes operating in parallel) may be introduced. The idea of the inception layer is to cover a bigger area, but also keep a fine resolution for small information on the images. The idea is that a series of gabor filters with different sizes, will handle better multiple objects scales. With the advantage that all filters on the inception layer are learnable. The most straightforward way to improve performance on deep learning is to use more layers and more data. Study shows that incorporating Inception module increases the accuracy rate. GoogleNet uses 9 Inception modules.

- Residual Network (ResNet) layers may be introduced by feeding the output of two successive convolutional layer AND also bypass the input to the next layers. The idea of the residual network is use blocks that re-route the input, and add to the concept learned from the previous layer. The idea is that during learning the next layer will learn the concepts of the previous layer plus the input of that previous layer. This would work better than just learn a concept without a reference that was used to learn that concept.

- Performance of proposed CNN could be analyzed for Bangla compound characters and digits.

References

[1] Suen, CY, Berthod, M. and Mori, S., *Automatic recognition of handprinted characters—the state of the art.,* Proc IEEE 68(4):469–487, 1980.

[2] Govindan, VK. and Shivaprasad, AP., *Character recognition: a review. Pattern Recognit*, 7:671–683, 1990.

[3] Trier, OD, Jain, AK. and Taxt, T., *Feature extraction methods for character recognition—a survey*,. Pattern Recognit 29(4):641–662, 1996.

[4] Plamondon, R. and Srihari, SN., *On-line and off-line handwriting recognition: a comprehensive survey*,. IEEE Trans Pattern Anal Mach Intell 22(1):63–84, 2000.

[5] Arica, N. and Yarman-Vural, F., *An overview of character recognition focused on off-line handwriting*,. IEEE Trans Syst Man Cybern Part C Appl Rev 31(2):216–232, 2001.

[6] Cheriet, M., Kharma, N., Liu, C-L. and Suen, CY., *Character recognition systems: a guide for students and practitioner*, Wiley, New York, 2007.

[7] Mori, S., Suen, CY. and Yamamoto, K., *Historical review of OCR research and development*, Proc IEEE 80(7):1029–1058, 1992.

[8] Uchida, S. and Sakoe, H., *A survey of elastic matching techniques for handwritten character recognition*, IEICE Transactions on Information and Systems E88-D(8): 1781–1790, 2005.

[9] Liu, C-L., Sako, H. and Fujisawa, H., *Performance evaluation of pattern classifiers for handwritten character recognition*, Int J Doc Anal Recognit 4(3):191–204, 2002.

[10] Park, H-S, Sin, B-K, Moon, J. and Lee, S-W, *A 2-D HMM method for offline handwritten character recognition*, Int J Pattern Recognit Artif Intell 15(1):91–105, 2001.

[11] Vinciarelli, A. and Bengio, S., *Writer adaptation techniques in HMM based off-line cursive script recognition*, Pattern Recognit Lett 23:905–916, 2002.

[12] Al-Omari, FA and Al-Jarrah, O., *Handwritten Indian numerals recognition system using probabilistic neural networks*, Adv Eng Inform 18(1): 9–16, 2004.

[13] Liu, C-L and Fujisawa, H., *Classification and learning methods for character recognition: advances and remaining problems*, Stud Comput Intell (SCI) 90:139–161, 2008.

[14] Kim, D. and Bang, S-Y, *A handwritten numeral character classification using tolerant rough set*, IEEE Trans Pattern Anal Mach Intell 22(9):923–937, 2000.

[15] Parizeau, M. and Plamondon, R., *A fuzzy-syntactic approach to allograph modeling for cursive script recognition*, IEEE Trans Pattern Anal Mach Intell 17:702–712, 1995.

[16] Hanmandlu, M., Ramana and Murthy, OV, *Fuzzy model based recognition of handwritten numerals*, Pattern Recognit 40(6):1840–1854, 2007.

[17] Dong, J-X, Krzyak, A. and Suen, CY, *An improved handwritten Chinese character recognition system using support vector machine*, Pattern Recognit Lett 26:1849–1856, 2007.

[18] Camastra, F., *SVM-based cursive character recognizer*, Pattern Recognit 40:3721–3727, 2007.

[19] LeCun, Y., Bottou, L., Bengio, Y. and Haffner, P., *Gradient-based learning applied to document recognition*, Proc IEEE 86(11): 2278–2324, 1998.

[20] Srihari, SN, Cohen, E., Hull, JJ. and Kuan, L., *A system to locate and recognize ZIP codes in handwritten addresses*, Int J Res Eng Post Appl 1(1):37–56, 1989.

[21] Marti, U-V and Bunke, H., *The IAM-database: an English sentence database for offline handwriting recognition*, Int J Doc Anal Recognit 5:39–46, 2002.

[22] Tang, Y., *Off-line recognition of Chinese handwriting by multifeature and multilevel classification*, IEEE Trans Pattern Anal Mach Intell 20:556–561, 1998.

[23] Shi, D., Damper, RI and GUNN, SR, *Offline handwritten Chinese character recognition by radical decomposition*, ACM Trans Asian Lang Inf Process 2(1):2748, 2003.

[24] Lee, SW and Park, JS, *Nonlinear shape normalization methods for the recognition of large-set handwritten characters*, Pattern Recognit 27(7):895–902, 1994.

[25] Yamada, H., Yamamoto, K. and Saito, T., *A non-linear normalization method for handprinted Kanji character recognition—line density equalization*, Pattern Recognit 23(9):1023–1029, 1990.

[26] Miyao, H., Maruyama, M., Nakano, Y. and Hananoi, T., *Off-line handwritten character recognition by SVM on the virtual examples synthesized from on-line characters. In: Proceedings of the eighth international conference on document analysis and recognition*, pp 494–498, 2005.

[27] Sethi, IK and Chatterjee, B., *Machine recognition of constrained handprinted Devanagari*, Pattern Recognit 9(2):69–75, 1977.

[28] Parui, SK, Chaudhuri, BB, Dutta and Majumder, D., *A procedure for recognition of connected hand written numerals*, Int J Syst Sci 13:1019–1029, 1982.

[29] Dutta, AK and Chaudhuri, S., *Bengali alpha-numeric character recognition using curvature features*, Pattern Recognit 26:1757– 1770, 1993.

[30] Bhattacharya, U., Das, TK, Datta, A., Parui, SK and Chaudhuri, BB, *A hybrid scheme for handprinted numeral recognition based on a self-organizing network and MLP classifiers*, Int J Patt Recog Artif Intell 16:845–864, 2002.

[31] Bhattacharya, U. and Chaudhuri, BB, *Fusion of combination rules of an ensemble of MLP classifiers for improved recognition accuracy of handprinted Bangla numerals*, In: Proceedings of the eighth international conference on document analysis and recognition, pp 322–326, 2005.

[32] Bhattacharya, U. and Chaudhuri, BB, *Handwritten numeral databases of Indian scripts and multistage recognition of mixed numerals*, IEEE Trans Pattern Anal Mach Intell 31(3):444–457, 2009.

[33] Rahman, AFR, Rahman, R. and Fairhurst, MC, *Recognition of handwritten Bengali characters: a novel multistage approach*, Pattern Recognit 35:997–1006, 2002.

[34] Bhowmick, TK, Bhattacharya, U. and Parui, SK, *Recognition of Bangla handwritten characters using an MLP classifier based on stroke features*, In: Proceedings of 11th international conference on neural information processing, pp 814–819, 2004.

[35] Bhattacharya, U., Parui, SK. and Shaw, B., *A hybrid scheme for recognition of handwritten Bangla basic characters based on HMM and MLP classifiers*, In: Proceedings of 6th international conference on advances in pattern recognition, pp 101–106, 2007.

[36] Hull, JJ, *A database for handwritten text recognition research*, IEEE Trans Patt Anal Mach Intell 16:550–554, 1994.

[37] Khosravi, H. and Kabir, E., *Introducing a very large dataset of handwritten Farsi digits* *and a study on their varieties*, Pattern Recognit Lett 28:1133–1141, 2007.

[38] Al-Maadeed, S., Elliman and D., Higgins, CA, *A database for Arabic handwritten text recognition research*, In: Proceedings of the eighth international workshop on frontiers in handwriting recognition, p 485, 2002.

[39] Su, T., Zhang, T. and Guan, D., *Corpus-based HIT-MW database for offline recognition of general-purpose Chinese handwritten text*, Int J Doc Anal Recognit 10:27–38, 2007.

[40] Saito, T., Yamada, H. and Yamamoto, K., *On the database ELT9 of handprinted characters in JIS Chinese characters and its analysis (in Japanese)*, Trans IECEJ 68-D(4):757–764, 1985.

[41] Al-Ohali, Y., Cheriet, M. and Suen, C., *Databases for recognition of handwritten Arabic cheques*, Pattern Recognit 36:111–121 , 2003.

[42] Noumi, T., Matsui, T., Yamashita, I., Wakahara, T. and Tsutsumida, T., Tegaki Suji database 'IPTP CD-ROM1' no ichi bunseki (in Japanese). In: 1994 autumn meeting of IEICE, vol D-309, September 1994, 1994.

[43] Bhattacharya, U., Shridhar, M. and Parui, SK, *On recognition of handwritten Bangla characters*, In: Proceedings of 5th Indian conference on computer vision, graphics and image processing, pp 817–828, 2006.

[44] George, A. and Gafoor, F., *Contourlet Transform Based Feature Extraction For Handwritten Malayalam Character Recognition Using Neural Network*, IRF Int. Conf. Chennai, pp: 107-110, 2014.

[45] Nemmour, H. and Chibani, Y., *Handwritten Arabic Word Recognition based on Ridgelet Transform and support Vector Machines*, IEEE, pp: 357-361, 2011.

[46] Moni, B. S., and Raju, G, *Modified Quadratic Classifier and Directional Features for Handwritten Malayalam Character Recognition*, IJCA Special Issue on Computer Science-New Dimensions and Perspectives, pp: 30-34, 2011.

[47] Nusaibath, C. and Ameera, M. P. M., *Off-line Handwritten Malayalam Character Recognition using Gabor Filters*, Int. J. of Computer Trends and Technology, pp: 2476-2479, 2013.

[48] Lecun,Y. and Bengio, Y., *Pattern Recognition and Neural Networks,* in Arbib, M. A. (Eds), The Handbook of BrainTheory and Neural Networks, MIT Press 1995.

[49] Singh, P. and Budhiraja, S., *Offline Handwritten Gurmukhi Numeral Recognition using Wavelet Transforms*, I. J Modern Education and Computer Science, pp: 34-39, 2012.

[50] Chen, G. Y. and Kegl, B., *Invarient Pattern Recognition using Contourlets and Adaboost*, Pattern Recognition Society Elsevier, pp: 1-13, 2012.

[51] Gonzalez, A., Bergasa, L. M., Yebes, J. J., and Bronte, S, *A Character Recognition Method in Natural Scene Images*, Pattern Recognition (ICPR), pp: 621-624, 2012.

[52] T. K. Bhowmik, P. Ghanty, A. Roy and S. K. Parui, *SVM-based hierarchical architec-tures for handwritten Bangla character recognition*, International Journal on Document Analysis and Recognition, vol. 12, no. 2, pp. 97-108, 2009.

[53] S. Basu, N. Das, R. Sarkar, M. Kundu, M. Nasipuri and D. K. Basu, *A hierarchicalapproach to recognition of handwritten Bangla characters*, Pattern Recognition, vol. 42, pp. 1467–1484, 2009.

[54] Bhattacharya, U., Shridhar, M., Parui, S. K., Sen,P. K. and Chaudhuri, B. B., *Offline recognition of handwritten Bangla characters: An efficient two-stage approach*, Pattern Analysis and Applications, vol. 15, no. 4 , pp. 445-458, 2012.

[55] Rahman, Md. M., Akhand, M. A. H., Islam, S., Shill, P. C. and Rahman, M. M. H.,*Bangla Handwritten Character Recognition using Convolutional Neural Network,* Int.J. Image, Graphics and Signal Processing, vol. 08, pp. 42-29, 2015.

[56] Center for Microprocessor Application for Training Education and Research Retrived July 10, 2017 from https://code.google.com/archive/p/cmaterdb/

[57] Kaur, K., and Garg, N. K., *Use of 40-point Feature Extraction for Recognition of Handwritten Numerals and English Characters*, IJCTA, pp: 1409-1414, 2014.

[58] Aggarwal, A., Rani, R. and RenuDhir, *Handwritten Devanagari Character Recognition Using Gradient Features*, Pattern Recognition (ICPR), pp: 621-624, 2012.

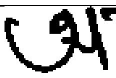
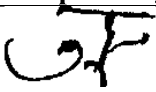
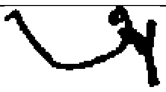
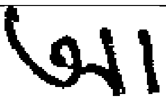
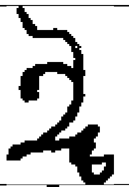
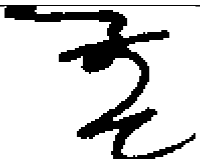
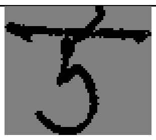
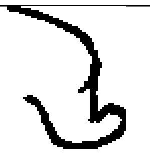
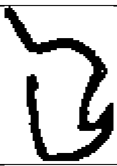
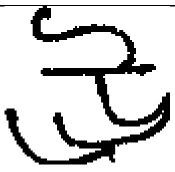
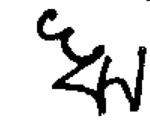
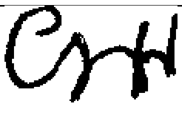
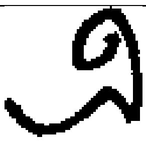
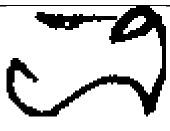
[59] Ciresan, Dan, Meier, U., and Schmidhuber, J., *Multi-column deep neural networks for image classification,* IEEE Conference on Computer Vision and Pattern Recognition (New York, NY: Institute of Electrical and Electronics Engineers (IEEE)), 2012.

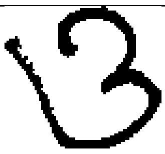
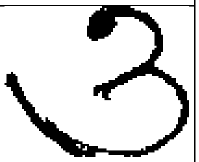
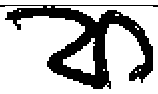
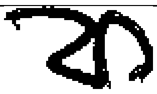
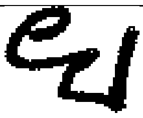
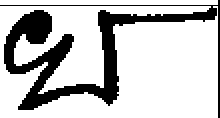
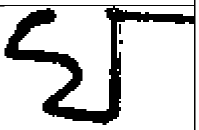
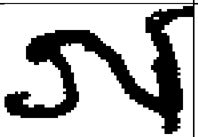
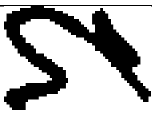
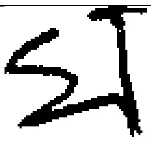
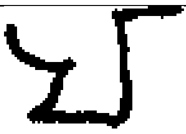
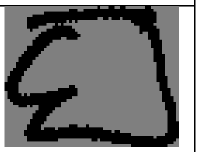
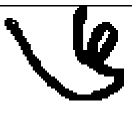
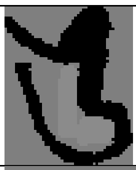
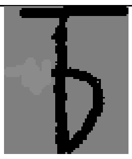
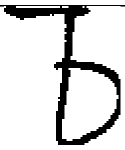
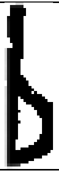
[60] Ciresan, Dan, Meier, U., Masci, J., Gambardella, L.M., and Schmidhuber, J., *Flexible, High Performance Convolutional Neural Networks for Image Classification*, Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence-Volume Two: 1237–1242, 2013.

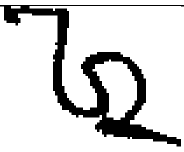
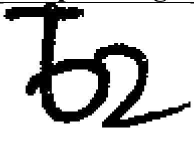
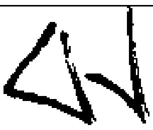
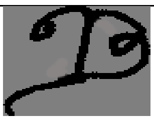
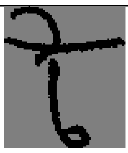
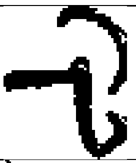
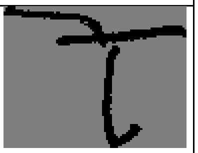
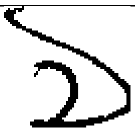
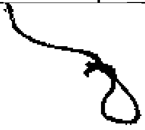
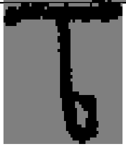
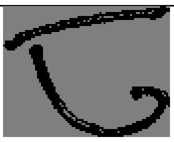
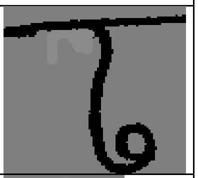
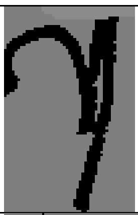
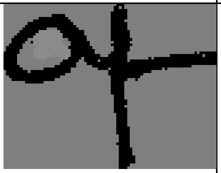
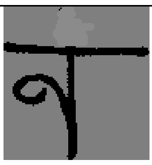
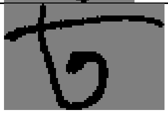
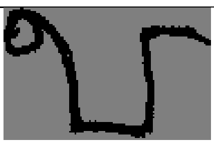
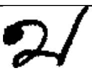
[61] Russakovsky, O., *ImageNet Large Scale Visual Recognition Challenge*, International Journal of Computer Vision, 2014.

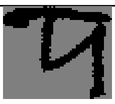
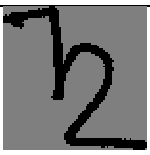
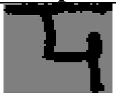
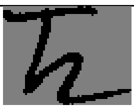
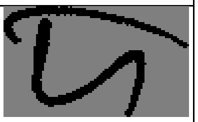
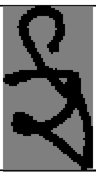
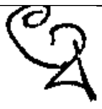
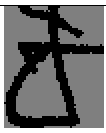
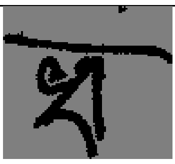
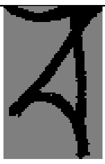
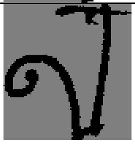
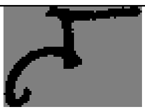
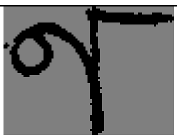
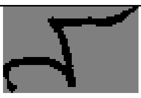
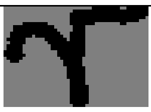
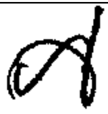
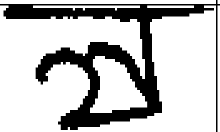
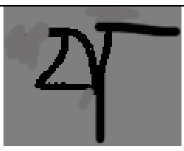
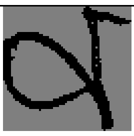
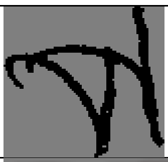
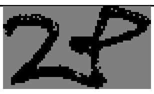
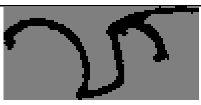
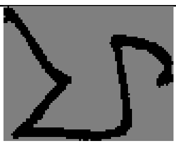
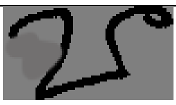
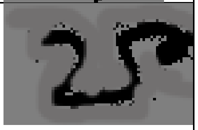
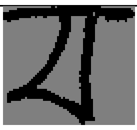
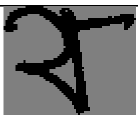
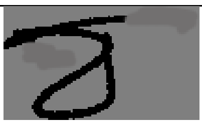
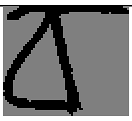
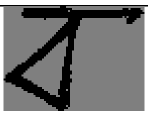
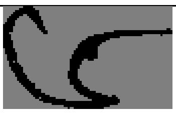
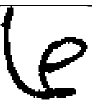
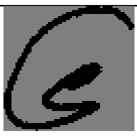
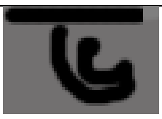
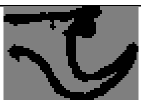
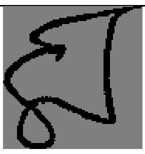
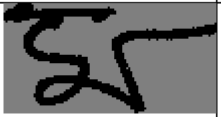
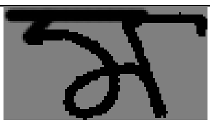
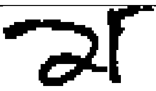
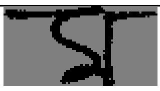
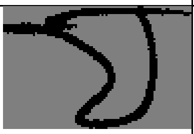
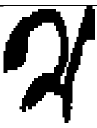
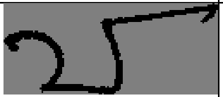
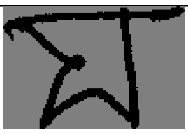
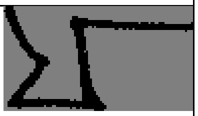
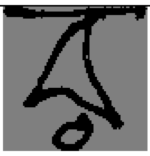
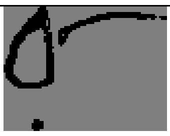
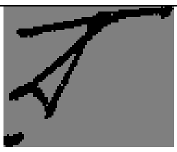
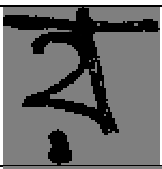
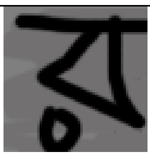
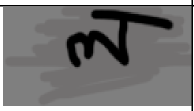
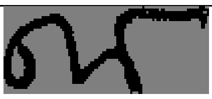
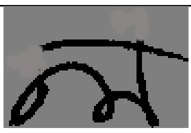
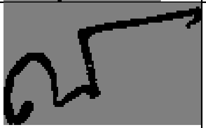
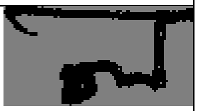
Appendix A

Table A.1: Samples of Database CMATERdb 3.1.2:

Character	Sample Images				
অ					
আ					
ই					
ঈ					
উ					
ঊ					
ঋ					
এ					

Character	Sample Images				
ঐ					
ও					
ঔ					
ক					
খ					
গ					
ঘ					
ঙ					
চ					

Character	Sample Images				
ছ					
জ					
ঝ					
ঞ					
ট					
ঠ					
ড					
ঢ					
ণ					
ত					
থ					

Character	Sample Images				
দ					
ধ					
ন					
প					
ফ					
ব					
ভ					
ম					
য					
র					
ল					

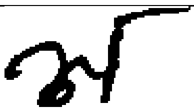
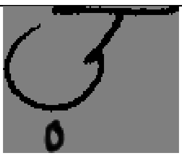
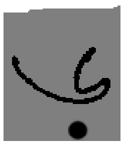
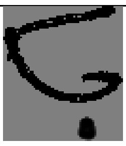
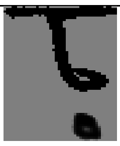
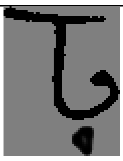
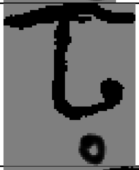
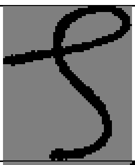
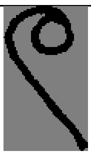
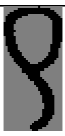
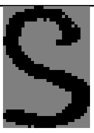
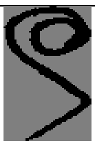
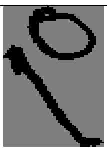
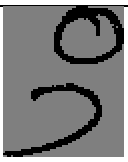
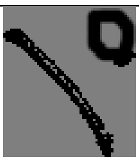
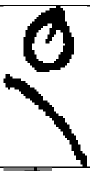
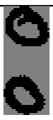
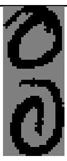
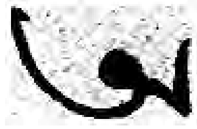
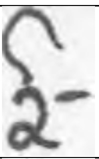
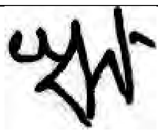
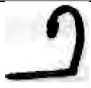
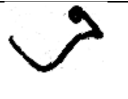
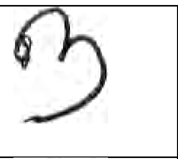
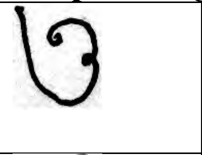
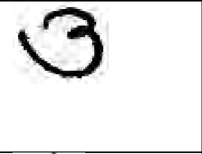
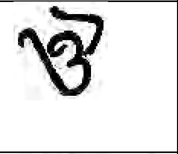
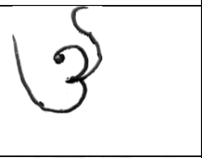
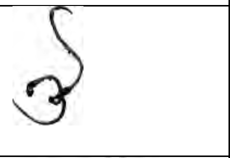
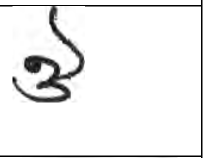
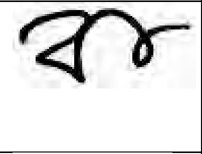
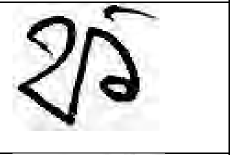
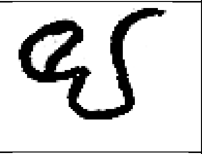
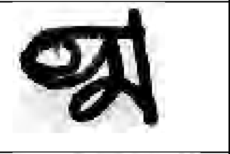
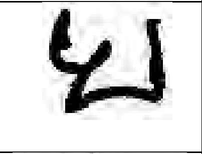
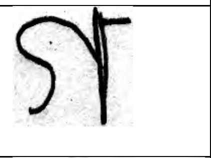
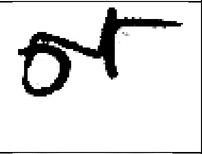
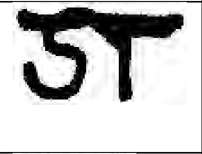
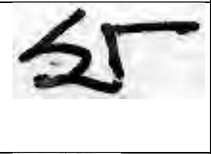
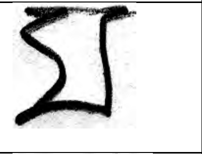
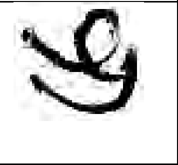
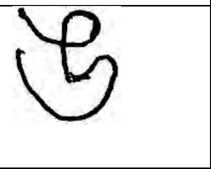
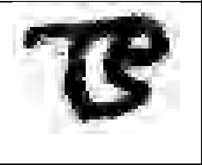
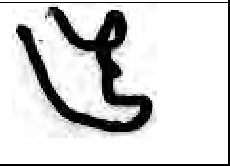
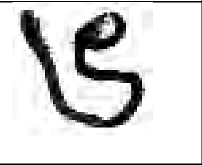
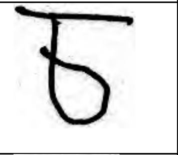
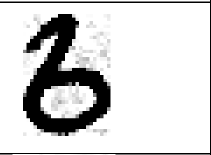
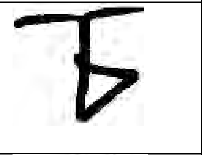
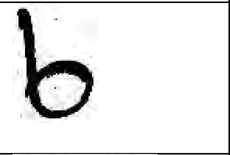
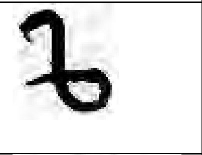
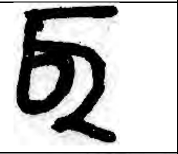
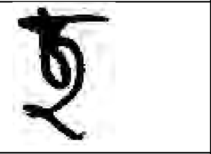
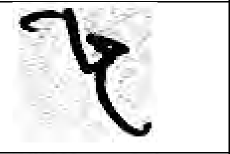
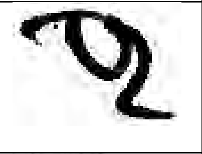
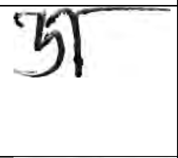
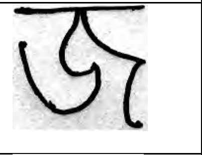
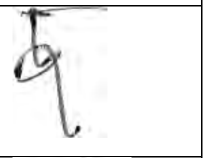
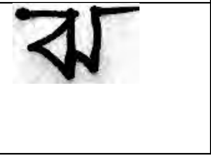
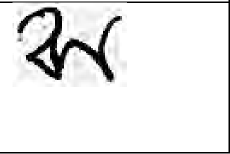
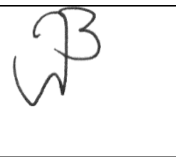
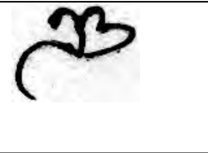
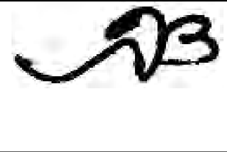
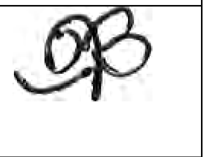
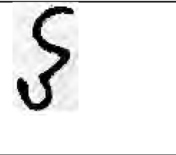
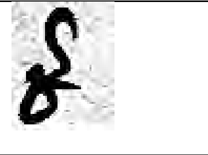
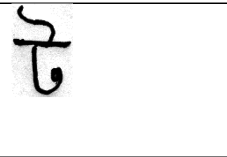
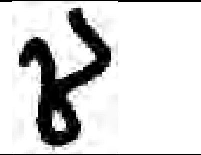
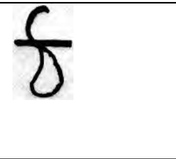
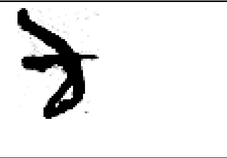
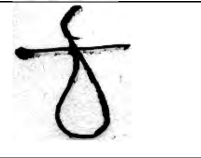
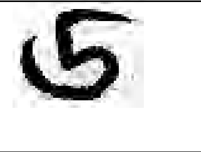
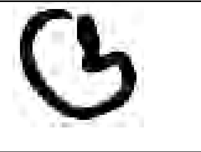
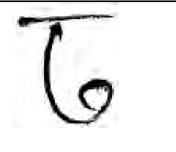
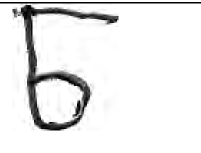
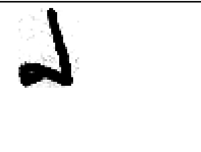
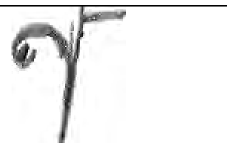
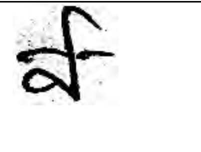
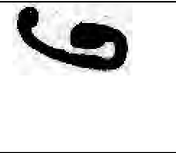
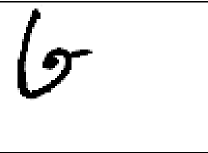
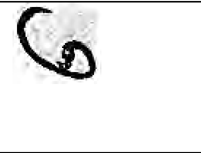
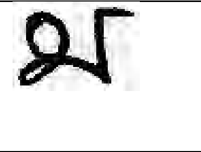
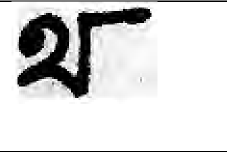
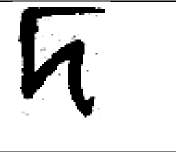
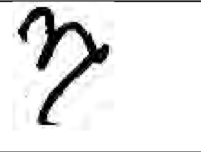
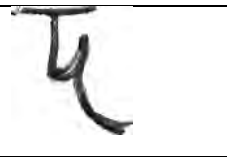
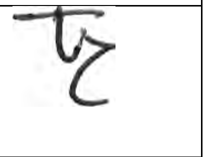
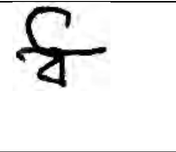
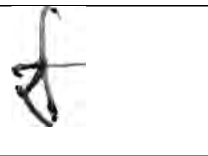
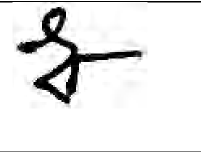
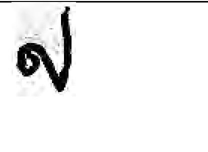
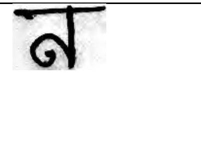
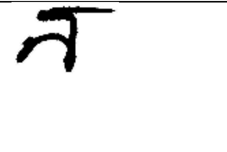
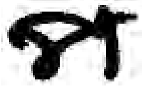
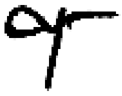
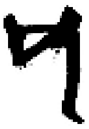
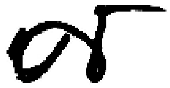
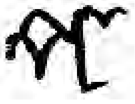
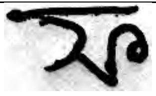
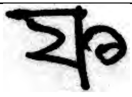
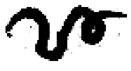
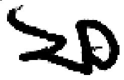
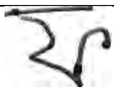
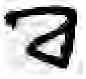
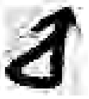
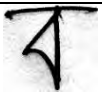
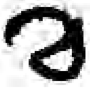
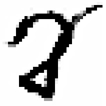
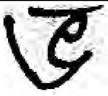
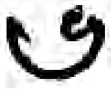
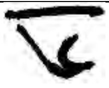
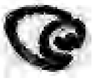
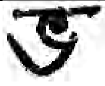
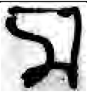
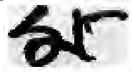
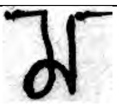
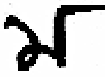
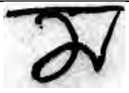
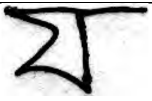
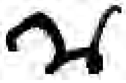
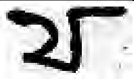
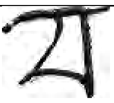
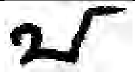
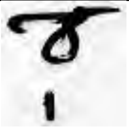
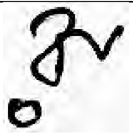
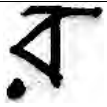
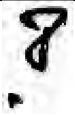
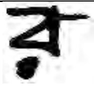
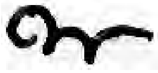
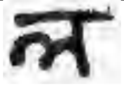
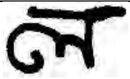
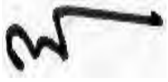
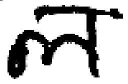
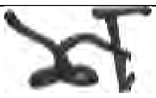
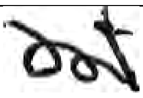
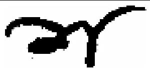
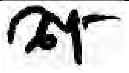
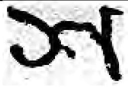
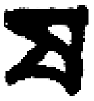
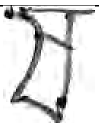
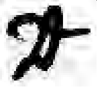
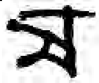
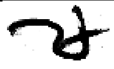
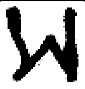
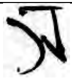
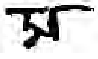
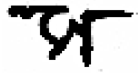
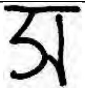
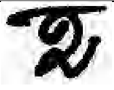
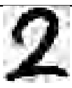
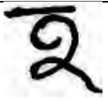
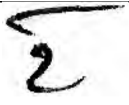
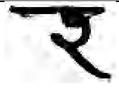
Character	Sample Images				
শ					
ষ					
স					
হ					
ড়					
ঢ়					
য়					
ৎ					
s					
:					
u					

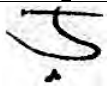
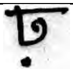
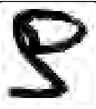
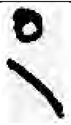
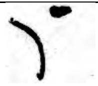
Table A.2: Samples of Database BBCD:

Character	Sample Images				
অ					
আ					
ই					
ঈ					
উ					
ঊ					
ঋ					
এ					
ঐ					

Character	Sample Images				
ও					
ঔ					
ক					
খ					
গ					
ঘ					
ঙ					
চ					
ছ					
জ					
ঝ					

Character	**Sample Images**				
ঞ					
ট					
ঠ					
ড					
ঢ					
ণ					
ত					
থ					
দ					
ধ					
ন					

Character	Sample Images				
প					
ফ					
ব					
ভ					
ম					
য					
র					
ল					
শ					
ষ					
স					
হ					

Character	Sample Images				
ড়					
ঢ়					
য়					
ৎ					
s					
:					
u					