A Script Independent Multi-Scale Deep Quad Tree Based Feature Extraction Technique for Recognition of Handwritten Indic Characters

A thesis submitted in partial fulfilment of the requirement for the

Degree of Master of Computer Science and Engineering

of

Jadavpur University

By

Ritesh Sarkhel

Registration No.: 103612 of 2008-09

Examination Roll No.: M4CSE1611

Under the Guidance of

Dr. Nibaran Das

Department of Computer Science and Engineering

Jadavpur University, Kolkata-700032

India

2016

FACULTY OF ENGINEERING AND TECHNOLOGY

JADAVPUR UNIVERSITY

Certificate of Recommendation

This is to certify that the dissertation entitled "A Script Independent Multi-scale Deep Quad Tree Based Feature Extraction Technique for Recognition of Handwritten Indic Characters" has been carried out by Ritesh Sarkhel (University Registration No.: 103612 of 2008-09, Examination Roll No.: M4CSE1611) under my guidance and supervision and be accepted in partial fulfilment of the requirement for the Degree of Master of Computer Science and Engineering. The research results presented in the thesis have not been included in any other paper submitted for the award of any degree in any other University or Institute.

……………………………………………………

Dr. Nibaran Das (Thesis Supervisor)

Department of Computer Science and Engineering

Jadavpur University, Kolkata-32

Countersigned

…………………………………………………..

Prof. Debesh Kumar Das

Head, Department of Computer Science and Engineering,

Jadavpur University, Kolkata-32.

…………………………………………………..

Prof. Sivaji Bandyopadhyay

Dean, Faculty of Engineering and Technology,

Jadavpur University, Kolkata-32.

FACULTY OF ENGINEERING AND TECHNOLOGY

JADAVPUR UNIVERSITY

Certificate of Approval*

This is to certify that the thesis entitled "A Script Independent Multi-scale Deep Quad Tree Based Feature Extraction Technique for Recognition of Handwritten Indic Characters" is a bona-fide record of work carried out by Ritesh Sarkhel in partial fulfilment of the requirements for the award of the degree of Master of Computer Science and Engineering in the Department of Computer Science and Engineering, Jadavpur University during the period of June 2015 to May 2016. It is understood that by this approval the undersigned do not necessarily endorse or approve any statement made, opinion expressed or conclusion drawn therein but approve the thesis only for the purpose for which it has been submitted.

……………………………………………………………………………..

Signature of Examiner 1

Date:

……………………………………………………………………………..

Signature of Examiner 2

Date:

*Only in case the thesis is approved

FACULTY OF ENGINEERING AND TECHNOLOGY

JADAVPUR UNIVERSITY

Declaration of Originality and Compliance of Academic Ethics

I hereby declare that this thesis entitled "A Script Independent Multi-scale Deep Quad Tree Based Feature Extraction Technique for Recognition of Handwritten Indic Characters" contains literature survey and original research work by the undersigned candidate, as part of his Degree of Master of Computer Science & Engineering.

All information have been obtained and presented in accordance with academic rules and ethical conduct.

I also declare that, as required by these rules and conduct, I have fully cited and referenced all materials and results that are not original to this work.

Name: Ritesh Sarkhel

Registration No: 103612 of 2008-09

Exam Roll No.: M4CSE1611

Thesis Title: A Script Independent Multi-scale Deep Quad Tree Based Feature Extraction Technique for Recognition of Handwritten Indic Characters

…..…………………………………………..

Signature with Date

Acknowledgement

I would like to start by thanking the holy trinity for helping me deploy all the right resources and for shaping me into a better human being.

I would like to express my deepest gratitude to my advisor, **Dr. Nibaran Das**, Assistant Professor, Department of Computer Science and Engineering, Jadavpur University for his admirable guidance, care, patience and for providing me with an excellent atmosphere for doing research. Our numerous scientific discussions and his many constructive comments have greatly improved this work.

Words cannot express my indebtedness to **Prof. Mita Nasipuri**, Department of Computer Science and Engineering, Jadavpur University for her amazing guidance and supervision. I am deeply grateful to her for the long discussions that helped to enrich the technical content of this manuscript. Without her enthusiasm, encouragement, support and continuous optimism this thesis would hardly have been completed.

I would like to thank **Prof. Debesh Kumar Das**, Head, Department of Computer Science and Engineering, Jadavpur University also **Prof. Sivaji Bandyopadhyay**, Dean, Faculty of Engineering and Technology, Jadavpur University for providing me with moral support at times of need.

I would like to acknowledge Mr. Aritra Das and Mr. Swarnendu Ghosh for assisting me in the same task and helping me to understand many Python and Theano libraries and packages needed to complete the thesis.

Most importantly none of this would have been possible without the love and support of my family. I extend my thanks to my parents, especially to my mother whose forbearance and whole hearted support helped this endeavor succeed. Finally I would also like to mention my brother without whose continuous support this work could have never been able to flourish.

This thesis would not have been completed without the inspiration and support of a number of wonderful individuals — my thanks and appreciation to all of them for being part of this journey and making this thesis possible.

……………………………………………..

Ritesh Sarkhel

Registration No: 103612 of 2008-09

Exam Roll No.: M4CSE1611

Department of Computer Science & Engineering

Jadavpur University

Table of Contents

List of figures

List of tables

Preface

The purpose of this document is to be treated as the master's thesis by the author for the partial fulfilment of the Masters in Computer Science and Engineering curriculum in Jadavpur University. In this document, the author has presented an overview of his original contribution and the background knowledge necessary to understand its significance.

In the present work, a multi-scale deep quad-tree based CNN feature extraction technique has been described. The proposed method has been tested on isolated handwritten character images. Styles of handwriting vary from writer to writer. It may also vary for a single individual based on various different factors. Therein lies the challenge of designing a powerful and efficient feature set for handwritten character recognition. Although there has been significant improvement in the development of Optical Character Recognition (OCR) systems for many European languages including English, German, French, Spanish etc. and Asian languages such as Chinese, Japanese etc., surprisingly there has not been much focus on Indian languages until recently. In this work, the author has focused on the recognition of handwritten characters of Indian languages. The document is divided in several chapters for better readability. The chapters provide an overview of the necessary background knowledge before introducing the author's contribution.

Before it all begins, the author would again like to thank his family for their constant and unconditional support, without whom none of this would have been possible. The author is also

immensely grateful to his research guide Dr. Nibaran Das, whose kind words and judicious insight has always been an inspiration to him.

Introduction

The main objective of optical character recognition (OCR) applications is to automate the recognition of digitally scanned page of printed or handwritten texts. OCR applications have been one of the most successful applications of intelligent pattern recognition task since the 1950's. Due to its high quality performance for printed or handwritten characters, OCR systems have also become a major commercial success. This success of OCR applications to recognize printed texts however, is still to be translated to unconstrained handwritten texts. One of the most challenging facets in recognition of handwritten characters, no matter what language it belongs to, is to train the system in a way so that it can adjust for the largely varying handwriting style from one individual to another.

Since the focus of research on optical character recognition shifted from printed text to handwritten character recognition, there has been a lot of activity in this field. However, until the last decade, most of the works seemed to focus on languages which mainly use Roman scripts. There has been surprisingly little work done for the recognition of unconstrained handwritten character recognition of Indian languages. Huge alphabet and complex character shapes make the job of recognizing handwritten Indian characters and numerals much more challenging. Although several important works[1][2][3] have emerged from this new found focus on Indian languages, the proposed approaches seem to focus on only one or two specific languages at a time. Although some of the researchers have tried to propose features which are generic in terms of being tailored to only a specific script, number of works focusing on this approach is still relatively few. From feature selection to selection of the classifier models, the focus has mainly been on proposing script specific or language specific approaches.

The dearth of a ubiquitous, script independent approach for the number of Indian languages is what motivated us to propose such a system. Instead of taking an explicit feature based approach and toiling to propose a one-size-fits-all feature of the hugely varying, complex Indian languages, we have explored a different approach in the present work. We hypothesize that non-explicit features can be extracted from a wide variety of Indic languages using a singular approach, which will lead us to their successful recognition. A multi-scale, multi-column deep architecture has been proposed in the present work for this purpose. The deep neural network based non-explicit feature extraction method and the proposed multi-scale multi-column convolutional neural network (MMCNN) based architecture marks the contribution of the present work.

The proposed Convolutional Neural Network based architecture has been tested for six publicly available benchmark datasets of isolated handwritten characters and numerals of Indian languages. It has been proved from the experimental results that the proposed approach provides state-of-the-art results for all of these datasets. Subsequently, the superiority of this singular, ubiquitous approach towards the recognition of handwritten characters of Indic languages has been statistically proved.

Chapter 1

An evolution of OCR systems

Handwriting is a skill which stylistically varies from person to person, even for a single individual from time to time depending on various factors. Handwritten texts consist of artificial markings on a surface and they attempt to communicate something in relation to a specific language[4]. These markings, representing characters of a particular language combine to form words and sentences pertaining to a language, according to the rules of that language[5]. Handwritten texts were first introduced to expand human memory and facilitate communication. Even in today's world, a pen and paper outwits the need to use digital computers, PDAs, fax machines and printers in many cases. With every technology push, the role of handwriting and handwritten messages have become more well-defined and the niche of handwriting has become more popularized. The latest example of this is the introduction of digital pens to PDAs and tablets. As a thumb rule it seems that as the length of handwritten messages decrease, the number of people using handwriting increases[6].

Several types of recognition, interpretation and identification tasks can be associated to handwritten texts. In this section, we provide definitions of some of these tasks associated with the analysis of handwritten texts. *Handwriting Recognition*[4] is the task of transforming a language represented in its spatial form i.e. the graphical markings to the symbols they represent. The symbolic representation for most of the languages are represented by 8-bit ASCII characters or 16-bit Unicode[7] representations. *Handwriting Interpretation*[4] is the task of understanding the meaning of the symbol represented by the graphical markings. The process of handwritten address interpretation in automated postal sorting machines is an example of *Handwriting Interpretation. Handwriting Identification*[4] is the process of identifying writers from handwritten texts, provided that different writers have different handwriting styles. *Handwriting Recognition* and *Handwriting Interpretation* have applications in forensic analysis. It can be used to detect individual writers based on their individualistic writing styles. It can also be used as an aid for the blinds, automatic text entry from handwritten forms in library catalogue processing, ledgering, desktop publication, document processing, language processing, automatic postal document sorting etc.

1.1. Handwriting Input

Handwriting data can be converted to a digitized form by either scanning a paper with handwritten texts written on a paper or directly from a digital pen on an electronic display such as PDAs or digitizers etc. with a liquid crystal display. These approaches can be classified into two major categories: *on-line* and *off-line handwriting recognition.* In case of *on-line handwriting recognition,* in addition with the image of the text, the sequence of two-dimensional points comprising the text is also stored in increasing order. The supplemental spatio-temporal information provided by the stroke sequences prove to be very helpful in recognizing the texts successfully. On the other hand, in case of *off-line handwriting recognition*, only digital images of the handwritten texts are taken as the input. Where the on-line case deals with the spatio-temporal information of the handwritten texts, the off-line case handles only the spatio-luminance information available with the digitized image. The difference between the inputs of *on-line* and *off-line handwritten texts* is shown in Fig. 1.1.

It should however be noted that the recognition rates reported for *on-line handwritten* texts are higher than their *off-line* counterpart.

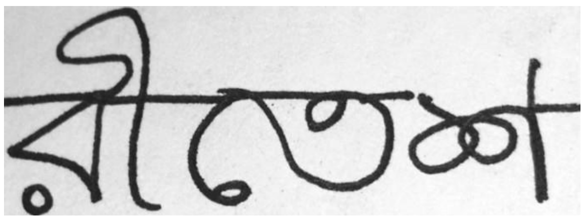
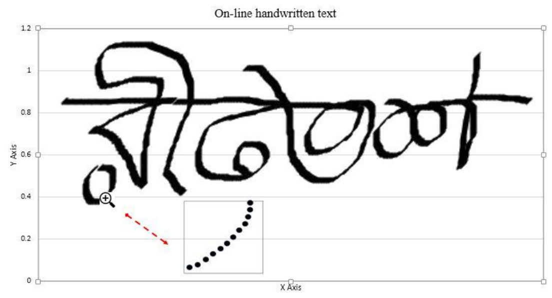
Input for off-line handwritten recognition	*Input for on-line handwritten recognition*
	

Fig. 1.1: *A comparison between on-line and off-line handwritten texts*

1.2. The evolution of Optical Character Recognition (OCR) systems

Optical Character Recognition (OCR) system is a process of automatic recognition of characters from an optically scanned or digitized page of texts. OCR is one of the most fascinating and challenging areas of pattern recognition. It can contribute immensely to the advancement of an automation process and can improve the interface between man and machine in many applications. The origin of OCR systems can be found in 1870, when Carey invented a retina scanner, an image transmission system based on the mosaic of photocells[2][8]. The next big breakthrough came in 1890, when Nipkow[2] invented the sequential scanner which proved to be very useful for modern television and reading machines. However, OCR systems were initially considered as an aid to the blind and successful attempts to this were made by Tyurin[2] in 1900. Based on effectiveness, robustness and versatility the development of OCR systems can be broadly classified into four[2] major generations.

The first generation OCR systems are characterized by the constrained character shapes it reads. One of the examples of such first generation OCR systems is IBM 1418, which was developed only to read only a special font, IBM 407[9]. These OCR systems function solely based on template matching by utilizing positional relationships in the character shapes.

The second generation systems are characterized by their ability to recognize a set of machine printed characters and handwritten characters. Functioning based on structural analysis method, these systems came into appearance from the middle of 1960s to early 1970s[9]. Some of the popular systems of this era were the famous IBM 1287 system[9] and the first ever automatic postal sorting machine developed by Toshiba.

Third generation OCR systems can be characterized by their ability to recognize larger set of poor quality machine printed characters and larger set of handwritten characters. Marked from 1965 to 1975, commercially successful OCR systems were one of the major contributions[10] of this era.

The fourth generation OCR systems are capable to process complex documents containing texts along with graphical representations, tables, unconstrained handwritten texts, poor quality or noisy documents like fax, coloured documents etc. Several commercially successful large scale deployment of OCR systems are made possible during this era. About 60%[10] postal sorting of handwritten letters are made via OCR systems in the United States. Several successful OCR systems working as an aid to blinds have also come to surface during this period of time. One such example is the integrated OCR system, developed by Xerox-Kurzweil, which transforms printed of handwritten text written in English to composite speech in the same language. Successful commercial deployment of OCR systems has been possible for several different languages as well such as Arabic, German, Chinese, Roman, Japanese scripts[11][12][13][14][15].

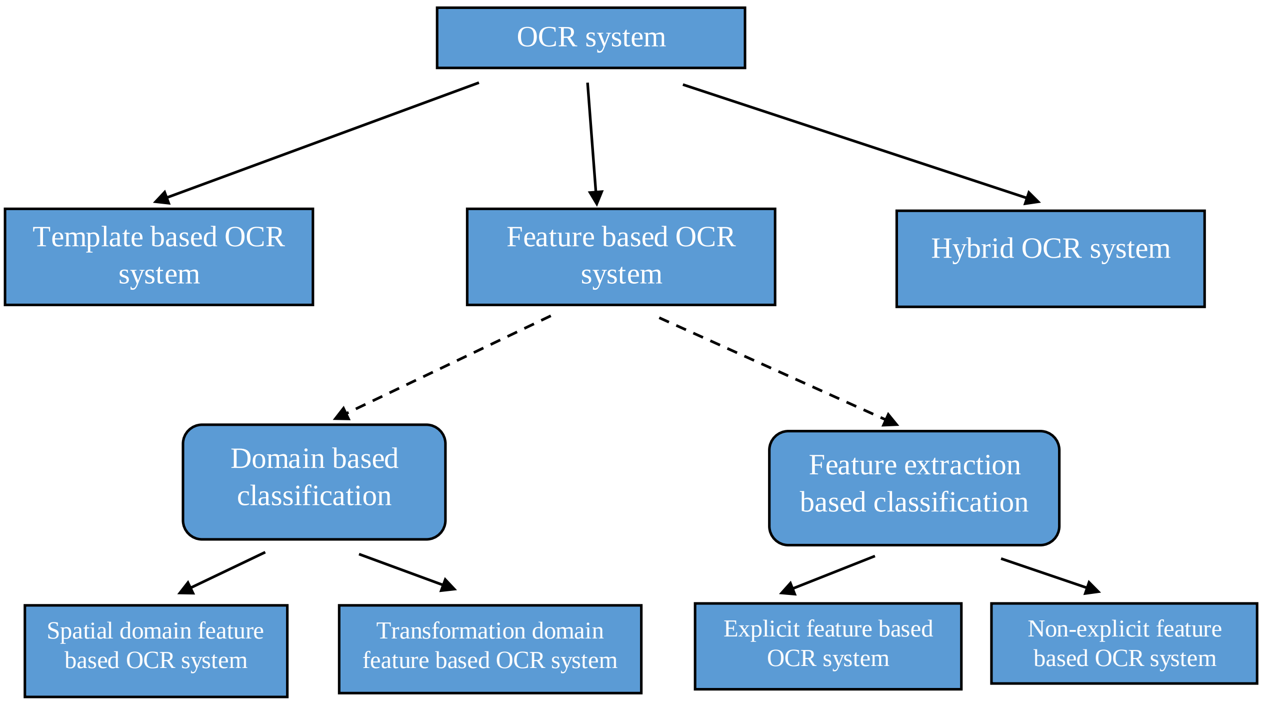
1.2.1. A brief overview on the topography of OCR systems

Traditionally pattern recognition tasks can be classified into two major categories: *template based* and *feature based*. Earlier attempts of OCR systems were mainly dependent on template based techniques where character recognitions were done based on the degree of correlation between the character image and the template. However, modern OCR systems[1] also incorporate feature based approaches with traditional template matching techniques to produce better results.

Feature based approaches can be classified into two[2] major categories: *spatial domain* feature based techniques and *transform domain* techniques. Whereas spatial domain features[16][17][3][18] are extracted directly from the raw pixel data of the digitized image, transform domain features[19][20][21][22] are extracted from the transformed pattern image. In case of transform domain features, digitized images are first transformed to another space using Cosine, Slant, Fourier, Wavelet etc. transformation techniques and subsequently features are extracted. After features are extracted, sophisticated classification models are utilized for recognition of printed or handwritten character images. Researchers working on OCR systems have proposed a wide array of features for automatically recognizing printed or handwritten characters. While most of these features are generic, some[23][3] of them utilize script specific properties to improve the performance of the underlying classifier. Syntactic or formal grammar based features[24][23], moment based features[25], graph-theoretic approaches[26], shadow based features[27][18], gradient based features[28][16] etc. are some of such examples. Contrary to the above mentioned genre of OCR systems, where features are handcrafted and need to be explicitly extracted from digitized pattern images, some researchers have proposed OCR systems where features do not have to be explicitly extracted. As raw pattern image or normalized images are fed into the system, it adjusts itself accordingly to minimize the misclassification error. Artificial neural network[12][29][30][31][32] based approaches and HMM or Markov-model based approaches[33][34][35] to automatically extract statistically significant features are examples of such *non-explicit feature based*[2] OCR systems. As described above, a topography of OCR systems is shown in Fig. 1.2.

1.3. OCR of Indian languages

Despite of huge commercial success of modern era OCR systems, until recently the focus of OCR research has been mainly circling around languages based on Roman scripts. In the present work, we have focused on the development of handwritten character recognition of Indian languages.



OCR system
Template based OCR system
Feature based OCR system
Hybrid OCR system
Domain based classification
Feature extraction based classification
Spatial domain feature based OCR system
Transformation domain feature based OCR system
Explicit feature based OCR system
Non-explicit feature based OCR system

Fig. 1.2: *A schematic diagram of the topography of OCR systems*

1.3.1. Properties of Indian languages

There are eighteen official languages in India. Among these Hindi and Bangla are the first and second[2] most popular language in India and fourth and fifth[2] most popular languages in the world. Although stylistically varied from each other, most of the Indian languages have originated from the Brahmi scripture[36]. Some of these scripts are used in more than one language. For example, the Devanagari script is used for Sanskrit, Hindi, Marathi, Nepali and Rajasthani languages. Similarly Bangla script is used for Bangla and Assamese languages. Fig. 1.3 shows example of some of the official Indian languages.

The concept of upper and lower case characters is not present in Indian language alphabets. The writing direction for most of the languages are from left to right except Urdu and some other scripts belonging to the Perso-Arabic[2] group of scripts. Apart from the familiar vowels and consonants i.e. the *basic characters*, Indian language alphabets also have *compound characters. Compound characters* are generally comprised of two or more *basic characters.*

Ten Rupees

(a)

दस रुपये

(b)

দশ টাকা

(c)

દસ રૂપિયા

(d)

दहा रुपयांची

(e)

பத்து ரூபாய்

(f)

పది రూపాయలు

(g)

പത്ത് രൂപയാണ്

(h)

Fig. 1.3: *(a) - (h) Respective representations of English, Hindi, Bangla, Gujarati, Marathi, Tamil, Telugu and Malayalam scriptures denoting the same word*

For example, in Bangla alphabet, there are 50 *basic characters* and 334 *compound characters*[16]. Shapes of these *compound characters* are usually more complex compared to their *basic* counterparts. Some samples of *basic* and *compound* characters of the Bangla alphabet is shown in Fig. 1.4.

প ল

(a)

প্র শ্ন

(b)

Fig. 1.4: *Samples of Bangla (a) basic and (b) compound characters*

1.3.2. Related works on OCR of Indian languages

Due to its huge popularity, most of the works focusing on developing of an OCR for Indian languages have worked on the recognition of handwritten characters or digits of Bangla and Hindi scripts. However, as the present work tries to encompass more than just two of the most popular Indian languages, a brief overview on related previous works focusing on a wide array of Indian languages is presented in this section. In the present work, we have covered isolated handwritten characters and numerals of Bangla, Hindi, Tamil and Telugu languages.

As mentioned before, Hindi is written using Devnagari script. The works on OCR of Devnagari scripts started from around 1970s. The first significant contribution to this field was made by R M K Sinha[37]. A syntactic pattern analysis system was proposed and applied to Devnagari script in this work. Mahabala and Sinha[23] proposed a syntactic pattern analysis method which uses an embedded picture language to describe the primitive structural components of a character. Sethi et al.[38] proposed a Devnagari handwritten numeral recognition system using decision tree classifier. They also advocated using the presence or absence of certain primitive shapes or structures like horizontal line, vertical line, C curve, Slant curve etc. The OCR system proposed by Bajaj et al.[39] for the recognition of Devnagari numerals use two types of features in the recognition process. The first kind provides a coarse shape classification whereas the second type of features provide more intrinsic details of the character shapes. Recently, Kekre et al.[40] have proposed an OCR for Devnagari numerals recognition using LBG quantization with gradient masks which shows promising results. A concise overview on the development of OCR systems for recognition of Devnagari texts can be found in [41].

Although works[42][43] on a comprehensive Bangla OCR system started from the early 1990's, significant performance on Bangla script has not been reported till the middle[2] of nineties. The first work on the complete Bangla OCR of printed texts is due to Chaudhuri et al.[1]. The method proposed in [1] takes a hybrid approach for recognition of Bangla characters. While Bangla Basic characters are recognized by a feature based decision tree classifier, Bangla Compound characters are classified using a template matching technique on a pre-computed cluster. Garain et al.[44] have proposed a scale and style invariant feature and run number based template matching technique for recognition of Bangla Compound characters. More recently, Das et al.[45] have put forth a Genetic Algorithm based approach to select the most informative set of local regions from the handwritten Bangla characters. Gradient, shadow and convex hull based features are extracted and the images are classified using a SVM based classifier. A two-pass approach for recognition of handwritten Bangla characters have been proposed by Das et al. in [3]. The first pass is used to coarsely cluster the characters and the subsequent second pass is used to assign class membership to the characters from each cluster. A cost effective OCR system for isolated handwritten Bangla characters has been proposed by Sarkhel et al[17]. The proposed method uses axiomatic fuzzy set theory for combining the pareto-optimal fronts achieved by NSGA-II and NSHA algorithms, therefore finding the most optimized configuration for the OCR, both performance wise and associated recognition cost wise. There has been significant improvement in developing an OCR for recognition of Bangla numerals as well. Khan et al.[46] have obtained promising results by proposing a feature based approach for handwritten isolated Bangla digit recognition using a Sparse Representation Classifier (SRC). Eight directional quad tree based features have been proposed by Roy et al.[28] which shows exciting results. Basu et al.[18] have proposed a feature-based hierarchical approach for segmentation and recognition of Bangla numerals from Bangla texts.

Development of an OCR for the recognition of printed or handwritten Tamil characters started in the 1970s. Siromoney et al.[47] proposed a encoded string dictionary for the representation of Tamil characters. The scheme employs row wise and column wise scanning of the feature matrix for the recognition of characters. Chandrasekaran et al.[48] took a similar approach of encoding characters to a binary array. The feature matrix is composed by scanning the array both row wise and column wise. Features depend on the nature and number of runs of ones in the array. Sutha et al.[49] proposed a Fourier descriptor based feature extraction technique after tracing the boundary of the character. The characters are classified using a feed forward multi-layer perceptron. After thresholding, skeletonizing and line segmentation, a feature based approach is taken for character recognition. A SVM based classifier is used for the classification tasks. Chinnuswamy et al.[50] have proposed a graph-based approach from recognition of handwritten Tamil characters. Every character is transformed into a labelled graph which essentially represents the relationship between the primitive components comprising of the character shapes. Recognition tasks are carried out by performing topological matching on the test characters and calculating the degree of correlation.

The first reported work on OCR of Telugu script is due to Rajasekaran et al.[51]. They proposed a two pass approach where the primitive shapes are recognized at the first stage. After the primitive shapes are recognized based on a knowledge based search and removed, rest of the pattern images are thresholded, skeletonized and the boundary is encoded by tracing points along. The encoded boundary shapes are classified using a decision tree classifier. Sukhaswami et al.[52] proposed a neural network based approach for printed Tamil character recognition. They proposed a multiple neural network with associative memory (MNNAM) for parallel processing of the train set. A two stage OCR system for the recognition of printed Telugu characters is proposed by Srinivas et al.[53]. After preprocessing, binarization and skew correction, a feature based approach is taken to group the test images into clusters during the first stage while class membership is assigned to the pattern images in each cluster. Negi et al.[54] have proposed a composite, connected component and fringe distance based template matching approach for the recognition of printed Tamil characters. There has been significant number of works on the development of OCR for Tamil digits too. Singh et al.[55] have taken a feature based approach where they have extracted features such as the bounding box co-ordinates, area of the bounding box, centroid, major and minor axis, equiv-distance etc. and used this feature set for the classification on handwritten numerals, using an MLP as the classifier. Rajashekararadhya et al.[56] have proposed a feature based approach where zone based distance metrics are computed for each character image and the classification is done using a feed forward neural network. Promising results have also been achieved by using eight directional gradient based features in [57].

Based on our literature survey, *template based* approach (shown in Fig. 1.2) towards OCR is not only difficult to implement as the task of proposing generic character templates factoring in all possible deformations is extremely hard, performance of such systems is also not satisfactory which can be ultimately extended to commercial success[58][2]. That is why in the present work, we have explored the avenue of feature based approach towards OCR for the recognition of handwritten Indian characters and numerals. Before going into the schemes undertaken in our work, a brief overview on the approaches of feature based OCR taken by contemporary researchers is presented in Chapter 2. Self-adaptive, non-explicit feature extraction from pattern images is described in details in Chapter 3.

Chapter 2

An overview of feature based OCR systems

What makes the task of handwritten character recognition exceptionally hard is the stylistic variability of handwritten texts from person to person. Shape, size even orientation of handwritten characters may vary from one individual to another. Considering the complex shape of the characters and numerals of many Indian languages, if a multilingual OCR for recognition of Indian characters and numerals is to be designed, the challenge increases many folds[2][4]. As the system not only has to deal with the stylistic variability of handwritten characters, it also has to cope with the specificities of large variety of alphabets from multiple languages. Several different approaches have been taken by researchers in the pattern recognition community to address this problem. Thousands of different features[58] have been proposed for the development of feature based OCRs. These features are designed in such a way that they can be used to identify the invariant local patterns within the image which will in turn help to identify the image itself. The success of handwritten character recognition system depends on how well these features are designed i.e. to what extent they can extract and incorporate information about invariant local patterns from the images of isolated characters or numerals. The following section contains a brief overview of various features used for the recognition of handwritten characters and/or numerals.

2.1. A brief overview on feature based approaches of handwritten character and numeral recognition

Feature selection is a very important step in any pattern recognition task. Selecting the best feature-set for a pattern image often proves to be a major factor in identifying that image subsequently. Upon reviewing contemporary literature, we have discovered that there are hundreds of different feature extraction techniques in OCR research community. Some features are generic i.e. they can be used for any script along with any classifier whereas some of the features depend on the specificities of a particular script and/or can be used with only a particular classifier. Given the huge number of OCR related research works[58][59][2] being published every year, a comprehensive comparison of every method along with their corresponding performance evaluation is not within the scope of this document. Instead, a representative selection is made from the plethora of possible approaches to illustrate the different principles that can be used. A topography[58] of feature extraction methods for OCR related tasks is shown in Fig. 2.1.

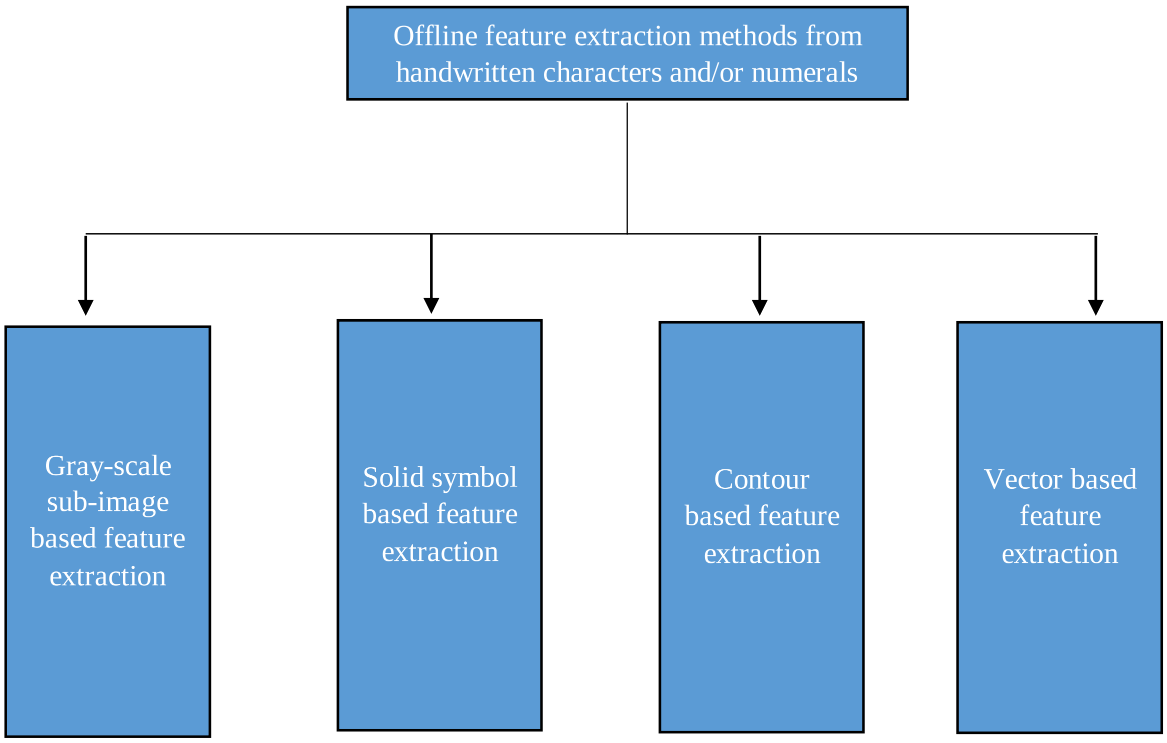

Offline feature extraction methods from handwritten characters and/or numerals
Gray-scale sub-image based feature extraction
Solid symbol based feature extraction
Contour based feature extraction
Vector based feature extraction

Fig. 2.1: *A topology of offline feature extraction techniques for handwritten characters and/or numerals*

2.1.1. Gray-scale sub-image based feature extraction

One of the most prominent sub-image based feature extraction techniques is *template matching technique*. In template matching, the character image itself is used as the feature vector. In recognition stage, a similarity measure between each candidate template T_i where $\mathrm{i} = 1$ to n and the character image I is computed. The character is assigned the class label k, if the template T_k has the highest similarity measure and if this similarity is above a pre-defined threshold. Else, the character remains unclassified. Generally distance metrics such as mean-square distance (D)[60] etc. are used as similarity metric for evaluating the quality of the feature vector in template matching techniques. Using character skeletons as templates, Bimbo et al.[61] have proposed a *deformable template matching* technique for recognition of numerals from low quality credit card slips. All of the approaches above are performed in a spatial domain[2]. Andrews et al.[62] have proposed a *unitary transform* approach where the feature vector is reduced in size significantly. They have proposed a unitary KL space transform where pixels are sorted based on variance and only the pixels with highest variance is used as feature vector. Other unitary transforms such as cosine transform, slant transform etc. can also be used[63] for space transformation and the compressed image can be used subsequently for classification purposes. The same technique can be applied in a transformed space using Haar or Hadamard transforms[63] also.

The OCR system proposed by Calera[64] introduced a *zoning* technique for character recognition. In this approach, a $\mathrm{n} \times \mathrm{m}$ grid is superimposed on the character image and each cell of the grid is taken as a separate feature. As the feature vector comes together, a similarity or dissimilarity metric can be used to assign class labels to the pattern image. An example of static zoning grids superimposed on handwritten character images is shown in Fig. 2.2.

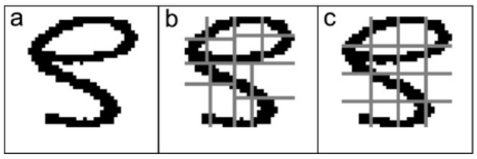


Fig 2.2: *A demonstration of zoning techniques. (a) Input handwritten character image (b) CG based zoning gridlines (c) equal partitioning based zoning gridlines*

[Image Source: Sarkhel et al.[16]]

Hu's *geometric moment invariants*[65] have been extensively[66][67] used in different pattern recognition tasks. Rotation invariant *Zernlike moments*[68] have also been used for recognition of binary solid symbols[66][69]. Khotanzad et al.[69] have proposed using the amplitudes of Zernlike moments as features. These moment based features can be made scale and translation invariant also[66]. While the first-order regular moments can be used to find the image centre, the zeroth order central moment gives a size estimate of the pattern image. Examples of images derived from *Zernlike moments* is shown in Fig. 2.3.

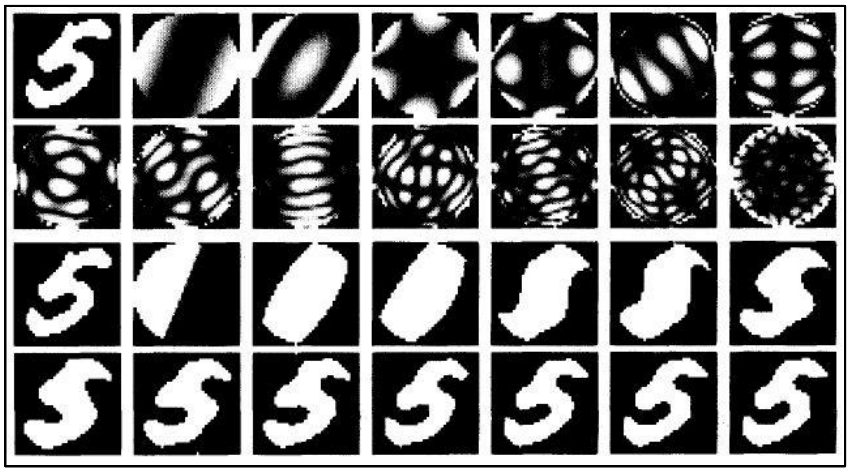


Fig. 2.3: *Images derived from Zernike moments*[58][68].
Rows 1-2: Input image of the digit "5" and contributions from the Zernike moments of order 1 to 13. The images are equalized histograms for highlighting the details.
Rows 3-4: Input image of digit "5" and images reconstructed from the Zernike moments of order up to 1 to 13 respectively.
[Image source: Trier et al. [58]]

2.1.2. Solid symbol based feature extraction from binary images

In binary template matching, several different similarity metrics, besides *mean square distance* (D) has been proposed. Tubbs[70] has suggested using *Jaccard distance* or *Yule distance* for measuring the similarity between the character template and pattern image.

The template matching procedure can also be repeated using vertical and horizontal *projection histograms*[71] of the character image. Using a fixed number of bins on each axis and normalizing them by the total number of print pixels in the character image, this feature can be made scale independent. As discussed before, *projection histograms* can be used in a similar fashion to match templates, using a similarity metric. Examples of vertical and horizontal projection histograms of the input image "5" is shown in Fig. 2.4.

Das et al.[27] have proposed a shadow based feature set where the features are computed by computing lengths of the projections on each octant of the tight enclosing box of the character image. Performance of this feature set has been evaluated[29][18][72] on some databases of handwritten characters and it is very promising. Basu et al.[73] have proposed a powerful convex hull based feature set for recognition of multilingual, multi-oriented handwritten characters. A graphical representation of the proposed feature extraction technique is shown in Fig. 2.5.

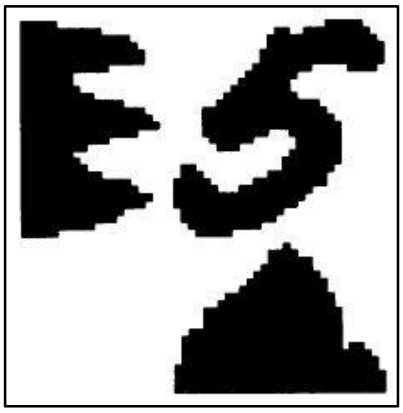
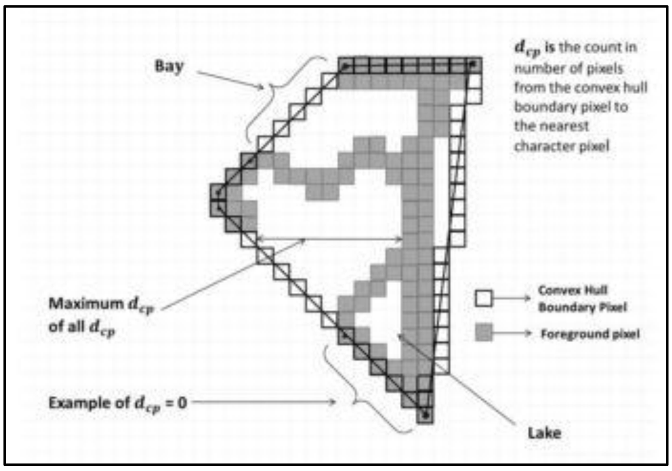


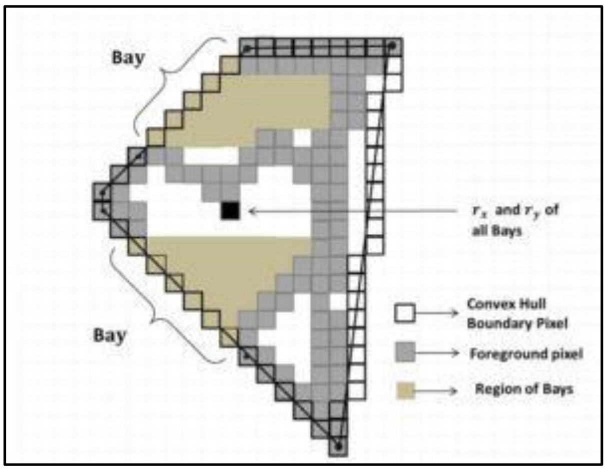
Fig. 2.4: *Vertical and horizontal projection of the input image of handwritten digit '5'*[58]

[Image source: Trier et al. [58]]

As binary images are special cases of gray-scale images discussed above, Hu's *geometric moment invariants*[65] and *Zernlike moments*[68] can be used for binary images also. However, in the binary case the contrast invariants are not needed.


Bay
d_{cp} is the count in number of pixels from the convex hull boundary pixel to the nearest character pixel
Maximum d_{cp} of all d_{cp}
Convex Hull Boundary Pixel
Foreground pixel
Example of d_{cp} = 0
Lake

(a)



(b)

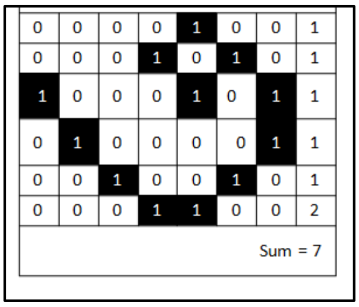
Fig 2.5: *Extraction of convex hull based features from handwritten characters (a) the features are extracted from left to right, based on d_{cp}, along with lake features, (b) the coordinates of centre of gravity (r_x , r_y) of a hypothetical region created by joining the bay pixels up to the nearest character pixels along left to right direction d_{cp}*

[Image Source: Basu et al.[73]]

2.1.3. Contour based feature extraction

As defined by Trier et al.[58], the closed outer contour curve of a character is "*a closed piecewise linear curve that passes through the centres of all the pixels which are four-connected to the outside background and no other pixels*". The pixels are visited in clockwise or counter-clockwise order and the edge pixel may be visited twice at locations where the object is only one-pixel wide. Each line segment comprising of the contour is a straight line between two eight-connected neighbouring pixels. If the contour curve is approximated by a parametric expression, coefficients of that expression can be used as features.

Kimura et al.[74] have suggested using *contour profiles* for feature extraction from handwritten characters. The motivation behind using *contour profiles* is approximating each half of the contour using the spatial variables x or y. Although Kimura et al.[74] have used outer vertical profiles, any one between vertical or horizontal profiles or outer or inner profiles may be used for subsequent feature extraction. Das et al.[27] have suggested using a quad-tree based hierarchical longest run based feature for this purpose. The longest run of black pixels are computed across four axes. Sarkhel et al.[17] have shown significantly improved result by applying zoning techniques with longest run based features on a dataset of handwritten characters. Fig. 2.6 shows an example of the longest run based features extracted from the handwritten characters.

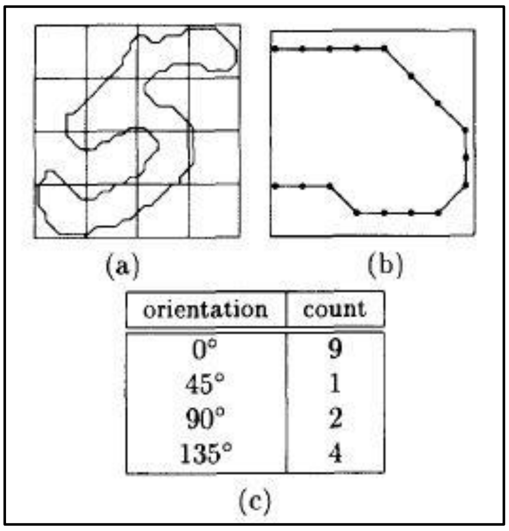


0	0	0	0	1	0	0	1
0	0	0	1	0	1	0	1
1	0	0	0	1	0	1	1
0	1	0	0	0	0	1	1
0	0	1	0	0	1	0	1
0	0	0	1	1	0	0	2
							Sum = 7

Fig. 2.6: *An example of longest run based feature extracted from horizontal axis*

[Image source: Sarkhel et al.[16]]

Kimura et al.[74] have used *zoning* techniques on contour curves. For each zone, the contour curve between two neighbouring pixels has been clustered into four groups based on their orientations. A graphical overview of *zoning* techniques applied over contour curves is shown in Fig. 2.7. Takahashi[75] has also used slice zones for extracting orientation histograms. He has used both inner and outer contour profiles for feature extraction. The curvature value, contour tangent and the point's zonal position are extracted as feature values.



orientation	count
0°	9
45°	1
90°	2
135°	4

Fig. 2.7: *Zoning of contour curves(a)* 4 × 4 *grid superimposed on character; (b) close-up of the upper right corner zone; (c) histogram of orientations for this zone.*
[Image source: Trier et al. [58]]

*Sekita et a*l.[76] have proposed detecting the high curvature points on a character image and approximate the curve between them using a *spline curve* approximation. Coordinates of both of the breakpoints and the parameters of the *spline curve* are used as features. Taxt et al.[77] have also proposed a smoothed *spline curve approximation* for outer contour of handwritten character images. The smoothed spline curve is divided into M equal parts. For each part, average curvature distance and mean distance of contour curve points from N equally spaced points are used as features. This feature is already rotation and translation invariant. If normalized[58], it can be size invariant also.

Kuhl et al.[78] have proposed using *Elliptic Fourier Curve* to approximate the closed contour of handwritten character image. Coefficients of the curve can be used as features. After normalization[58] it can be made size invariant also. Lin et al.[79] have extended this work and derived features which are also rotation invariant.

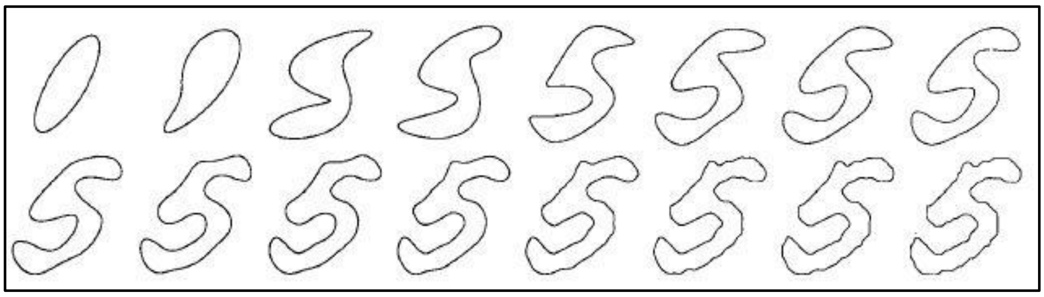


Fig. 2.8: *'5' reconstructed by elliptic Fourier descriptors of orders up to 1, 2, .. 10; 15, 20, 30, 40, 50 and 100, respectively.*
[Image source: Trier et al.[58]]

2.1.4. Vector based feature extraction

Character skeletons can be obtained by thinning the raster representation of the character image. Once the character skeleton is obtained, a *character graph* can be formed by approximating the line segments as edges and junction points as nodes of the graph. The curved parts of the skeleton can be approximated as arcs. Wang et al.[80][81] have proposed a method to derive *character graphs* from gray scale images of characters. Each character image is conceived as a 3D surface, where the gray-level of each pixel is mapped to the z coordinate. Clearly, the value of z ranges from 0 to 255. Topographic analysis is performed on the character image to find ridges and saddles, which is in turn used to find the character graph with line segments, arcs and junction points. This method is used when even the best possible binarization method fails to preserve the shape details of the character.

Deformable template matching for character skeletons has also been proposed by researchers[82][83][84]. Wakahara[83][84] proposed to introduce deformation to each template a number of small steps, called *local affine transforms* (LAT) for matching the candidate input pattern with the template.

Discrete features can also be extracted from thinned character images. Researchers like Kundu et al.[85], Ramesh[86], Basu et al.[73] have proposed using features like the number of loops, the number of T-joints, the number of X-joints, the number of bend points, convex-hull based features, centroid based features, width-to-height ratio of the bounding box, presence of an isolated dot, total number of endpoints and number of endpoints in each of the four directions, the number of semi-circles in each of these four directions, number of crossings with vertical and horizontal axes etc. Samples of a few such discrete features extracted from character skeletons is shown in Fig. 2.9.

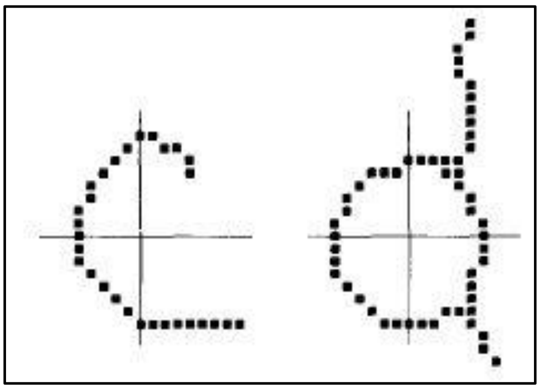


Fig. 2.9: *Thinned letters "c" and "d". Vertical and horizontal axes are placed at the centre of gravity. "c" and "d" both have one semicircle in the West direction, but none in the other directions. "c" has one horizontal crossing and two vertical crossings.*
[Image source: Kundu et al.[85]]

Holbaek-Hanssen et al.[87] proposed a *zoning* technique where they divided the character skeleton into zones. The normalized length of line segments, presence or absence of junction points in each zone etc. are used as features. The final feature vector comprises of the vectors from each zone. Although these features are size invariant, they are not rotation invariant. As character graphs can be traversed a closed curve, Taxt et al.[77] have extended *Elliptic Fourier Descriptors*[78] for representing the character skeleton. They have found out that for character graphs with two line endings, no junctions and no loops, some of the descriptors will be zero, while for graphs with junctions or loops, all descriptors will be nonzero. Characteristics for size and rotation invariance have also been found.

From the brief overview of the feature based approaches of handwritten character and/or numeral recognition, it is clear that there are a plethora of approaches to extract meaningful features from character images. Most of the times, the success of the OCR system depends on these handcrafted features and the classifier model being employed. However, arriving at the most informative feature with least computational burden can be a challenging task. If the proposed OCR is tasked with the recognition of multilingual scripts, each with its own complexities and peculiarities, the job gets even harder. This was one of the motivations behind exploring the non-explicit feature based avenue of OCR research in our present work. An overview of similar approaches taken by contemporary OCR researchers is described in Chapter 3.

Chapter 3

An Artificial Neural Network based approach towards OCR systems

The performance of an OCR system depends on the power of the prediction model, working behind the scenes. Prowess of the prediction model however, depends not only on the classifier used but on how the pattern images are being described while training the model also. In *feature based approaches* (see Fig. 1.2), explicit features are used to describe pattern images. As discussed in Chapter 2, there are a plethora of ways to extract features from isolated images of handwritten characters and/or numerals. Deciding on the feature set ultimately depends on the particularities of the script and the experience of the researcher. Evidently, if the task at hand is to propose a universal OCR for recognizing handwritten characters and/or numerals of multilingual Indian scripts, the challenge of proposing a suitable feature set increases many folds. This motivated us to explore the avenue of *non-explicit feature based approach* (see Fig. 1.2) in our present work. *Non-explicit feature based approach* includes a set of methods in which instead of deriving explicit features from the pattern images, raw image or pixel data is directly fed into the classifier which trains itself to minimize a squared error against a set of given image labels. The trained system can thus be used for classifying unknown patterns.

One of the most popular approaches towards *non-explicit feature based OCRs* (see Fig. 1.2) is due to using Hidden Markov Models or Markov Models of first or second order. Researchers like Kundu et al.[85], Hull et al.[88], Britto et al.[33] etc. have proposed HMM or Markov model based approaches towards recognition of handwritten or printed texts. Statistically derived parameters play a vital role in this approach. However, Markov networks need a large number of training samples for estimating the probabilistic parameters up to a reliable degree of certainty. Similar approaches have been taken by researchers like Bhowmik et al.[89], Al-Muhtaseb et al.[90], Bhattacharya et al.[35], Parui et al.[35] etc. for recognition of handwritten Oriya, Arabic, Devnagari and Bangla characters and numerals respectively. Although promising, one of the main disadvantages of this approach is the long training time and huge number of training samples required for setting up the parameters of the model to a certain degree of confidence.

Another popular example of *non-explicit feature based approach* is artificial neural networks. Several works[31][32][91][92] on artificial neural network based OCR system have been published in recent years. Some of the significant works using this approach was done by Lecun et al.[93][31], Cao et al.[94] and Takahashi et al.[75]. Artificial neural networks based approach for the recognition of handwritten and/or printed Indian characters and/or numerals have been demonstrated by researchers like Bhattacharya et al.[30], Dutta et al.[95], Ul-Hassan et al.[12] etc. Artificial neural networks can take as input explicit features[96][94] extracted from the input images or a scaled or subsampled input image[93] itself. In the first case, the neural network can be viewed as a *pure classifier*, constructing some complicated decision boundaries, whereas in the second case it can be viewed as a *combined feature extractor and classifier*.

The objective of our present work is to propose a fast, universal OCR system for the recognition of handwritten, multi-lingual, complex Indian characters and numerals. Being motivated by the simplicity and elegance of multi-layer feed forward neural networks to both extract and classify unknown pattern images with noticeable superiority[93][31][97][98] over many of its contemporaries, requiring lesser amount of training samples and its ability to provide superior performance even on augmented datasets[31], we have decided to take an artificial neural network based approach in our experimental setup. A multi-layer convolutional neural network based architecture is proposed in our present work for both extraction and classification of handwritten, multilingual, complex Indian characters and numerals. The motivation behind using a deep architecture is described in the following section. In the following section a brief overview on the workings of a multi-layer neural network is provided. Once the basic workings of a multi-layer perceptron is established, we move on to describing deeper networks and convolutional neural networks in general in the next chapter.

3.1. A brief overview on Artificial Neural Networks

The long course of evolution has provided human brain with several desirable properties[99] such as large-scale parallelism, adaptability, generalization ability, learning ability, inherent coherent information processing, ability to infer from contextual information, fault tolerance etc. The inability of centralized von-Neumann architecture[100] or even modern general purpose parallel computers to achieve these properties is what led to the development of Artificial Neural Networks (ANN). Inspired by biological neural networks, ANNs are essentially a network of parallel computing systems consisting of a large number of general or special purpose processors with dense interconnections. It attempts to use some organizational principles believed to be used in the human brain for reaching the superior performance it was set out to achieve.

3.1.1. A brief timeline of research in Artificial Neural Networks

The timeline of ANN research can be categorized into three broad periods. The first period of research activity which started in the 1940s was due to McCulloch and Pitts'[101] pioneering work. The second period can be marked by Rosenblatt's perceptron convergence theorem and the notable work by Minsky and Papert[102]. Discouraged by the findings of Minsky and Papert[102] regarding the limitation of a simple perceptron, a lull ensued in neural network research which lasted almost 20 years. Since the early 1980s, ANNs have received considerable renewed interest. The major developments behind this resurgence include Hopfield's energy approach[103] and the introduction of back-propagation learning algorithm for multilayer perceptrons, proposed by Werbos[104] and popularized by Rumelhart et al.[105]. Since then ANNs have been extensively used for wide array of challenging tasks[99], such as classification, approximation, clustering, prediction, optimization, content retrieval etc.

3.1.2. How does an ANN work?

In general, ANNs try to minimize a cost function (e.g. least-squared error) by adjusting the connection weights over multiple iterations i.e. they try to learn the weights over multiple epochs. The learning algorithm can be supervised, unsupervised or hybrid, depending on the task at hand. A taxonomy of ANNs based on its architecture and the learning algorithm used is shown in Fig. 3.3.

Although the first ever perceptron proposed by McCulloch and Pitts was single layer (shown in Fig. 3.1), as the complexity of the tasks increased, researchers felt the need of increasing the depth of these networks. Hence multi-layer neural networks came into the picture.

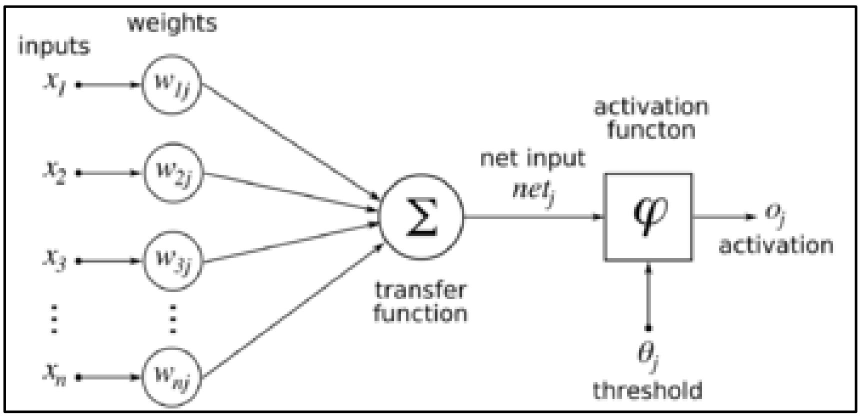


Fig. 3.1: *McCulloch-Pitts model of single layer perceptron*

[Image Source: [106]]

Typically, a standard N-layer feed-forward network consists of an input layer, (N-1) hidden layers, and an output layer of units successively connected, either fully or locally, in a feed-forward fashion with no connections between units in the same layer and no feedback connections between layers. The most popular class of multi-layer feed-forward networks is multi-layer perceptrons in which each computational unit implements an *activation function* which typically is either a thresholding function or the sigmoid function. According to the universal approximation theory by Hornik et al.[107], multi-layer perceptrons can form arbitrarily complex decision boundaries and represent any boolean function. The development of the *backpropagation*[104][105] learning algorithm for a multi-layer perceptron has made the task of training these networks much easier.

An example of a three-layer feed-forward neural network is shown in Fig. 3.2. Each unit in the first hidden layer helps form a *hyperplane*[99] in the input space. *Hyperplanes* can be used to approximate decision boundaries between pattern classes. Subsequently, each unit in the second hidden layer forms a *hyperregion*[99] from the outputs of the previous layer units. A decision region can be obtained by performing an AND operation on the *hyperplanes* obtained from the previous layer. Finally, the output layer combines the decision regions made by the units in the second hidden layer by performing logical OR operations and assigns the class labels to unknown pattern images. This whole process is completed by the predefined learning algorithm over multiple epochs until some termination criteria is met.

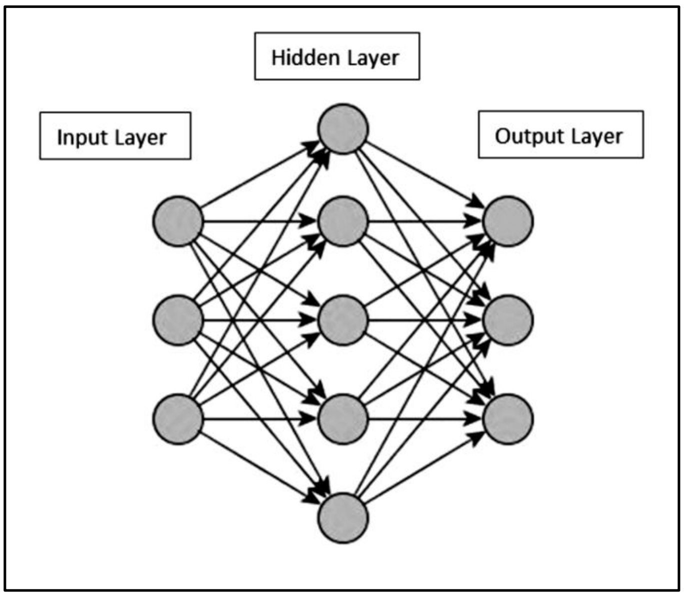


Fig. 3.2: *A three-layer feed-forward neural network*

However, although the power of the ANN increase with increasing number of layers and number of neuron per layers, as the number of neurons increase and networks become deeper, some road blocks have come into appearance. A brief account of such difficulties in using deep learning models in is discussed in the following section.

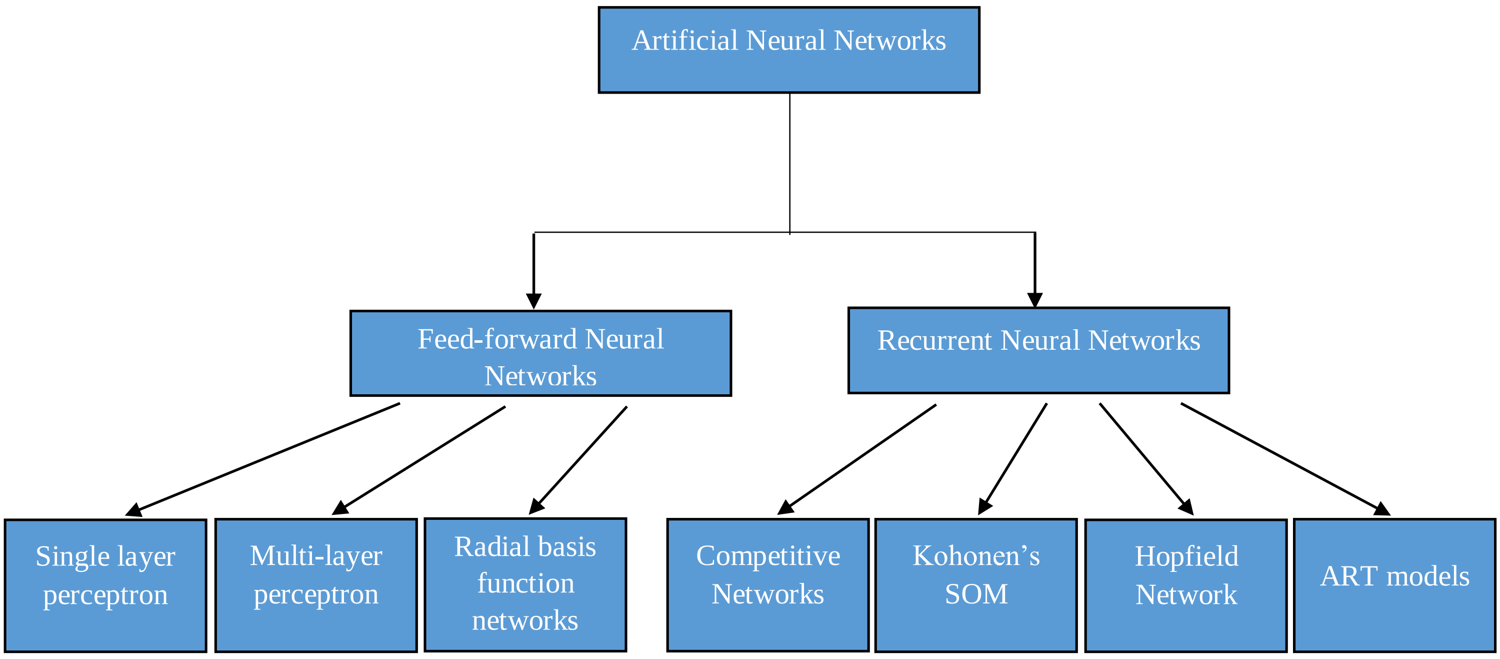

Artificial Neural Networks
Feed-forward Neural Networks
Recurrent Neural Networks
Single layer perceptron
Multi-layer perceptron
Radial basis function networks
Competitive Networks
Kohonen's SOM
Hopfield Network
ART models

Fig. 3.3: *A topology of Artificial Neural networks*

3.2. Problems in training a multi-layer neural network

The concept of multilayer neural networks as well as the general idea of deep architectures have existed since the *backpropagation algorithm*[104] was proposed in the late 1980s. As the tasks got more complex, ANN researchers felt the necessity of increasing the depth of the artificial neural networks for better performance more than ever. However, as the number of layers and number of neurons per layer increased, several unforeseen problems came along. Some of the various impediments in using DNNs are discussed briefly in this section.

3.2.1. The vanishing gradient problem

For understanding the vanishing gradient problem, a little background on the *backpropagation algorithm* is needed. The idea of backpropagation is explained on a simplified network in shown in Fig. 3.4 which introduces the concept of backpropagation between 2 layers. The same idea, however, can be generalized and extended for larger networks.

Let's suppose that the error measure used by the simplified network shown in Fig. 3.4 is least-squared-error, which essentially is the sum of the square of the difference between the true label and the label assigned by the ANN for all pattern images. Therefore, after adjusting for the sign and denominator we have the error term,

$$E = -\frac{1}{2} \sum_{j \in output\ layer} (y_j - t_j)^2 \tag{3.1}$$

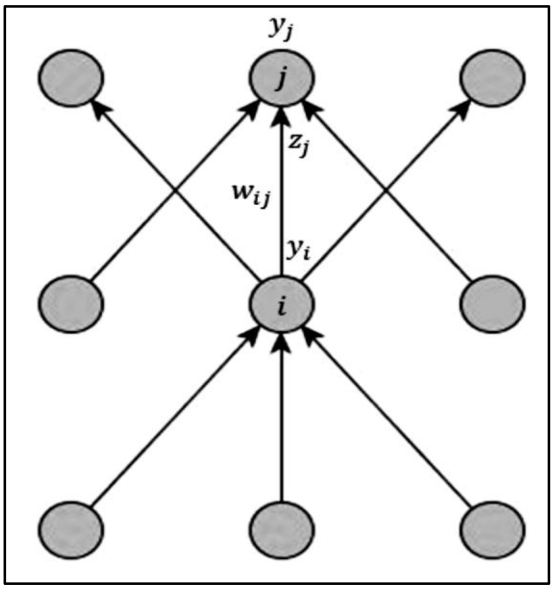


Fig. 3.4: *A simplified Neural Network alike structure for illustrating backpropagation*

Let, y_i and z_i refer to the input and output of the node i respectively. Then the error gradient is calculated as:

$$\frac{\partial E}{\partial y_i} = \sum_j \frac{\partial E}{\partial z_j} \frac{\partial z_j}{\partial y_j} \tag{3.2}$$

Since $z_j = w^T y$ and if we consider w_{ij} as the weight between unit i and j, it can be written as the following:

$$\frac{\partial E}{\partial y_i} = \sum_j w_{ij} \frac{\partial E}{\partial z_j} \tag{3.3}$$

The equation 3.3 provides us with the error derivative with respect to the node i. This can be backpropagated to the preceding layer in a similar fashion. The weight term w_{ij} between i and j is updated following the direction of the steepest gradient ascent, as follows:

$$w_{ij} = w_{ij} + \eta \frac{\partial E}{\partial w_{ij}} \tag{3.4}$$

Where $\frac{\partial E}{\partial w_{ij}}$ denotes the amount of change needed to be done to the term w_{ij} and it can easily computed as following:

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial z_j}{\partial w_{ij}} \frac{\partial E}{\partial z_j} = y_i \frac{\partial E}{\partial z_j} \tag{3.5}$$

The equation 3.4 can therefore be rewritten as:

$$w_{ij} = w_{ij} + \eta \times y_i \frac{\partial E}{\partial z_j} \tag{3.6}$$

Now, as the basic workings of *backpropagation algorithm* is established, let us turn our focus to the main topic of this section i.e. the *vanishing gradient problem.*

The vanishing gradient problem refers to the phenomenon of diminishing the error gradient as it propagates down the network when using logistic activation functions. This results in the updates to the weights of the lower layers of the neural network to be extremely small. Although this problem was initially discovered in recurrent neural networks[108], it has since been generalized to feed forward models as well.

To demonstrate the vanishing gradient problem, a simplistic example as described in the following section. Let's consider a simplified multilayer feed forward neural network (shown in Fig. 3.5) structure with 4 hidden layers with 1 hidden unit each with a logistic activation function.

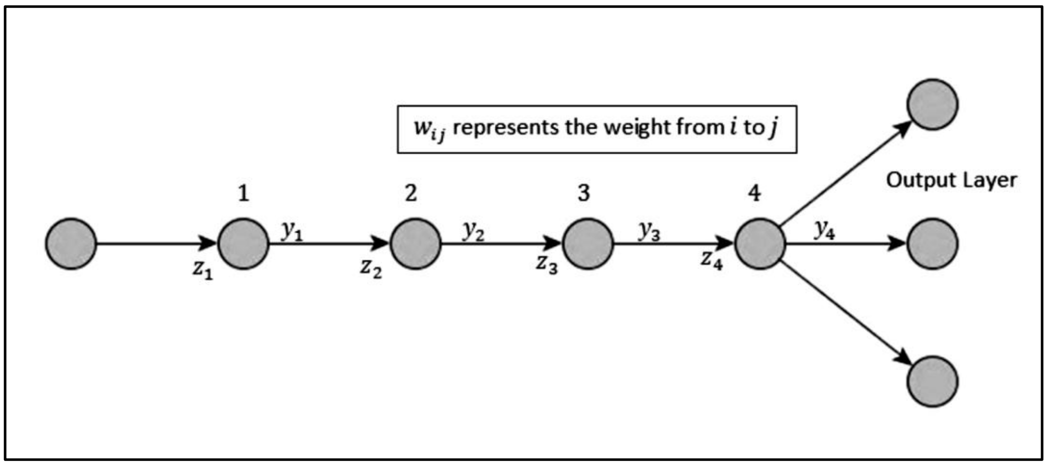


Fig. 3.5: *A simplified Multi-layer Feed-forward Neural Network structure for illustrating the Vanishing Gradient problem*

Using the least-squared-error term, the error E can be computed as following:

$$E = -\frac{1}{2} \sum_{j \in output\ layer} \left(y_j - t_j\right)^2$$

Clearly, following the equation 3.2,

$$\frac{\partial E}{\partial z_4} = \frac{\partial E}{\partial y_4}\frac{\partial y_4}{\partial z_4} = \left((y_4 - t_4) \frac{\partial y_4}{\partial z_4} \right) \tag{3.7}$$

Propagating the error gradient back to layer 3, we get:

$$\frac{\partial E}{\partial z_3} = \frac{\partial E}{\partial z_4}\frac{\partial z_4}{\partial y_3} = w_{34}\frac{\partial E}{\partial z_4} \tag{3.8}$$

Extending the error gradient to layer 1 as equation 3.3, we get the following:

$$\frac{\partial E}{\partial y_1} = w_{12}\frac{\partial y_1}{\partial z_1} w_{23}\frac{\partial y_2}{\partial z_2} w_{34}\frac{\partial y_3}{\partial z_3}\frac{\partial y_4}{\partial z_4}\frac{\partial E}{\partial y_4} \tag{3.9}$$

The logistic or sigmoid activation function was introduced due to the inability of threshold neurons, used in initial simple perceptrons[101] to perform complex tasks. During the 90s, it was alongside the hyperbolic tangent, one of the most popular activation functions for neural networks. A graphical representation of the logistic activation function is shown in Fig. 3.6. The function itself is defined as following:

$$f(z) = \frac{1}{1 + e^{-z}} \qquad (3.10)$$

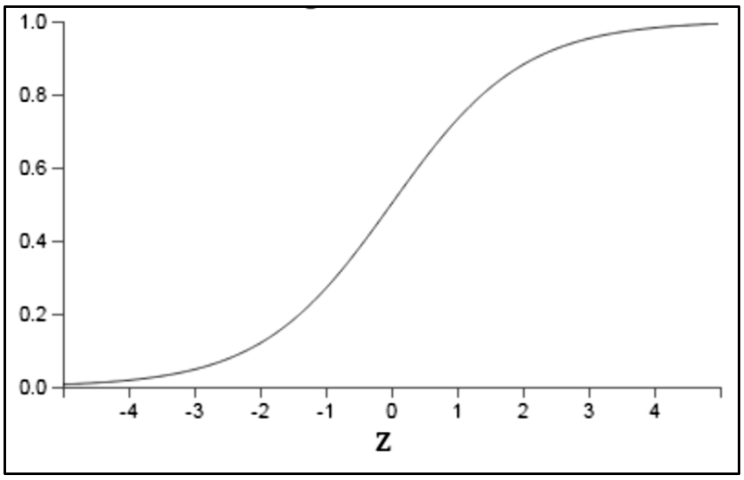


Fig. 3.6: *A graphical representation of logical activation function (f(z))*

The value of $\frac{\partial y_i}{\partial z_i}$ in the equation 3.9 is basically the derivative of the activation function used in each node of the neural network and varies when we use different activation functions. One of the most popular activation function during the 90's was the logistic function (defined by equation 3.10). A graphical representation of the derivative ($f'(x)$) of the logistic function is given in Fig. 3.7. It is clear from Fig. 3.7 that the max value of derivative function $f'(x)$ is at 0, where its value is 0.25. However, the standard technique for initializing weights in a neural network is to randomly initialize them with a mean of 0 and a standard deviation of 1. Therefore, generally $|w_{ij}| < 1$. Henceforth, the value of the terms of form $|w_{ij}\frac{\partial y_i}{\partial z_i}| < 0.25$ in equation 3.9. Evidently, the further back we go through the network, the smaller the error gradient gets in value due to the product of many such terms. This results in the weight updates being smaller or closer to zero for the nodes near the input layer. This is known as the *vanishing gradient problem*. This problem can be alleviated by using activation functions like Rectified Linear Units in deeper neural networks.

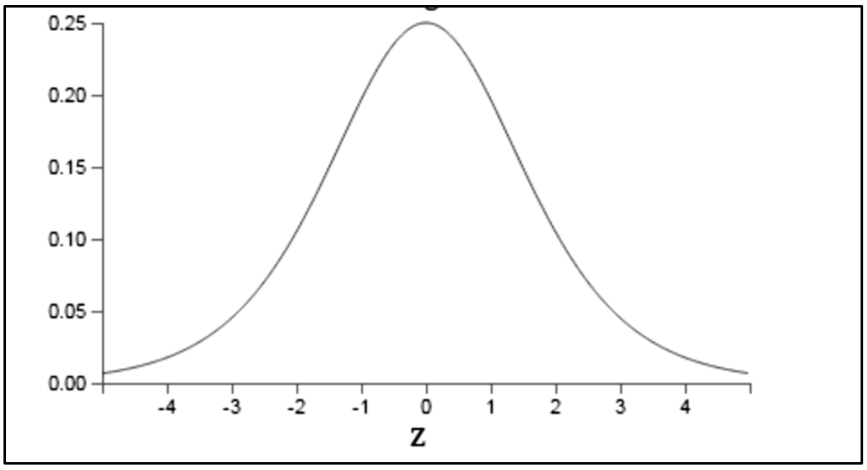


Fig. 3.7: *Derivative* ($f'(z)$) *of the logistic activation function* ($f(z)$)

3.2.2. Lack of the necessary hardware support for running Deep Neural Networks

Another notable bottleneck for deploying large neural network models is the lack of proper hardware support. In the 90s, it was impossible for large neural networks to be deployed due to the inability to provide the necessary computation power required.

Traditional CPU based computing is inadequate for processing the huge amount of network parameters, performing bookkeeping jobs and performing huge number of computations required during training. Following the seminal work done by Alex Krizhevsky[109] using GPU based libraries for processing enormous amounts of data with vectorised versions of the learning algorithms has become quite norm. The parallel processing ability of GPU cores has made the learning process much faster than before.

Another major component of a successful deep learning model is a large training dataset. Current deep learning architectures use datasets of the order of tens of gigabytes. Enough RAM to hold this amount of data was not available until very recently. The quantum leap in the size of RAM memory has definitely helped modern computing systems in this regard.

Hence, harnessing the computing power of deep neural networks has become possible only recently.

3.3. Theoretical advantages of deep architectures

In this section, a motivating argument for exploring the avenue of learning algorithms for deep architectures in our present work is presented. This part of the thesis describes the motivation behind using the deep architectures and learning algorithms described in the later sections. The main objective of this section is to establish that some functions cannot be efficiently represented by architectures that are too shallow. The results suggest that it would be worthwhile to explore learning algorithms for deep architectures, which might be able to represent some functions otherwise not efficiently representable. Where simpler and shallower architectures fail to efficiently represent and hence to learn a task of interest, we can hope for learning algorithms that could set the parameters of a deep architecture for this task.

A function is said to be *compact* when it has few computational elements, i.e. few degrees of freedom that need to be tuned by learning over iterations. So for a fixed number of training examples, and short of other sources of knowledge injected in the learning algorithm, we would expect that compact representations of the *target function* i.e. the function we are trying to learn or approximate, would yield better generalization. More precisely, target functions which could have been compactly represented by a depth N architecture might require an exponential number of computational elements to be represented by a depth $N - 1$ architecture. Since the number of computational elements that can be afforded ultimately depends on the number of training examples available to tune them, the consequences are not just computational but also statistically quite significant i.e. when using an insufficiently deep architecture for representing a target function, poor generalization may happen.

Let us first consider the case of fixed-dimension inputs, where the computation performed by the machine can be represented by a directed acyclic graph. Each node of the computes a function on its inputs, each of which is the output of another node in the graph or one of the external inputs to the graph. For better understanding, the whole graph can be viewed as a *circuit* that computes a function applied to the external inputs. When the set of functions allowed for the computation nodes is limited to *logic gates*, such as $\{AND, OR, NOT\}$, this becomes a *boolean circuit*, or a *logic circuit*.

To formalize the notion of depth of architecture, the notion of a *set of computational elements* must be introduced first. One example is the set of computations that can be performed logic gates. Another is the set of computations that can be performed by an artificial neuron, based on the values of its connection weights. It is quite clear that a function can be expressed by the composition of computational elements from a given set. It is defined by a graph which formalizes this composition, with one node per computational element. Depth of the architecture refers to the depth of the graph, i.e. the length of the longest path from an input node to an output node. Now, when the set of computational elements is actually the set of computations an artificial neuron can perform, depth of the architecture corresponds to the number of layers in a neural network.

Let us explore the notion of depth with examples of architectures of different depths. Consider the function $f(x) = x * sin(a * x + b)$. It can be expressed as the composition of simple operations such as addition, subtraction, multiplication, and the sin operation, as illustrated in Figure 3.8. In the example, there would be a different node for the multiplication $a * x$ and for the final multiplication by x. Therefore, each node in the equivalent graph is associated with an output value obtained by applying some function on the input values some of which maybe the outputs of other nodes of the graph themselves. For example, in a logic circuit each node can compute a Boolean function taken from a small set of Boolean functions. The graph as a whole has input nodes and output nodes and computes a function from input to output. The *depth* of an architecture is the maximum length of a path from any input of the graph to any output of the graph, i.e. 4 in the case of the function $f(x)$ in Fig. 3.8.

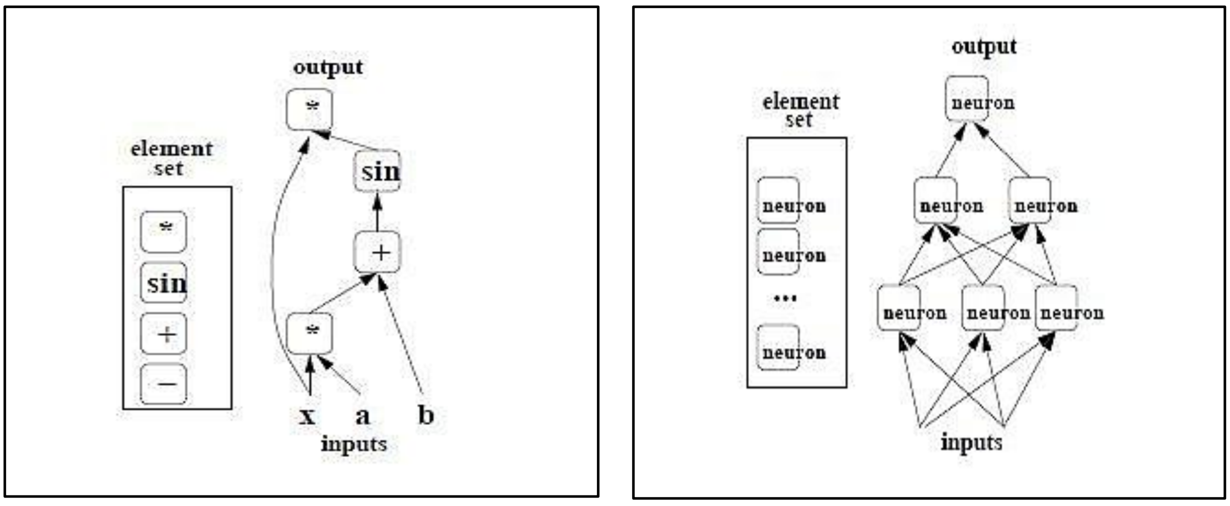


Fig. 3.8: *Examples of functions represented by a graph of computations, where each node is taken from some set of allowed computations. (a) The elements are $\{*, +, -, sin\} \cup R$. The architecture computes $x * sin(a * x + b)$ and has depth 4. (b) The elements are artificial neurons computing $f(x) = tanh(b + w'x)$; each element in the set has a different-valued (w, b) parameter pair. The architecture is a multi-layer neural network of depth 3.*
[Image source: Bengio[110]]

Although depth depends on the choice of the set of allowed computations for each element, graphs associated with one set can often be converted to a graph associated with another using any transformation technique which multiplies depth. Theoretical results[111] suggest that it is not the absolute number of levels which matters, but the relative number of levels which are required to represent efficiently the target function.

3.3.1. Formal argument: better computational complexity

One of the most formal arguments about the power of deep architectures can be made by investigating the computational complexity of circuits. The basic conclusion that these results suggest is that *when a function can be compactly represented by a deep architecture, it might need a very large architecture to be represented by an insufficiently deep one.*

A two-layer circuit of logic gates can be used to represent any boolean function[111][112]. To understand the limitations of shallow architectures, we first need to consider that with two-layer logical circuits, most boolean functions require an *exponential* (with respect to input size) number of logic gates[113] to be represented. More interestingly, there are functions computable with a polynomial-size logic gates circuit of depth N which will require an exponential number of logic gates when restricted to a depth of N − 1[114]. This can be proved based on earlier results[115] which show that d-bit *parity circuits of depth 2 have exponential size*. The d-bit *parity function* is defined as usual:

$$(b_1, b_2, \dots, b_d) \in \{0,1\}^d \rightarrow \begin{cases} 1 & if \sum_{i=1}^{d} b_i \\ 0 & otherwise \end{cases} \tag{3.11}$$

Many of the results for boolean circuits can be generalized to architectures whose computational elements are *linear threshold* units (also known as artificial neurons[101]), which compute:

$$f(x) = \begin{cases} 1 & if\ wx + b \geq 0 \\ 0 & otherwise \end{cases} \tag{3.12}$$

with parameters w and b. The *fan-in* of a circuit is the maximum number of inputs of a particular element. Circuits are often organized in layers, like multi-layer neural networks, where elements in a layer only take their input from elements in the previous layers. The *size* of a circuit is the number of its computational elements (excluding input elements, which do not perform any computation). The following theorem proposed by Hastad et al.[114] regarding *monotone weighted threshold circuits* which are essentially multi-layer neural networks with linear threshold units and positive weights should be mentioned when trying to represent a function compactly representable with a depth k circuit:

Theorem 3.1. *A monotone weighted threshold circuit of depth* $k-1$ *computing a function* $f_k \in F_{k,N}$ *has size at least* 2^{cN} *for some constant* $c > 0$ *and* $N > N_0$.

The class of functions $F_{k,N}$ contains functions with N^{2k-2} inputs, defined by a depth k circuit that is a tree. At the root of the tree there are function values whereas at the leaves there are unnegated input variables. The ith level from the bottom consists of AND gates when i is even and OR gates when i is odd. The *fan-in* at the top and bottom level is N and at all other levels it is N^2.

The above results however, do not prove that other classes of functions like those which needs to be learnt for performing many intelligent tasks require deep architectures, nor that these demonstrated limitations apply to other types of circuits.

However, these theoretical results beg the question whether the 1, 2 and 3 layer architectures or the so called shallow architectures which are typically found in most learning algorithms inadequate for the representation of more complicated functions required for more intelligent tasks. Relevant results such as the above theorem also suggest that *there might be no universally right depth*: each function might require a particular minimum depth given a set of computational elements. Therefore we should strive to develop learning algorithms that use the data to determine the depth of the final architecture.

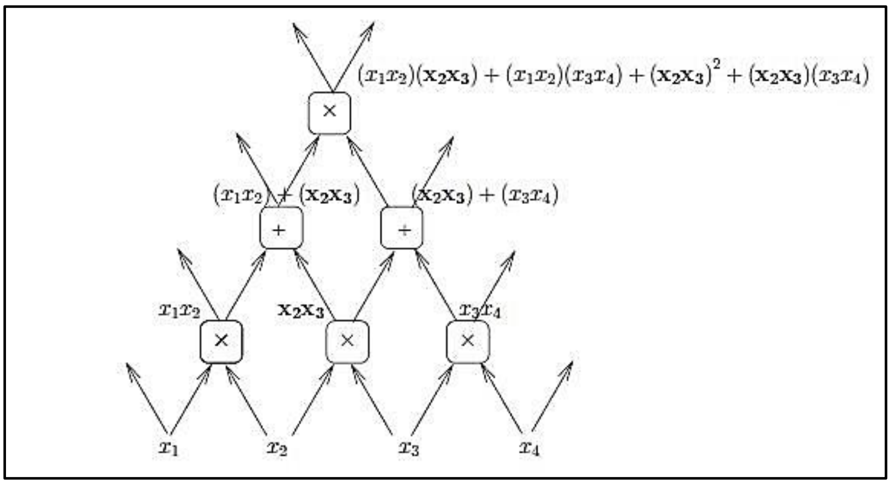


Fig. 3.9: *Example of a polynomial circuit illustrating the factorization enjoyed by a deep architecture*
[Image source: Bengio[110]]

3.3.2. Informal arguments

Depth of the architecture is generally connected to the notion of highly-varying functions. It can be argued that, deep architectures can compactly represent highly-varying functions which could have otherwise be represented by a very large, shallow architecture. A function is said to be *highly-varying* when a piecewise approximation of that function would require a large number of computational elements. A deep architecture is a composition of many operations, and it could in any case be represented by a possibly very large 2 layer architecture; whereas, the composition of computational units in a small but deep circuit can actually be seen as an efficient factorization of a large but shallow circuit. Reorganizing the way in which computational units are composed can have a drastic effect on the efficiency of representation size.

For example, imagine a depth 2k representation of polynomials where odd layers implement products and even layers implement sums. This architecture can be seen as a particularly efficient factorization, which when expanded into a depth 2 architecture such as a sum of products, might require a huge number of terms in the sum: consider a level 1 product (like x_2x_3 in Fig. 3.9) from the depth 2k architecture. It could occur many times as a factor in many terms of the depth 2 architecture. It can be seen in this example that deep architectures can be advantageous if some computations can be shared, in that case, the overall expression to be represented can be factored out, i.e., represented more compactly with a deep architecture.

To conclude, a number of computational complexity results strongly suggest that functions that can be compactly represented with a depth k architecture could require a very large number of elements in order to be represented by a shallower architecture. Since each element of the architecture might have to be selected, i.e., learned, using examples, these results suggest that depth of architecture can be very important from the point of view of statistical efficiency.

3.4. An overview of the development of Deep Learning based OCR systems

The explosive interest in Deep Learning is new but the foundations of the research dates back decades. The inspiration for deep learning or even deep architectures is drawn from the human brain itself which presumably works through multiple layers of abstraction. Human mind takes a hierarchical approach towards information processing. Realization of a similar idea in the area of machine learning has demonstrated a high degree of promise for researchers to achieve human like performance in important problems such as handwritten character recognition, speech recognition, objection recognition, natural image understanding and so on.

The first steps in realizing the importance of deep architectures was taken by Hubel et al.[116][117] when they tried to model the visual cortex of a cat. After discovering the presence of simple and complex cells, and that these cells were sensitive to certain areas of the visual field, they proposed an idea that a series of abstractions occur in a hierarchy to give rise to the idea of visual perception.

Within the next decade or so, the basic structure of the neural network was formed slowly. Although the findings by Minsky et al.[102] on the limitations of a single perceptron contributed to the slow growth to some extent, by the 1980s important contributions in neural networks began to emerge. The first computational models exploiting the local connectivities between neurons and on hierarchically organized transformations of the image was Fukushima's Neocognitron[118] as he recognized that when neurons with the same parameters are applied on patches of the previous layer at different locations, a form of translational invariance is obtained, coming very close to human like visual perception.

It was a deep architecture which shared many characteristics with the *Convolutional Neural Networks* proposed later put forth by Lecun et al.[93]. The idea of weight sharing was first introduced by the Neocognitron model. It was used for several applications including handwritten text recognition to prove its effectiveness. Another major contribution to training multilayer neural networks around this time was the formal introduction of the *backpropagation algorithm*[104][105] which although saw a temporary reduction in popularity during the 2000s, it is till this day is an integral part of training deep neural network systems.

The first concepts of *Convolutional Neural Network* was formally introduced in 1989 when Lecun et al. applied[93] embedded the *backpropagation algorithm*[104] with the Neocognitron model. The daunting success achieved by this model led to numerous successful applications of the CNN model, from areas like handwritten character recognition[93], object detection in natural scenes[109] to image, speech recognition and time series modelling[119].

The idea of *Long Short Term Memory (LSTM) Recurrent Neural Networks* were introduced in 1997 by Hochreiter et al.[120] which were effective in dealing with many problems that traditional RNNs could not deal with as well as leading to introduction of Deep RNNs[121] in later decades. Researchers like Ray et al.[122], Ul-Hassan et al.[12] later adopted this idea and proposed a Bi-directional LSTM network for offline handwritten text recognition successfully.

In the 2000s, the major innovation in the area of deep learning research were the introduction of *greedy unsupervised layerwise training*[123] of deep neural networks and *stacked denoising* *autoencoders*[124][125]. Autoencoder based unsupervised approach towards handwritten character and digit recognition has been taken by researchers like Hinton et al.[126], Schwenk et al.[127][128]. More recently Pal et al.[129][130], Wang et al.[131] etc. have used a *denoising stacked autoencoder* for recognizing handwritten Bangla characters, digits and Chinese handwritten legal texts respectively.

Reinvigorated by new findings, enriched by overzealous researchers, propelled by the advantages of GPU based vectorized libraries, the area of deep learning has started to see some real progress in the recent years, flooding with publications of innovative architectures, training methods and applications of the deep architectures.

Chapter 4

An Overview of Convolutional Neural Networks

Deep supervised neural networks are generally considered to be too difficult to train before the use of unsupervised pre-training. There is, however, one notable exception: *convolutional neural networks* or *convolutional nets* in short. Convolutional nets are inspired by the structure of the visual cortex, and in particular by the models of it proposed by Hubel and Wiesel[116][117]. They discovered that there are cells in the visual cortex which are sensitive to various small sub regions of the visual field. These sub regions which form a tiled arrangement encompassing the entire visual field, can be thought as local filters exploiting the spatially local correlation in an image.

The first computational models relying on these local connectivities between neurons and hierarchically organized transformations of the image are found in Fukushima's Neocognitron[118]. He realized that a form of translational invariance is obtained when neurons with the same parameters are applied on patches of the previous layer at different locations. Later, following up on this idea, LeCun designed and trained convolutional networks using the error gradient, obtaining state-of-the-art performance[132][93] on several pattern recognition tasks. Modern understanding of the physiology of the visual system[133] has been found to be consistent with the hierarchical information processing style found in convolutional networks which helps explain why even to this day, pattern recognition systems based on convolutional neural networks are among the best performing systems.

The convolution net, based on which the seminal handwritten character recognition system[132] was proposed, has served as a machine learning benchmark for many years. LeCun's convolutional neural networks are organized in layers of two types: convolutional layers and subsampling layers. Each layer has a *topographic structure*, i.e., each neuron is associated with a fixed two-dimensional position that corresponds to a location in the input image, along with a receptive field which is the region of the input image that influences the response of the neuron. At each location of each layer, there are a number of different neurons, each with its set of input weights, associated with neurons in a rectangular patch in the previous layer. The same set of weights, but a different input rectangular patch, are associated with neurons at different locations. These networks can be viewed as a combined feature extractor and classifier. The graphical abstract of LeNet 5, a convolutional net proposed by Lecun et al.[132] is shown in Fig. 4.1.

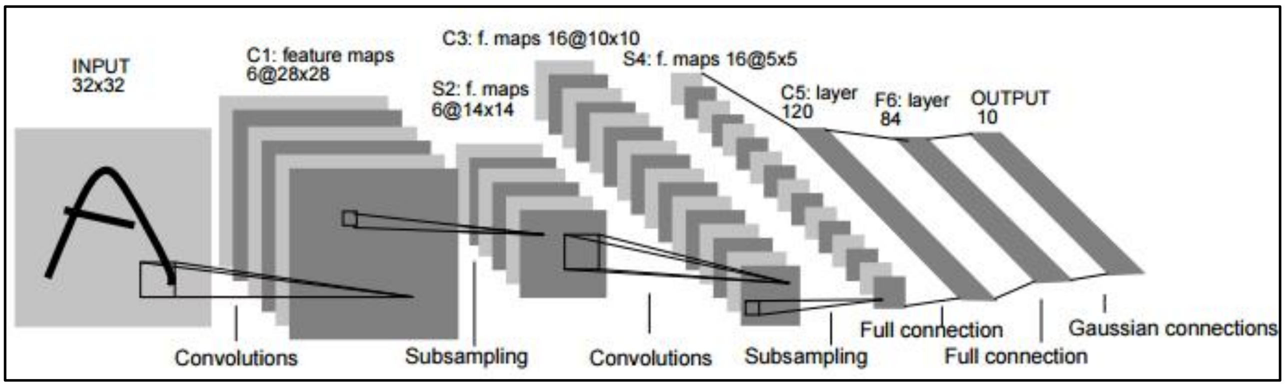


Fig. 4.1: *Graphical abstract of LeNet 5*
[Image source: Lecun et al.[132]]

As discussed earlier, *Convolutional Neural Networks* take a hierarchical approach towards feature extraction from a pattern image. The local kernels convolve on the pattern image extracting lower level features, whereas the subsampling layer performs an aggregation operation on these features and gives it a more compact form. A hierarchy of abstraction is thus formed which comes very close to visual perception of human brain. In the following section, an overview of the architecture of *Convolutional Neural Network* and some of its basic components are presented.

4.1. Architecture overview

As discussed in the previous chapter, Artificial Neural Networks (ANN) receive a single input vector and transform it through a series of hidden layers. Each hidden layer is made up of a set of neurons, where each neuron is fully connected to all of the neurons in the previous layer. The neurons in a single layer are completely independent and do not share any connections between each other. The last fully-connected layer is the output layer. It assigns class prediction scores in classification tasks.

Despite of its popularity, the problem with classical ANN is that they don't scale well to full images. For example, in the CMATERdb 3.1.1[134][135] dataset, images are only of size $32 \times 32 \times 3$ i.e. height of 32 pixels, width of 32 pixels and 3 colour channels. Hence, a single fully-connected neuron in a first hidden layer of a classical ANN, operating on this dataset would have to maintain and update a massive 32*32*3 = 3072 weights. This amount still seems manageable, but clearly this fully-connected structure does not scale to larger images. For example, an image of more respectable resolution, e.g. $100 \times 100 \times 3$, would lead to neurons that have 100*100*3 = 30,000 weights. Moreover, the architecture almost certainly will have several such neurons, so the number of parameters for the whole network would add up quickly. Clearly, this full connectivity is wasteful and the huge number of parameters would quickly lead to overfitting.

Convolutional Neural Networks (CNN) take advantage of the fact that the input consists of images and they constrain the architecture more intelligently. In particular, unlike a regular ANN, the layers of a Convolutional Net have neurons arranged in 3 dimensions, i.e. *width, height* and *depth*. It should be noted that the word *depth* here refers to the third dimension of an activation volume, not to the depth of a full Neural Network, which essentially refers to the number of layers in a network. For example, the input images in CMATERdb 3.1.1[134][135] are an input volume of activations, and the volume has dimensions $32 \times 32 \times 3$ i.e. width, height and depth respectively. In case of Convolutional Nets, the neurons in a layer will only be connected to a small subregion of the layer before it, as opposed to all of the neurons in a fully-connected manner. Moreover, the final output layer for this dataset would have dimensions of $1 \times 1 \times 10$, as by the end of the network architecture full image would be reduced to a single vector of prediction scores, one score for each class 0 to 9, arranged along the depth dimension. The conceptual difference between a classical ANN and a CNN is shown in a graphical representation in Fig. 4.2.

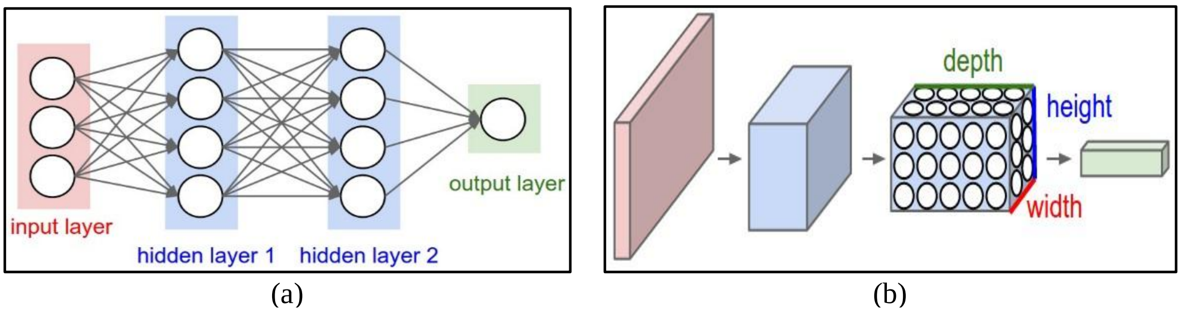


Fig. 4.2: *(a) A classical 3-layer ANN. (b) Neurons of a CNN arranged in three dimensions (width, height and depth), as visualized in one of the layers; the red input layer holds the image, so its dimensions are equal to the dimensions of the image, and the depth is 3 which signifies the red, green and blue colour channels*

[Image source: Karpathy[136]]

A brief discussion on each layer of a typical CNN architecture is presented in the following section.

4.2. Convolution layer

The convolution layer or CONV layer is the core building block of a CNN architecture that does most of the computational heavy lifting.

4.2.1. Overview and intuition

In this section, an overview of the CONV layer is introduced, without the complicated neuron analogies. Its parameters consist of a set of learnable filters. Each filter is spatially small, but extends through the full depth of the input volume. For example, a typical filter on a first layer of a CNN, tasked with the classification of the CMATERdb 3.1.1[134][135] of Cifar-10 images, might have the dimension of $5 \times 5 \times 3$ i.e. 5 pixels width and height, and as images have 3 colour channels, depth equals to 3). During the forward pass, each filter slides or *convolves* across the width and height of the input volume and compute dot products between the entries of the filter and the input at that position. As the filter slides over the width and height of the input volume, a 2-dimensional activation map that gives the responses of that filter at every spatial position, will be produced. Intuitively, the network will learn filters that activate when they see some type of visual feature such as an edge of some orientation or a blotch of some colour on the first layer, or eventually entire C-shape, X-junction, T-junction like patterns on higher layers of the network. Now, we will have an entire set of filters in each CONV layer, and each of them will produce a separate 2-dimensional activation map. These activation maps are stacked along the depth dimension and the output volume at the end of the CONV layer is produced.

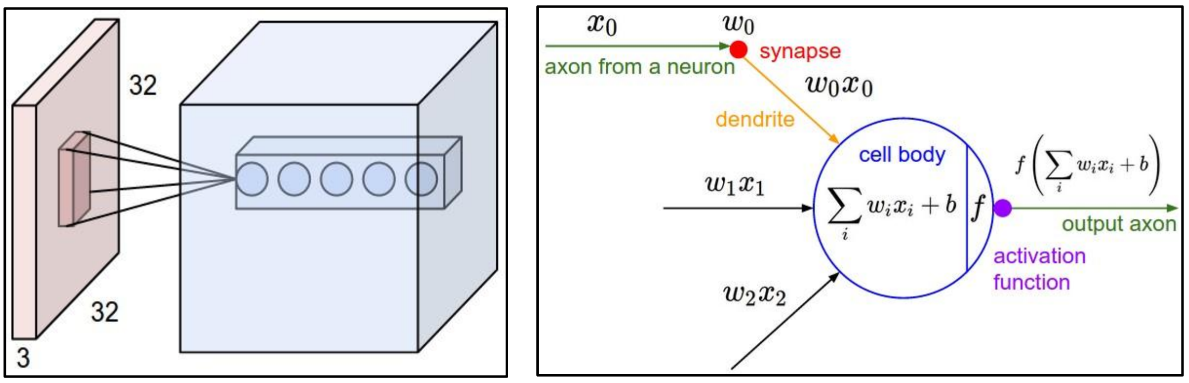
4.2.2. A neurological analogy

Looking at the function of CONV layer from a neurological perspective, every entry in the output volume can also be interpreted as an output of a neuron which deals with only a small region in the input and shares parameters with all neurons to the left and right spatially, since these numbers all result from applying the same filter.

4.2.3. Local connectivity

As discussed above, when dealing with high-dimensional inputs such as images, it is impractical to densely connect each neuron to all other neurons in the previous volume. Instead, in CONV layer each neuron is connected to only a local region of the input volume. The spatial extent of this connectivity is a hyperparameter called the *receptive field* of the neuron which is essentially equivalent to the filter size. The extent of the connectivity along the depth axis is always equal to the depth of the input volume. It is important to emphasize that this asymmetry in how the spatial dimensions i.e. height and width and the depth dimension are treated: the connections are local in space (along width and height), but always full along the entire depth of the input volume.

For example, suppose that the input volume has size [32 × 32 × 3], (e.g. a CIFAR-10 image or a CMATERdb 3.1.1 image). If the receptive field i.e. the filter size is 5 × 5, then each neuron in the CONV layer will have weights to a [5 × 5 × 3] region in the input volume, which comes to a total of 5*5*3 = 75 weights (and +1 bias parameter). It should be noted that the extent of the connectivity along the depth axis should be 3, since this is the depth of the input volume. A graphical representation of CONV layer neurons is presented in Fig. 4.3.



(a) (b)

Fig. 4.3: *(a) An example input volume in red (e.g. an 32 × 32 × 3 image), and an example volume of neurons in the first CONV layer in blue. (b) The neurons are same as classical ANNs but their connectivity is now restricted to be local spatially.*
[Image source: Karpathy[136]]

4.2.4. Spatial arrangement of neurons

In this section, the spatial arrangement of neurons in the CONV layer is discussed. Three hyperparameters control the size of the output volume: *depth, stride* and *zero-padding*. We discuss about these hyperparameters in the following section.

The *depth* of the output volume is a hyperparameter which corresponds to the number of filters that would be used for a particular task, each learning to look for something different in the input. For example, if the first Convolutional Layer takes as input the raw image, then different neurons along the depth dimension may activate in presence of various oriented edged, or blobs of colour. A set of neurons that are all looking at the same region of the input are generally referred to as a *depth column* or *fibre*. Second, the hyperparameter *stride* signifies the distance we slide the filter once an output volume is produced. If the *stride* is set to S then the filters are moved S pixels at a time. Larger *stride* values produce smaller output volumes spatially. Sometimes it is convenient to pad the input volume with zeros around the border. The size of this *zero-padding* is a hyperparameter. The nice feature of zero padding is that it will allow us to control the spatial size of the output volumes.

We can compute the spatial size (O) of the output volume as a function of the input volume size (I), the receptive field size of the Convolution Layer neurons (F), the stride with which they are applied (S), and the amount of zero padding used (Z) on the border. The formula for calculating how many neurons fit in the output volume is given by the following equation:

$$O = \frac{I - F + 2Z}{S} + 1 \qquad (4.1)$$

For example, for an 7×7 input and a 3×3 filter with stride 1 and pad 0 we would get an 5×5 output, whereas with stride 2 we would get an 3×3 output.

4.2.5. Parameter sharing

One of advantages of CNN models compared to classical ANNs is the parameter sharing scheme. Inspired by the structure of visual cortex, parameter sharing scheme is used in Convolutional Layers to control the number of parameters. For example, the CNN model proposed by Krizhevsky et al.[109], also known as *AlexNet*, which won the ImageNet 2012 challenge, used 55*55*96 = 290,400 neurons in the first CONV layer, each of which has 11*11*3 = 363 weights and 1 bias parameter. Together, this adds up to 290400 * 364 = 105,705,600 parameters on the first layer of the Convolutional Net alone. Clearly, this number is very high.

Following the parameter sharing scheme, however, the number of parameters can be drastically reduced by making one reasonable assumption: if one feature is useful to compute at some spatial position (x, y), then it should also be useful to compute at a different position $(x2, y2)$. In other words, denoting a single 2-dimensional slice of depth as a *depth slice* (e.g. a volume of size $[55 \times 55 \times 96]$ has 96 depth slices, each of size $[55 \times 55]$), we are going to constrain the neurons in each depth slice to use the same weights and bias. With this parameter sharing scheme, the first CONV layer in our example would now have only 96 unique set of weights, one for each depth slice, for a total of 96*11*11*3 = 34,848 unique weights, or 34,944 parameters (adding 96 bias parameters). Consequently, all 55*55 neurons in each depth slice will now be using the same parameters. In practice, during backpropagation every neuron in the volume computes its weight gradient, but these gradients are added up across each depth slice and only a single set of weights per slice is updated at the end.

If all neurons in a single depth slice use the same weight vector, then the forward pass of the CONV layer in each depth slice can essentially be computed as a *convolution* of the neuron's weights with the input volume (hence it gets its name). This is why it is common to refer to the sets of weights as a *filter* or a *kernel* which convolves with the input.

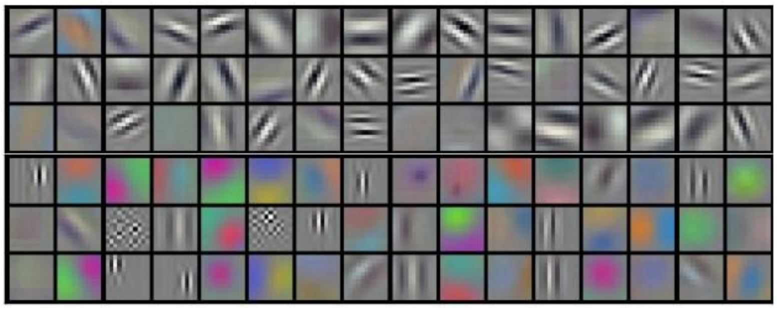


Fig. 4.4: *Graphical representation of all 96 filters, each of dimension* $[11 \times 11 \times 3]$, *shared by 55*55 neurons of each depth slice in AlexNet*

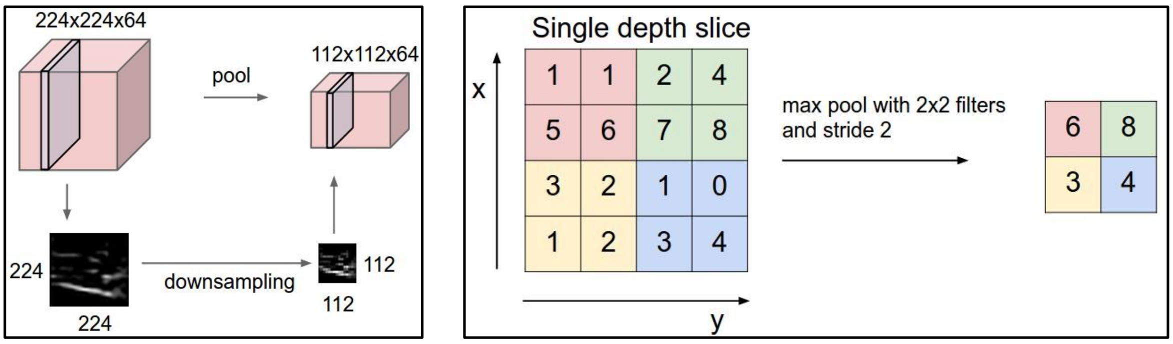
[Image source: Krizhevsky et al.[109]]

4.3. Pooling layer

A *Pooling layer* or *subsampling layer* performs an aggregation operation (typically max or average operation) on the values of neurons over a small region (also called *pooling window*) in a feature map. Pooling can also be overlapping in nature. It usually introduces an amount of local translational invariance since an activation is propagated forward ignoring minor positional variances, if it falls within the *pooling window.*

It is common to periodically insert a Pooling layer in-between successive CONV layers in a CNN architecture. It progressively reduces the spatial size of the vector representation and also the amount of parameters and computation in the network, which in turn helps avoid overfitting. Pooling also reduces computational complexity and allows for CONV layers in feature discovery of higher abstraction over lower granularity.

The Pooling layer operates independently on every depth slice of the input and resizes it spatially, using an aggregation operation. The most common form is a pooling layer with max operation, having filters or *pooling windows* of size 2×2 applied with a stride of 2. It essentially downsamples every depth slice in the input by 2 along both dimensions of spatial extent i.e. width and height, therefore discarding 75% of the activations. A graphical demonstration of a max pooling operation over a *depth slice* is shown in Fig. 4.5.



(a) (b)

Fig. 4.5: *Graphical demonstration of max pooling operation with a 2× 2 pooling window and pool stride of 2 in AlexNet*

[Image source: Karpathy[136]]

4.4. Fully connected layer

Neurons in a *fully connected* layer have full connections to all activations in the previous layer, as it normally can be seen in regular ANNs. Hence their activations can be computed in a traditional way i.e. a matrix multiplication followed by a bias offset.

4.5. Rectified linear unit

Rectified Linear Units[137] (RELU) are used as activation functions and generally used for enhancing the most interesting features in each CONV layer activation map. Each RELU unit is essentially a hidden node with activation function described as following:

$$f(x) = max(0, x + N(0, \sigma(x))) \tag{4.2}$$

Where $N(0, \sigma(x))$ denotes a Gaussian noise with a mean equal to 0 and variance of $\sigma(x)$. RELU layers ensure element-wise non-linearity, thresholding at zero. Rectified non-linearity has also been shown to converge faster[109] than non-saturating non-linearities such as hyperbolic tangent.

One of the major contributions of the Rectified Linear Unit is that it does not suffer from the vanishing gradient problem (discussed in Chapter 2). The reason for this is obvious from the derivative of the Rectified activation function as the value of its derivative does not gradually decrease, propagating towards higher layers.

4.6. Minibatch gradient descent based learning

In terms of training a Multilayer Neural Network with *backprogagation learning algorithm*, *stochastic* or *online learning* refers to updating the connection weights using a random example at every iteration of learning. Another approach is to update the weight on an average of the error after running through the entire training set. Although there are advantages to *stochastic learning* as it is faster compared to batch learning, but stochastic learning algorithms converge slowly or get stuck in local minima due to weight fluctuations caused by noisy samples. On the other hand, batch learning usually converges to a global minima but is obviously slower since the entire training set is required before updating the weights which makes vectorization even harder.

Minibatch gradient descent based learning can be thought to be a compromise between *stochastic learning* and *batch learning* as following this scheme, weights are updated more frequently than batch learning. It has a better representation of the error gradient since it uses an average error calculation, based on more than one sample, unlike stochastic gradient descent. It can typically average out noise while being vectorizable at the same time. The vectorizability of this learning method has made it hugely popular among researchers for its superior performance over large datasets.

A brief overview of Convolutional Neural Network, its basic components and an understanding of how it works has been presented in this chapter. As all the required knowledge necessary for understanding the motivation and significance of the present work has been laid out in the previous chapters, motivations and significance of our present work can now be described. An overview of the present work and the methodology adopted is discussed in the next chapter.

Chapter 5

Present work

The main objectives of the present work is two-fold: (a) propose a fast feature extraction technique which can be ubiquitously applied for multilingual and complex Indian scripts and (b) prove the superiority of the proposed method over its contemporaries. As mentioned earlier in Chapter 3, Neural Networks can be used as both feature extractor and classifier. Researches like Das et al.[138], Basu et al.[29] etc. have used multi-layer perceptrons for classifying pattern images, represented by explicitly extracted discrete feature vectors. On the other hand, researchers like LeCun et al.[93][132], Ciresan et al.[98] etc. have used deep neural network architectures as both feature extractor and classifier. From our discussion in Chapter 2, it is clear that identifying the right features for the representation of handwritten characters and/or numerals pertaining to a specific language is a challenging task. Clearly, if we were to propose an efficient OCR system which works with handwritten characters and/or numerals sprawling across multiple scripts, the challenge increases many folds. The system proposed in our present work has to overcome all of these difficulties and complications.

Inspired by the apparent ease[58][85][31] evident in *non-explicit feature based approaches* and the overwhelming success[132][109] of deep neural networks in wide array of applications, a similar approach towards recognition of multilingual handwritten characters and/or numerals is taken in our present work. A novel multi-scale deep quad tree based feature extraction technique is proposed in our present work. The proposed method has been tested on 6 publicly available benchmark datasets of isolated handwritten characters and numerals. The system has been able to achieve state of the art performance in all of them. The proposed deep architecture and the subsequent feature extraction technique mark the contributions of this work.

The proposed CNN based architecture and the feature extraction schemes proposed under our current experimental setup is described in the following sections.

5.1. Datasets used for the experiments

A set of publicly available benchmark datasets of isolated handwritten characters and numerals is used for experimental purposes in our current experimental setup. As discussed earlier, one of the objectives of the present work is to propose a character recognition system which sprawls across a number of Indian languages instead of focusing too much on a specific script. Hence, datasets of isolated, handwritten characters and numerals, belonging to several different Indian languages are chosen in our experimental setup. Details of each dataset is given in Table 5.1 below. More details regarding the number of writers, profile of the writers, data collection and processing methodologies etc. can be found in the references cited in the right most column of the table.

Index	***Name of the dataset***	***Database type***	***Language***	***Number of training samples***	***Number of test samples***	***Reference***
D1	CMATERdb 3.1.3	Bangla Compound character	Bangla	34229	8468	[139]
D2	CMATERdb 3.1.2	Bangla Basic character	Bangla	12000	3000	[134]
D3	HPL Offline Tamil ISO Character DB	Tamil character	Tamil	70000	12000	[140]
D4	CMATERdb 3.1.1	Bangla digit	Bangla	4000	2000	[139]
D5	CMATERdb 3.2.1	Hindi digit	Hindi	2000	1000	[141]
D6	CMATERdb 3.4.1	Telugu digit	Telugu	4000	2000	[141]

Table 5.1: *Databases used in current experimental setup*

5.2. Preprocessing steps

The images of each dataset are passed through a number of preprocessing steps. Each image of the isolated handwritten character or numeral is binarized using Otsu's method[63], blurred using a median filter[63], smoothed using a Gaussian distribution[63], resized to 64×64 pixels and centred by the tightest bounding box. A flowchart representing the sequence of these pre-processing steps is shown Fig. 5.1.

As discussed earlier (see Chapter 4), there are several schemes adopted by CNN architecture, which help reduce the number of parameters in a CNN model. For a kernel of spatial dimension $w \times h$, convolving on a RGB image, the number of parameters shared by the neurons in each depth slice is equal to $w * h * 3$. Although already significantly reduced from the number of parameters in a classical ANN model, it still may lead to overfitting. Binarizing the images at the start during the pre-processing steps help reduce the number of parameters shared by neurons by a significant 66% which in turn helps increase the convergence speed of the network. Binarizing the images also help subsequent processing of the images.

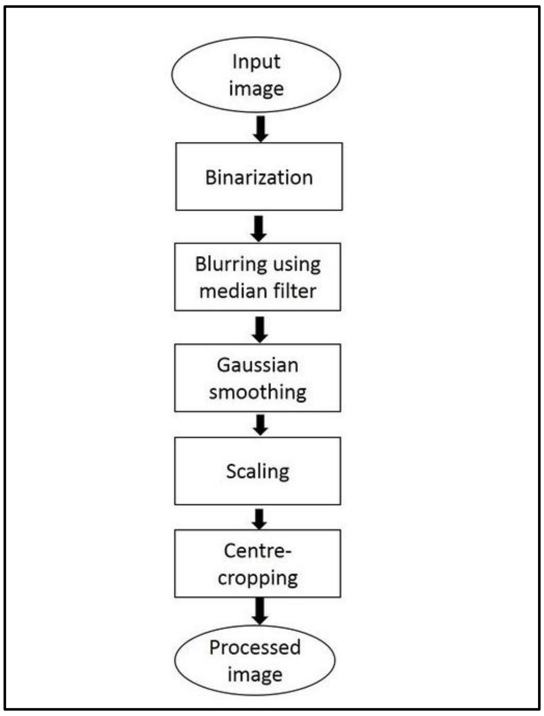


Fig. 5.1: *A flowchart of pre-processing steps taken in our current experimental setup*

Upon closer inspection of some of the datasets used in our experiment, it was found that some of the images in the dataset contained salt and pepper noise, which was adversely affecting the recognition performance of the prediction model. Hence, median filter based blurring was applied to the images to reduce the amount of noise from the isolated character and/or numeral images. Subsequently, smoothing was applied to the images to address some of the broken characters present in the datasets. A tight bounding box was taken and the images were centre-cropped to help improve[59][142] the recognition performance of the classifier models in subsequent steps. Finally, the images are scaled to 64×64 pixels to allow for various elastic deformations during the training process.

5.3. Overview of the proposed deep architecture

As discussed earlier, we have taken a *non-explicit feature based approach* towards handwritten character recognition in our present work. One of the greatest challenges in feature selection for handwritten character recognition task is that the feature should be expressive enough to detect invariant local properties from highly varying handwritten characters, authored by different writers. Evidently, if the task at hand is to propose such a feature which is also script independent, feature selection becomes more challenging than ever. Hence, a *multi-scale multi-column convolutional neural network* (MMCNN) based architecture is proposed in the present work for feature extraction from raw input images. Once the weights are learnt and the network converges, connection weights between the last fully connected (FC) layer and the softmax layer for each column of the network architecture can be considered[58] as implicit features. A graphical abstract of the deep architecture proposed in the present work is shown in Fig. 5.2.

Inspired by the microcolumns of neurons in the cerebral cortex area of human brain and encouraged by the superior performance of *multi-column deep neural network* (MCDNN) architecture by Ciresan et al.[98] which won the ICDAR 2012 challenge, a multi column CNN based architecture is proposed in the present work. The architecture consists of three columns, where each column is itself a CNN based multilayer network. As mentioned before, handwritten characters stylistically vary from one individual to another. Although convolutional neural networks (CNN) have shown their promise as a universal representation for recognition, global CNN activations still lack some geometric invariance, which limits their robustness for classification and matching of highly variable pattern images. To improve the invariance of CNN activations without degrading their discriminative power, a *multi-scale orderless pooling*[143] technique has been applied to the CNN based models comprising each column. This scheme essentially forms a spatial pyramid of CNN activations, where each level of the pyramid is associated with a specific *convolutional filter* or *kernel* (see Chapter 4). Following this scheme, the final convolutional feature set is obtained by concatenating the feature vectors obtained from each level of the spatial pyramid.

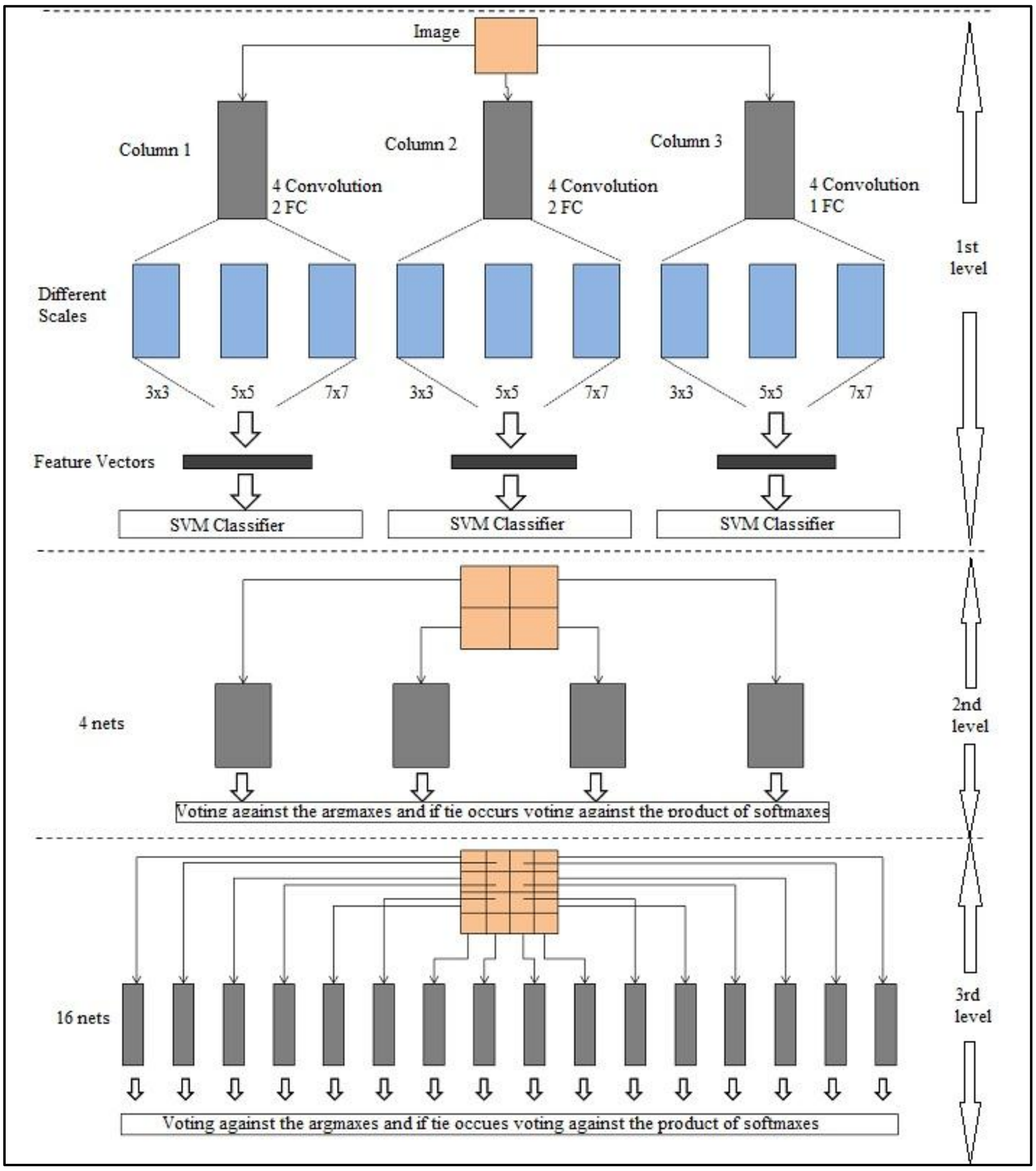
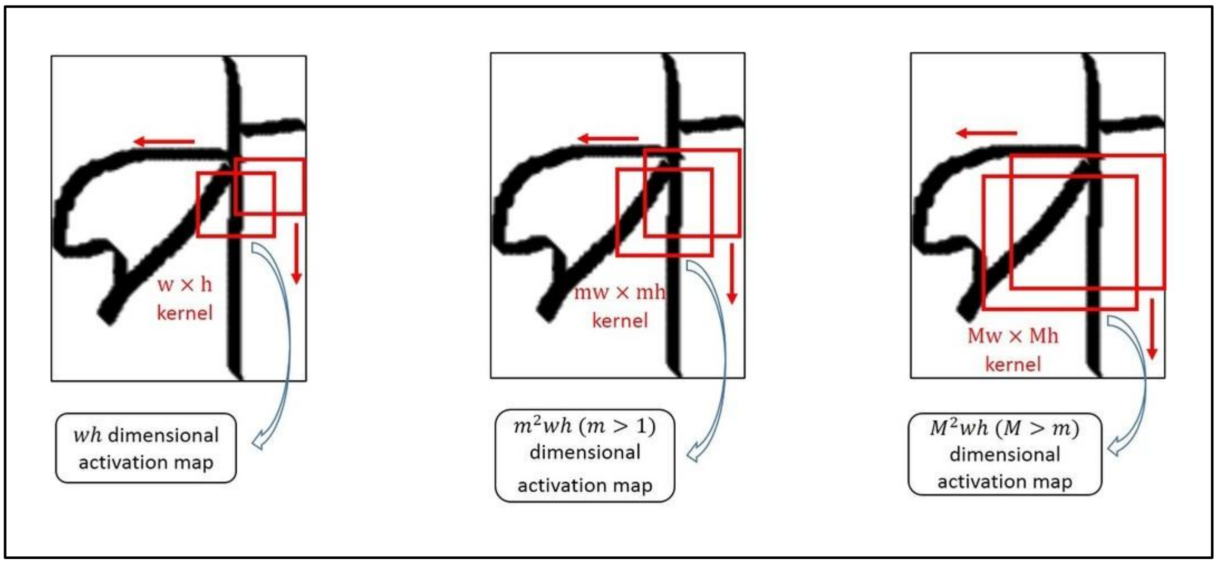


Fig. 5.2: *A complete overview of the proposed MMCNN based architecture*

An overview of the multi-scale pooling employed in the present work is shown in Fig. 5.3. Each column of the proposed deep network has 3 layers. Each layer $i, i = 1$ to 3 contains 4^{i-1} stacked Convolutional Neural Networks. As we go down the layers, lower layers of each column focuses on the local activations i.e. local characteristics of the pattern image, more than its previous layer. This is achieved by applying a *static zoning technique*[59] on each input image of the previous layer. For each input image I in the previous layer, a 2×2 grid is superimposed on I. The sub-images at each grid cell i.e. I_1, I_2, I_3, I_4 are treated as independent input images at the next layer. The immediate lower layer tasks a separate CNN for each sub-image $I_j, j = 1$ to 4. *Static zoning techniques* are used to divide the image into sub-images as opposed to *dynamic zoning techniques*, as they are relatively faster. Although researchers like Das et al.[45], Sarkhel et al.[16] etc. have advocated the use of CG based quad tree partitioning[27][28] of a handwritten character, in our experiments we have found that equal partitioning based quad tree approach works better than CG based quad tree partitioning in case of convolutional feature-vector based representation of a pattern image. It is also relatively faster. A comparative analysis of the proposed architecture working on both CG based quad tree partitioning and equal partitioning quad tree based approach, on some of datasets used in our experimental setup is shown in Fig. 5.4. Tasked with recognizing a sub-image, each smaller Convolutional network in the lower layer, carries on with its own recognition task. Once each smaller net in the stack has finished with the recognition task on a sub-image, aggregate result on the entire image is obtained by performing a majority voting based on class labels assigned to each sub-image. Tie cases are resolved based on the class scores assigned by the prediction model for each column. A mapping of the smaller CNN based networks (shown in Fig. 5.2) with the sub-image (shown in Fig. 5.5) it is assigned for recognition for both second and third layer of each column in described in Table 5.2 and Table 5.3 respectively. The indexes are same as shown in Fig. 5.2 and Fig. 5.5 respectively.

In the current experimental setup, a *percolated* or *staggered prediction model* is taken up for each column of the proposed architecture. Control reaches to the lower layer of stacked CNNs only for the misclassified class labels based on a 5 fold cross-validation performed on the dataset. After obtaining prediction scores from each column, the final class prediction for the test pattern image is obtained by performing an aggregation operation. More details regarding how the prediction model works for our proposed architecture will be discussed later in the following sections.



w × h kernel
wh dimensional activation map
mw × mh kernel
m^2wh $(m > 1)$ dimensional activation map
Mw × Mh kernel
M^2wh $(M > m)$ dimensional activation map

Fig. 5.3: *A demonstration of multi-scale pooling on a sample handwritten character image*

5.4. Column architecture

The propsoed approach is inspired by Hubel and Wiesel's seminal work[116] which identified orientation-selective simple cells with overlapping local receptive fields and complex cells performing down-sampling-like operations on cat's primary visual cortex. They were the among the first researchers to successfully propose a computational model which was able to mimic the visual perception of an intelligent animal.

As mentioned earlier, the proposed architecture contains three independent columns working in parallel. Each column has a layered architecture. In our experimental setup, there are three layers in each column. The first layer contains a multi-scale CNN based model. The second and third layer contains stacked Convolutional networks. There are respectively 4 and 16 smaller networks in the second and third layer of each column. While the convolutional net at the first layer of a column processes an entire input image, each network at the second and third layer, processes only $1/_4$ th and $1/_{16}$ th of the input image. A graphical representation of the task assignment in the stacked CNN models at lower layers of each column is shown in Fig. 5.2.

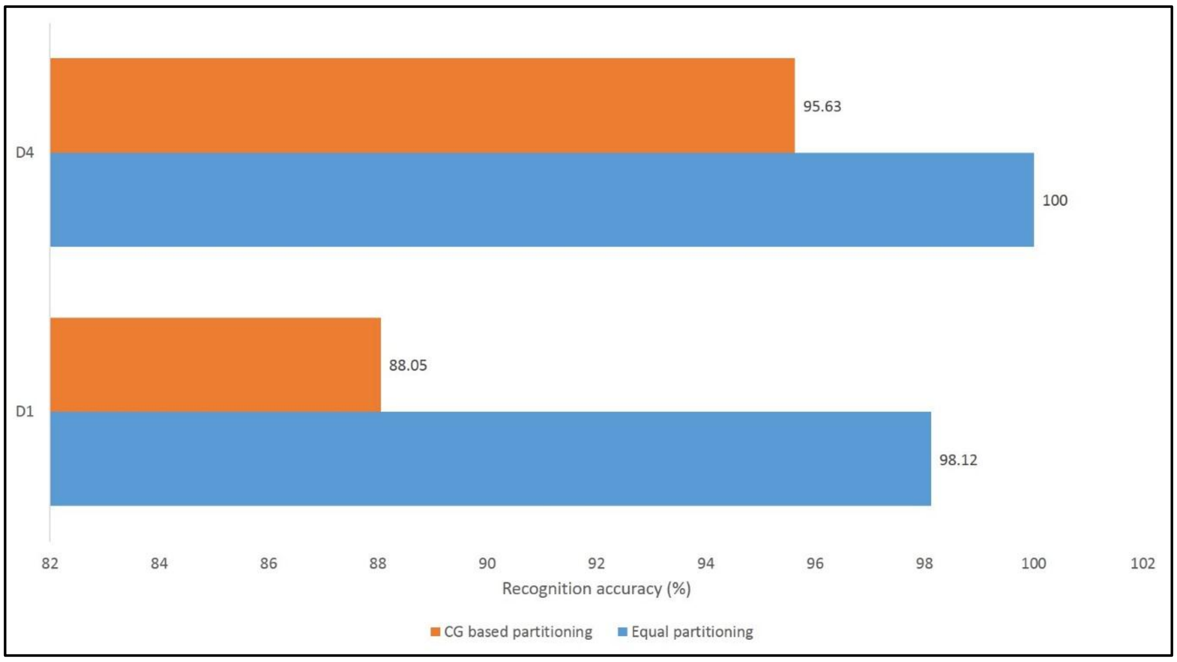


Fig. 5.4: *Comparison between equal partitioning and CG based partitioning on two separate datasets (database indexes are same as shown in Table 5.1)*

Let $I_{w \times h}$ denotes an input image. In the *percolated prediction model* (discussed in Section 6) adopted in our proposed architecture, the misclassified character classes based on a 5-fold cross-validation are percolated to the lower layer of stacked CNNs at each column. Suppose, the smaller CNN based models at the second and third layer are denoted as N_i, $\mathrm{i} = 1$ to 4 and M_i, $\mathrm{i} = 1$ to 16 respectively. If the control comes to the second layer of the column, the input image $I_{w \times h}$ is divided into 4 sub-images, $I_j, j = 1$ to 4 where each sub-image has the dimension of ${}^{w}/_{2} \times {}^{h}/_{2}$ (shown in Fig. 5.5(a)). One of the smaller networks $N_j, j = 1$ to 4 from the stacked CNNs at the second layer is assigned for the recognition of the sub-image I_j. Same steps are repeated for each sub-image $I_j, j = 1$ to 4 if the control comes to the third layer of the column. Each sub-image $I_{ij}, j = 1$ to 4, for each $i = 1$ to 4 (shown in Fig. 5.5(b)) is assigned to one of the smaller convolutional networks $M_j, j = 1$ to 16. Each of these smaller networks in the stacked second and third layers perform separate recognition tasks. Character class is assigned to the pattern image after performing a majority voting based on the sub-image class labels.

Parameters of the architectures of each column has been finalized empirically. Each column has 3 layers. While the first layer of each column is itself a CNN, second and third layers of the column are stacked CNNs, each having 4 and 16 CNNs respectively.

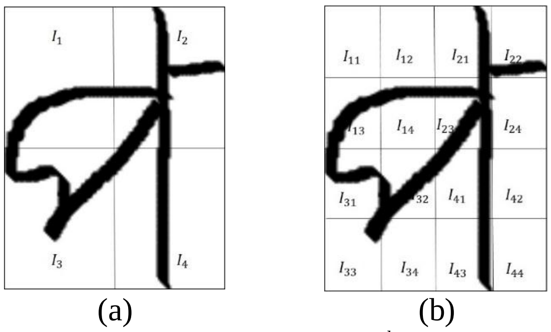


Fig. 5.5: *Equal partitioning of the input image for (a) 2nd layer and (b) 3rd layer of each column*

Sub-image index	***Second layer stacked CNN index***
I_1	N_1
I_2	N_2
I_3	N_3
I_4	N_4

Table 5.2: *Mapping of image quadrants with stacked convolutuional neural nets in second layer of each column*

Sub-image index	***Third layer stacked CNN index***
I_{11}	M_1
I_{12}	M_2
I_{13}	M_3
I_{14}	M_4
I_{21}	M_5
I_{22}	M_6
I_{23}	M_7
I_{24}	M_8
I_{31}	M_9
I_{32}	M_{10}

I_{33}	M_{11}
I_{34}	M_{12}
I_{41}	M_{13}
I_{42}	M_{14}
I_{43}	M_{15}
I_{44}	M_{16}

Table 5.3: *Mapping of image deca-hexadrants with stacked convolutuional neural nets in third layer of each column*

Architecture of the first layer of the first column can be described as 32C2-2P2-RELU-64C3-RELU-256C3-4P4-RELU-512C3-RELU-2048FC-RELU-1024FC-RELU-RBF_SVM, where XCy denotes a convolution layer with X number of kernels and a stride of y pixels, MPn denotes a max pooling layer with a M × M poling window and stride of n pixels, RELU denotes a RELU activation layer, nFC denotes a fully connected layer with n neurons and finally RBF_SVM denotes the SVM classifier with RBF kernel, used as a softmax classifier. Parameters of the SVM are tuned by performing β/γ variations. To introduce geometric invariance in the global activations extratced from the input image, multi-scale convolution is employed at the first convolutional layer of each column. In the proposed architecture, we have used three different kernel diemnsions which are 3 × 3, 5 × 5 and 7× 7 repectively. Following the same notations, architecture of each CNN of the stacked CNNs in second and third layer can be described as: 12C2-2P2-RELU-20C2-2P2-RELU-400C2-2P2-RELU-1024FC-RELU-512FC-RELU-RBF_SVM.

The first layer of the second column has the following architecture: 24C4-4P4-RELU-64C2-RELU-150C2-2P2-RELU-240C2-RELU-1024FC-RELU-512FC-RELU-RBF_SVM. The notations are same as mentioned above. Architecture of the stacked CNNs in the second and third layer are same as it was in the first column. Similar to first column, a SVM with RBF kernel is used as a softmax classifier for all the layers of the network.

The third column is constructed in a similar fashion as the first and second column. First layer of the column has the following architecture: 12C2-2P2-RELU-32C2-2P2-RELU-120C2-2P2-RELU-240C1-RELU-256FC-RELU-RBF_SVM. The notations are same as mentioned above. The stacked CNNs in the second and third layer are same as it was in the first ans second column. Similar to the previous two columns, a SVM with RBF kernel is used as a softmax classifier for all the layers of the network.

A graphical representation of how an input image propagates through the network is shown in Fig. 5.6. Due to space limitation, given the size of the proposed network, the representation is provided with respect to only the first column of the network.

Designing a network starts with the input size. For this problem we observed that 64×64 pixels are sufficient to properly represent details of a handwritten character and/or numeral. Our previous experiments with deep neural networks on many other data sets have led us to the decision that deep and wide networks learn best to generalize. Every convolutional and maxpooling layer decreases the map size from its previous layer. Evidently, maxpooling layers are the worst offenders. Therefore it is imperative to use smaller kernels for them. Convolutional layers are comparatively less problematic which allows convolutional filters to be larger in size. However, too large filters are not required because at least one of the even or odd sized filters will generate maps of even size, necessary for the next maxpooling layer with a 2×2 pooling window. The convolution and maxpooling layers extract features with increasing complexity at every layer, forming a hierarchy of feature abstraction. The convolution layers share weights, consequently the feature extractor contains only about 20% of the net's weights.

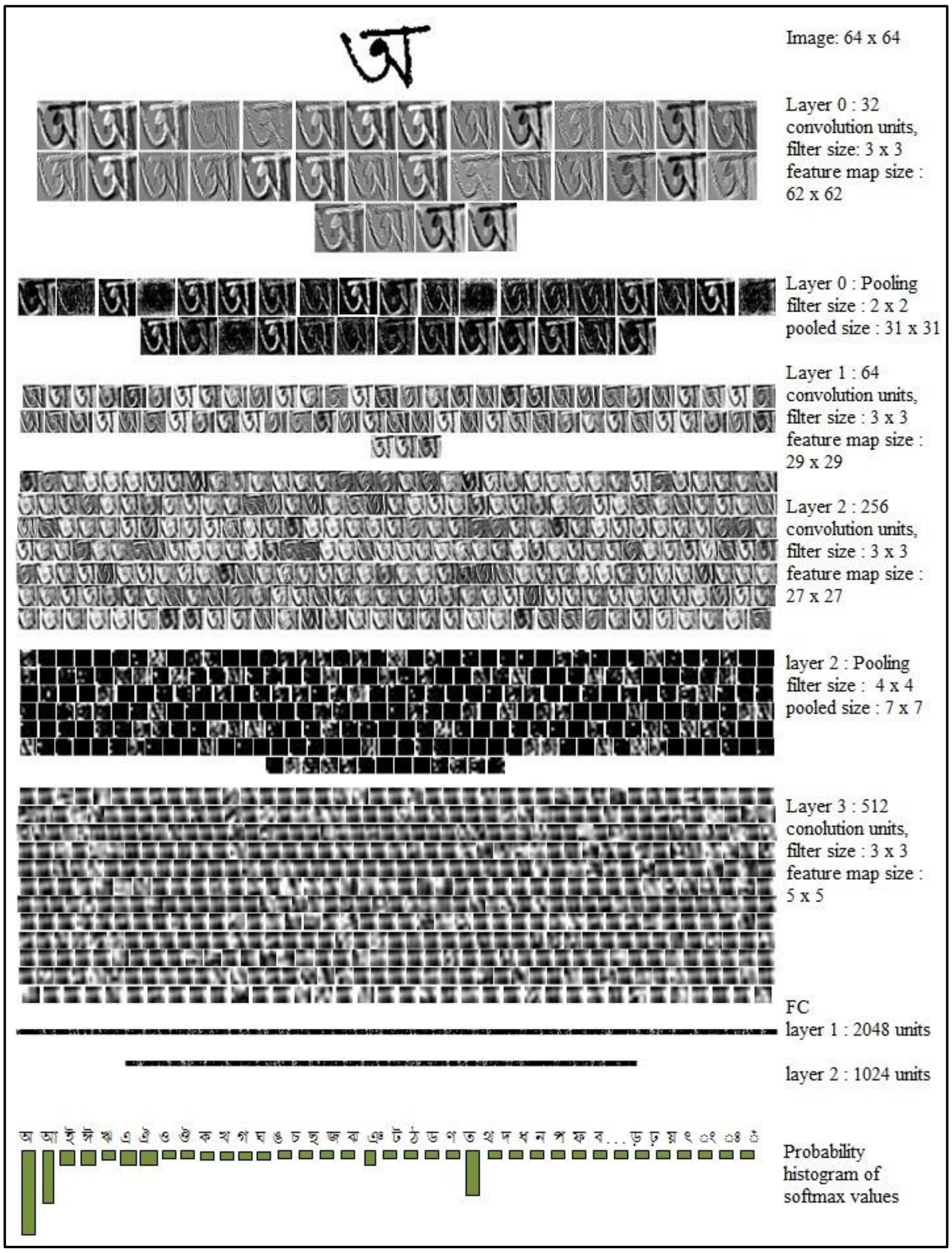


Image: 64 x 64
Layer 0 : 32 convolution units, filter size: 3 x 3 feature map size : 62 x 62
Layer 0 : Pooling filter size : 2 x 2 pooled size : 31 x 31
Layer 1 : 64 convolution units, filter size : 3 x 3 feature map size : 29 x 29
Layer 2 : 256 convolution units, filter size : 3 x 3 feature map size : 27 x 27
layer 2 : Pooling filter size : 4 x 4 pooled size : 7 x 7
Layer 3 : 512 conolution units, filter size : 3 x 3 feature map size : 5 x 5
FC layer 1 : 2048 units
layer 2 : 1024 units
অ আ ই ঈ ঋ এ ঐ ও ঔ ক খ গ ঘ ঙ চ ছ জ ঝ ঞ ট ঠ ড ণ ত থ দ ধ ন প ফ ব . . . ড় ঢ় য় ৎ ং ঃ ঁ
Probability histogram of softmax values

Fig. 5.6.1: *Forward propagation of an input image of Bangla handwritten chaarcter through the first layer of first column of the proposed MMCNN based architecture after learning is done*

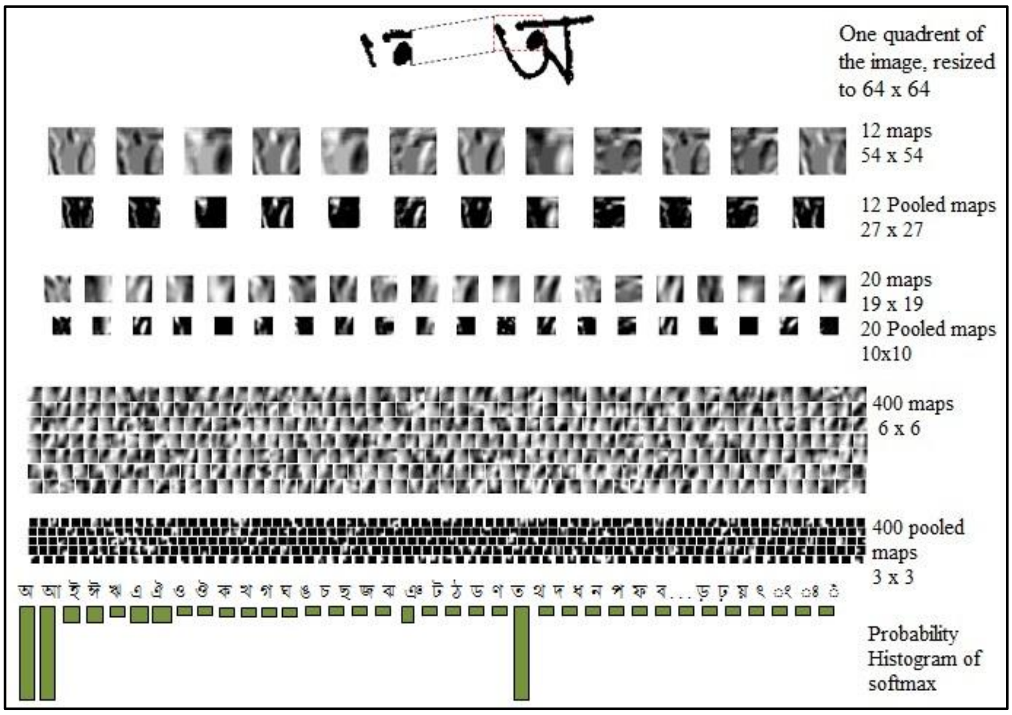


Fig. 5.6.2: *Forward propagation of an input image quadrant throught the second layer of the first column of the proposed MMCNN based architecture after learning is done*

Fig. 5.6: *Demonstartion of features extracted from the proposed MMCNN based architecture*

The network should also be wide enough, i.e., extract suffucient details from the pattern images by computing sufficiently expressive features to extract details required for high-quality recognition. We started with 32 maps on the first convolutional layer, then increased the number for every successive convolutional layer. In the present work, a RBF kernel based SVM is used as a softmax layer for each column. SVM is used instead of a MLP based classifier in our architecture as we have found in our experiments that the average performance of SVM is better than that of MLP. In our literature review, we have found that several contemporary researchers[72][3][57] have also came to the same conclusion.

5.5. Training the deep network

For faster learning of the large network, a mini-batch gradient descent based learning technique is used. Each column of the network is trained separately. The smaller networks in the second and third layers of each column are trained on the sub-images of the training images i.e. the quadrants and deca-hexadrants assigned to them, as shown in Table 5.2 and Table 5.3.

As suggested by LeCun et al.[144], before each epoch of mini-batch gradient descent based training, the dataset is randomly shuffled. They have shown in [144] that the data shuffling before each epoch of the learning algorithm, introduces heterogeneity in the dataset with which each weight update is calculated. According to them, this increases the convergence rate of the learning algorithm, as a deep network learns "the fastest when it sees the most unexpected sample". Besides global learning of connection weights, an adaptive local learning process *RMSProp*[145] is also used in our current experimental setup to increase the convergence rate of the parameter learning process. The method works by keeping a moving average of the squared gradient for each weight and dividing the gradient by the square root of this value. It has been shown that despite its limitations, RMSProp improves the learning process by help improving the convergence rate. The weight update at each iteration is done by following the below equation:

$$w_{t+1} = w_t - \frac{\alpha}{\sqrt{r_t}} f'(\theta_t) \qquad (5.1)$$

$$\text{Where, } r_t = (1-\gamma) f'(\theta_t)^2 + \gamma r_{t-1} \qquad (5.2)$$

In equation 5.1, r_t denotes the modified learning rate at tth epoch of the learning algorithm, α denotes the global learning rate and $f'(\theta_t)$ denotes the error-gradient. In equation 5.2, γ denotes a decay rate and r_{t-1} denotes the learning rate at the previous iteration of the algorithm.

The learning algorithm is not terminated until the cross-validation error rate converges or the number of epochs reaches the predefined maximum. A regularization method is also used in our current experimental setup to prevent the model from overfitting. Although the weight sharing scheme reduces the number of parameters by a significant 80%, compared to classical

ANNs, it still uses a huge number of parameters. Dropout regularization method is used in the present work to help reduce the chances of overfitting the network while training. The parameters of the learning algorithm used to train each layer of the three columns is given in Table 5.4.

Parameter name	***Parameter value***
Global learning algorithm	Mini-batch gradient descent algorithm
Local learning	RMSProp
Learning rate (α)	0.001
Decay rate (γ)	0.01
Dropout	0.5
Batch size	5
Maximum number of epochs	1000

Table 5.4: *Parameters of the learning algorithm used for all of the CNN based models used in proposed MMCNN architecture*

5.6. Extraction of feature set

A non-explicit feature based approach towards OCR system is taken in the present work. After the network is trained and weights are learned, a pattern image is propagated forward through the network. The extracted features are essentially connection weights between the last fully-connected layer and the softmax layer. All of the 512 kernels used by the first layer CNN based model of the first column for extracting feature from an input pattern image is shown in Fig. 5.7. For visual representation of how the extracted features emerge upon multiple epochs of the learning algorithm, we have shown the evolution of the convolution maps of a input test image over multiple epochs, when propagated forward through the first layer CNN based model of the first column in the proposed MMCNN architecture, in Fig. 5.8.

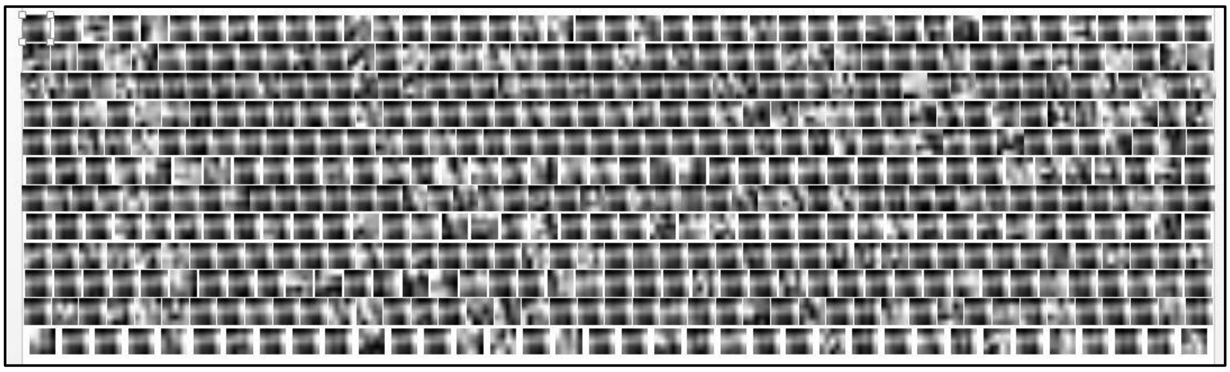


Fig. 5.7: *All 512 kernels used by the first layer CNN of the first column of the proposed MMCNN architecture*

It is evident that the dimension of the featurevectors extracted from each layer of the three columns depend on the number of neurons in the last fully-connected layer and the batch size. Following the scheme adopted in the present work, the featurevector dimension (D) can be computed according to the following equation:

$$D = N_{FC} \times b \qquad (5.3)$$

Where N_{FC} denotes the number of neurons in the last fully connected layer and b denotes the mini-batch size used in our experimental setup. Following the equation 5.3, dimension of the featurevectors extracted from each layer of the three columns in our architecture is shown in Table 5.5.

Column index	***Layer index***	***Dimension of featurevector***
1	1	5120
	2	2560
	3	2560
2	1	2560
	2	2560
	3	2560
3	1	1280
	2	2560
	3	2560

Table 5.5: *Dimensions of extracted featurevectors from different layers of the proposed MMCNN architecture*

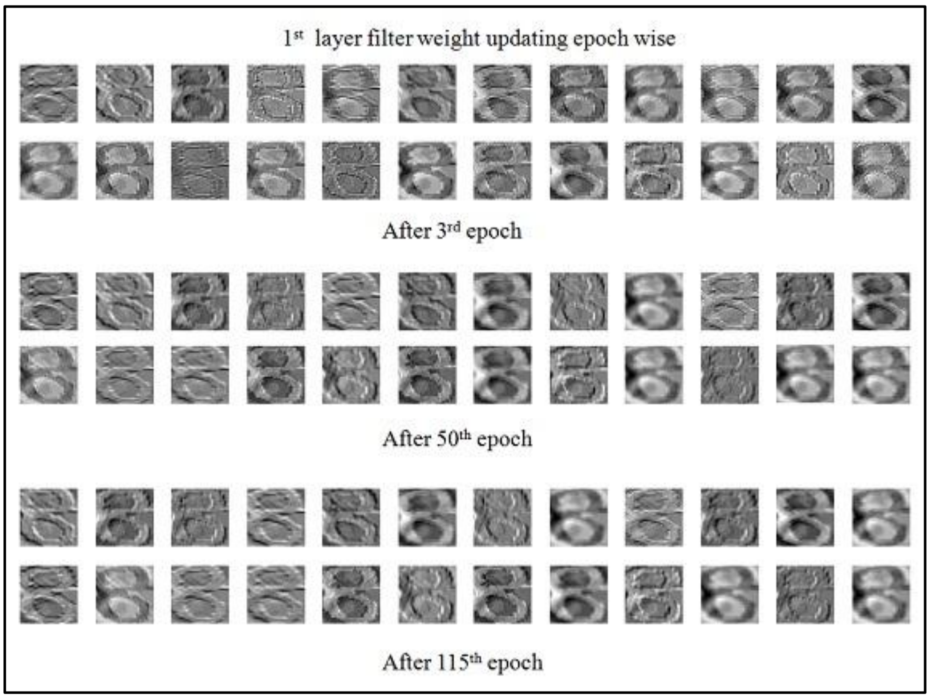


Fig. 5.8: *Evolution of feature maps of an input image when propagated through the first layer of the first column of the proposed MMCNN architecture*

5.7. The percolated prediction model

One of the objectives of the present work is to propose a *fast* feature extraction technique which can be ubiquitously applied for multilingual and complex Indian scripts. Due to the complexity of the task at hand, it was necessary to take a non-explicit feature based approach towards OCR system in the present work. A deep architecture is proposed in the scope of our current work for this purpose. After the network is trained, implicit connection weight based feature set is extracted from it. These features are subsequently used to represent the training dataset. Features from the test dataset is extracted in a similar fashion. Once the network converges i.e. the weights are learnt, the pattern image of which convolutional features need to be extracted is propagated forward through the learned network and consequently, the feature set is extracted from the connection weight between the last fully connected layer and the softmax layer. Considering that there is a total of 21 CNN based network models in each of the three columns of the proposed architecture, if all of the layers were to be used simultaneously for each and every test image, it will a take a long time for feature extraction as the pattern image needs to be propagated through all of them. Needless to say, the length of the feature vector to represent each pattern image would also be huge (shown in Table 5.5), which will increase the dimension of the feature space and consequently make the job of the classifier to find a decision boundary quite challenging. These difficulties motivated us to use a staggered or percolated prediction technique instead. A flowchart of the prediction model is shown in Fig. 5.9. The prediction algorithm is described in details as Algorithm 1 in the following section.

Algorithm 1: A percolated prediction algorithm for MMCNN architecture

Input: Test image I.

Output: Class label of I.

```
1   Let, the set of all possible character classes is denoted as $\Lambda$;
2   $suspicious\_labels = \Phi$;
3   for each column of the architecture
4    {
5       Perform 5-fold cross-validation of the train dataset using the first layer CNN based model;
6        for each misclassified character/numeral $C$ of the train dataset
7       {
8           if $C$ is incorrectly classified as $D$,
9           {
10              $suspicious\_labels = suspicious\_labels \cup D$;
11          }
12          $suspicious\_labels\_second\_level = \Phi$;
13            Perform 5-fold cross validation of the each of the four quadrants of the train dataset using
    the second layer of stacked CNN based models;
14            if $C_1$ is incorrectly classified as $D_1$,
15            {
```

```
16              $suspicious\_labels\_second\_level = suspicious\_labels\_second\_level \cup D_1$;
17          }
18          $suspicious\_labels\_second\_level = suspicious\_labels - suspicious\_labels\_second\_level$;
19      }
20  }
21  $unsuspicious\_labels = \Lambda - suspicious\_labels$;
22  Perform classification task on the input image I using the first layer CNN based model.
23  Let, the cardinality of the set $\Lambda$ is N i.e. $|\Lambda| = N$ and $P_i$ denotes the prediction score for class label $i, i = 1$ to N.
24  $probable\_label = \underset{i=1\ to\ N}{\operatorname{argmax}} P_i$;
25  if ($probable\_label \epsilon suspicious\_labels$)
26  {
27      for j = 1 to 4
28      {
29          Perform classification task on the quadrants $I_j, j = 1\ to\ 4$ of the input image I using the stacked CNN based models at the second layer. Each smaller network $M_j, j = 1\ to\ 4$ is assigned the task of classifying the quadrant $I_j$ (shown in Table 5.2).Let, $Q_i$ denotes the prediction score for class label $i, i = 1$ to N.
30          $probable\_label_j = \underset{i=1\ to\ N}{\operatorname{argmax}} Q_i$
31      }
32      Perform majority voting on the class labels $probable\_label_j, j = 1$ to 4. Let, the majority assigned label for $I$ is denoted as $probable\_label\_second\_level$.
33      if ($probable\_label\_second\_level \epsilon suspicious\_labels\_second\_level$)
34      {
35          for $i = 1\ to\ 4$
36          {
37              for $j = 1\ to\ 4$
38              {
39                  Perform classification task on the deca-hexadrant $I_{ij}, j = 1\ to\ 4, for\ each\ i = 1\ to\ 4$ of the input image I using the stacked CNN based models at the third layer. Each smaller network $N_j, j = 1\ to\ 16$ is assigned the task of classifying the quadrant $I_{ij}$ (shown in Table 5.3).Let, $R_i$ denotes the prediction score for class label $i, i = 1$ to N.
40                  $probable\_label_{ij} = \underset{i=1\ to\ N}{\operatorname{argmax}} R_i$
41                }
42              }
43        Perform majority voting on the class labels $probable\_label_{ij}, j = 1$ to 4, $for\ each\ i = 1\ to$ 4. Let, the majority assigned label for $I$ is denoted as $probable\_label\_third\_level$.
44    Perform majority voting between the class labels $probable\_label$, $probable\_label\_second\_level$ and $probable\_label\_third\_level$.; Tie cases are handled by assigning that class label which has the greatest prediction score. Let, the majority label assigned to the input image $I$ is $assigned\_label$.
45        return $assigned\_label$;
46          } else {
47              Perform majority voting between the class labels $probable\_label$ and $probable\_label\_second\_level$; Tie cases are handled by assigning that class label which has the greater prediction score. Let, the majority label assigned to the input image $I$ is $assigned\_label$.
48              return $assigned\_label$;
49          }
50      } else {
51    return $probable\_label$;
52    }
```

The methodology described in Algorithm 1proposes a staggered or percolated prediction model which is adopted by each column in the proposed architecture. As mentioned earlier, each of the three columns work parallel towards the recognition of a pattern image. Instead of involving all of the layers of a column, the algorithm essentially tries to come up with a set of character labels which can be potentially misclassified by the first layer CNN based on 5-fold cross-validation performed on the train dataset. The misclassified character labels due to the cross-validation by the first layer CNNs form the potentially vulnerable set of character labels with respect to the first layer of that column. If an input test image is assigned a character label which belongs to these vulnerable set of character classes, the input image is percolated through to the second layer of the column. Here, the image is divided equally into four quadrants and the smaller CNNs belonging to the stacked CNNs of the second layer are tasked to recognize the input image based on the quadrant which is assigned to it. The steps of finding the vulnerable character classes is repeated here again, but instead of processing the entire input image, each of the smaller nets in the stacked CNN perform cross-validations on their assigned quadrants. If the input image is assigned a character class belonging to one of these vulnerable character classes, determined by the stacked CNNs at the second layer of the column, the test image is percolated through to the third and final layer of the column. A stack of 16 smaller convolutional nets get assigned one of the dechexadrants of the original input image and perform their own classification task separately. The class labels are decided upon by performing a majority voting between the assigned labels proposed by each candidate layer. Tie cases are resolved based on the highest confidence score assigned by a prediction model. As mentioned earlier, each of the columns work independent of each other in this architecture. Once a candidate label is put forth by all of the three columns, an aggregation operation is performed on the test dataset and a final class label is retuned by the deep architecture. In our present work, we have experimented with average, weighted average and majority voting techniques as aggregation operations.

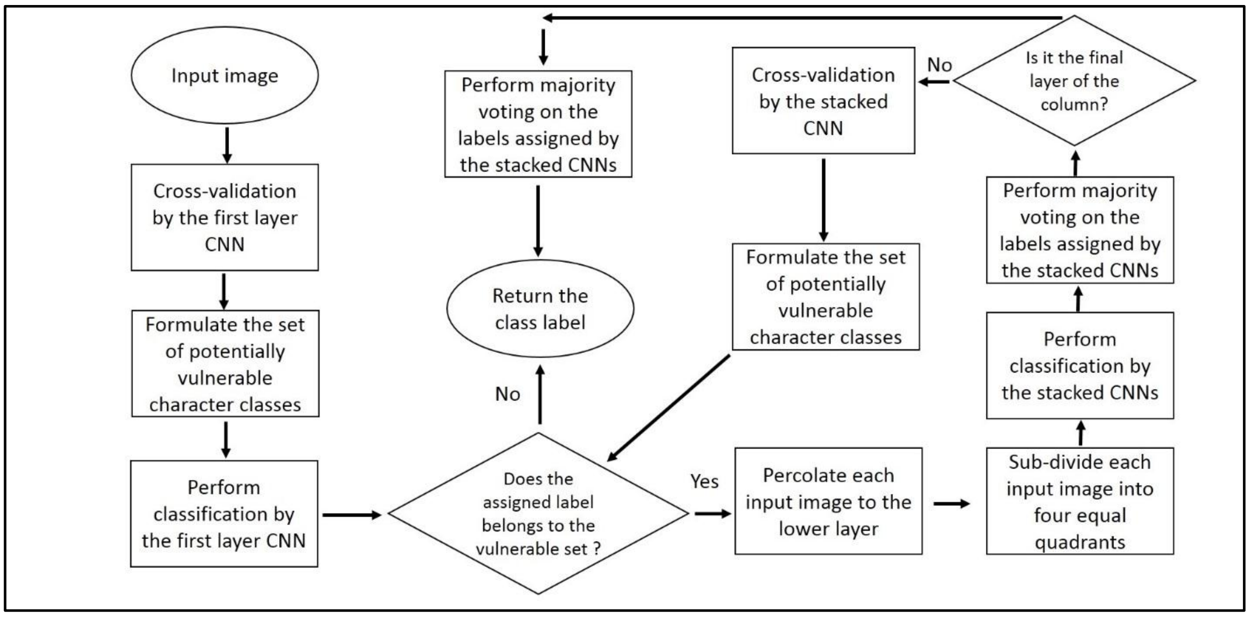

Input image
Cross-validation by the first layer CNN
Formulate the set of potentially vulnerable character classes
Perform classification by the first layer CNN
Does the assigned label belongs to the vulnerable set ?
Yes
No
Return the class label
Perform majority voting on the labels assigned by the stacked CNNs
Percolate each input image to the lower layer
Sub-divide each input image into four equal quadrants
Perform classification by the stacked CNNs
Perform majority voting on the labels assigned by the stacked CNNs
Is it the final layer of the column?
No
Cross-validation by the stacked CNN
Formulate the set of potentially vulnerable character classes

Fig. 5.9: *A schematic diagram of the percolated prediction model adopted in the present work*

Clearly, an input image I which is percolated to the second layer of the architecture is represented not only by the featurevector extracted by the first layer CNN, but the featurevectors extracted by the four smaller nets in the second layer as well. Each of the four smaller convolutional nets in the second layer, extracts a featurevector from a quadrant of the input image (as shown in Table 5.2), assigned to it. The final label assignment is done based on a majority voting of the proposed candidate labels. If the class label assigned by the second layer stacked CNNs belong to a potentially vulnerable class, the input image gets percolated through to the third and final layer of the network, where each of the 16 smaller nets, each are independently tasked with a deca-hexadrant of the original image I. Each of the 16 smaller nets belonging to the third layer, extracts a featurevector from a deca-hexadrant of the input image I (as shown in Table 5.3), assigned to it. The final label assignment is done based on a majority voting between all of the proposed candidate class labels.

This hierarchical structure of feature extraction discussed above, can be visualized as a quad-tree[27][28] structure of depths one or two respectively, as shown in Fig. 5.10. This multi-scale deep quad tree based feature extraction technique has shown promising results on the six publicly available datasets of isolated, handwritten character and numerals of Indian languages, used in our current experimental setup. More details about the experimental results are discussed in the next chapter.

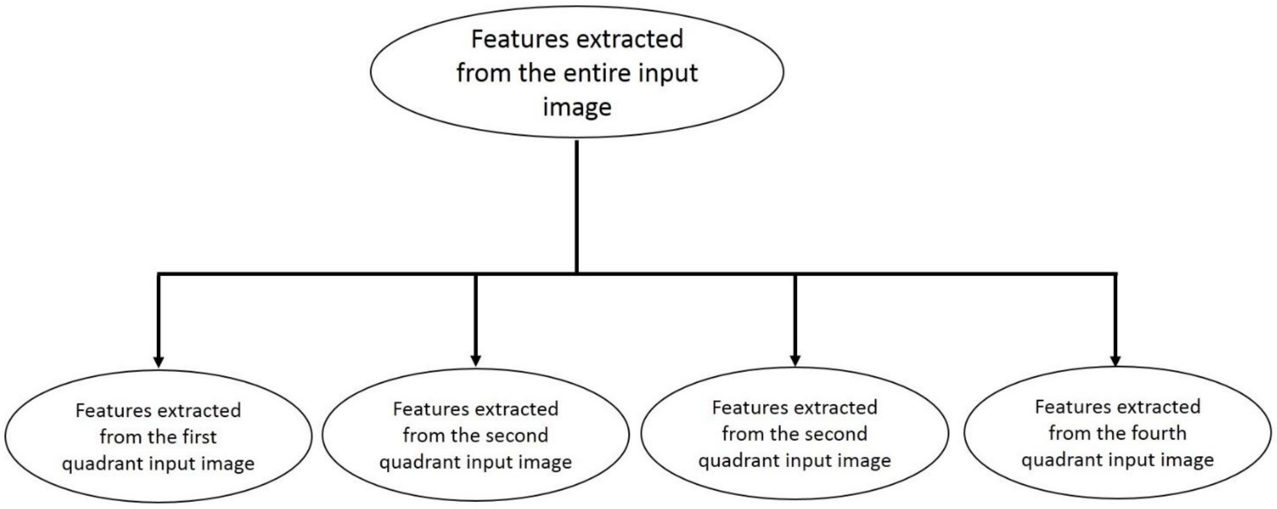

Features extracted from the entire input image
Features extracted from the first quadrant input image
Features extracted from the second quadrant input image
Features extracted from the second quadrant input image
Features extracted from the fourth quadrant input image

Fig. 5.10.1: *A first level quad-tree representation of extracted multi-scale convolutional features*

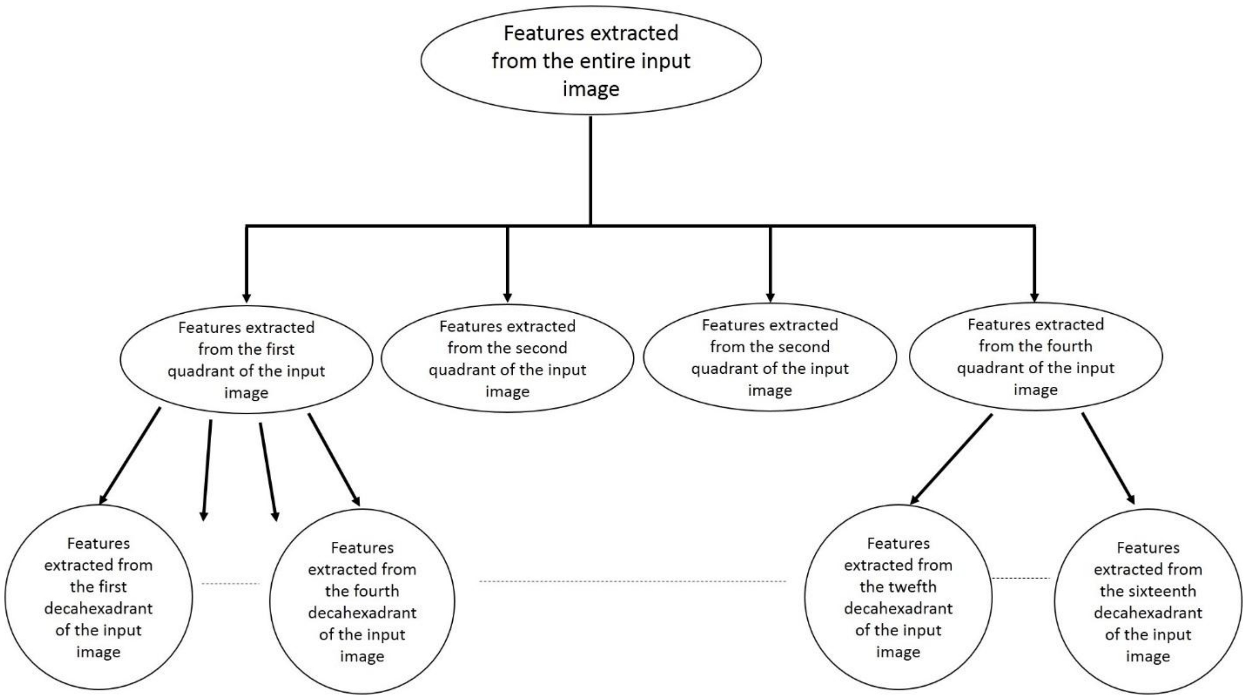

Features extracted from the entire input image
Features extracted from the first quadrant of the input image
Features extracted from the second quadrant of the input image
Features extracted from the second quadrant of the input image
Features extracted from the fourth quadrant of the input image
Features extracted from the first decahexadrant of the input image
Features extracted from the fourth decahexadrant of the input image
Features extracted from the twefth decahexadrant of the input image
Features extracted from the sixteenth decahexadrant of the input image

Fig. 5.10.1: *A second level quad-tree representation of extracted multi-scale convolutional features*

Fig. 5.10: *A multi-scale quad tree representation of the proposed feature extraction scheme*

Chapter 6

Results and discussion

In the present work, a multi-scale deep quad tree based ubiquitous feature extraction technique is proposed. A *multi-scale multi-column convolutional neural network* (MMCNN) based deep architecture is proposed for this purpose. A schematic diagram of the proposed architecture is shown in Fig. 5.2. A *mini-batch gradient descent based learning* algorithm with globally decaying learning rate is used for global learning and *RMSProp* is used for adaptive local learning of network parameters. Parameters of the learning algorithm is shown in Table 5.4. Although several schemes has been taken to increase the convergence rate of the learning algorithm, it still takes a long time to train such a huge network. Besides training each column of the network in parallel, we have also utilized several GPU based apis which use vectorized libraries for faster matrix operations. The Python based *Pylearn2* library is used to perform all of the deep learning based training and testing jobs in our current experimental setup. All of the dependencies of *Pylearn2* including *NumPy*, *SciPy* and *Theano* which were all used throughout the implementation moderately. Basic image processing operations were done using MATLAB and large datasets handling were performed using HDF5 file format through the use of the h5py library. All of the experiments were performed using systems with Intel core i5 processor, 4GB RAM and a NVIDIA GeForce 750 Ti graphics card with 2GB internal memory, having 640 CUDA cores. The resultant speedup in convergence rate of the algorithm using this configuration, as opposed to using a CPU driven system is shown in Fig. 6.1.

In the following section the recognition rate achieved by the proposed system on 3 publicly available datasets of isolated handwritten characters and 3 publicly available datasets of isolated handwritten numerals of Indian languages are reported. Experiments on each dataset have been repeated 25 times and averaged. The average success-rate is reported in this document. Although there are several other works[122][130][129] in the literature which have proposed deep neural networks for the recognition of handwritten characters and/or numerals of Indian languages, to the best of our knowledge, none of those approaches have experimented with such a large number of datasets belonging to different Indian languages. Also the approaches proposed by the researchers in the previous works have generally focused on a specific language. In these two regards, this is the first work which proposes a generic, ubiquitous feature set for handwritten characters and/or numerals belonging to different scripts. The average per character/numeral prediction time taken by the proposed system is also given. Finally, a comparison with several other contemporary methods are provided to prove the superiority of the proposed MMCNN architecture.

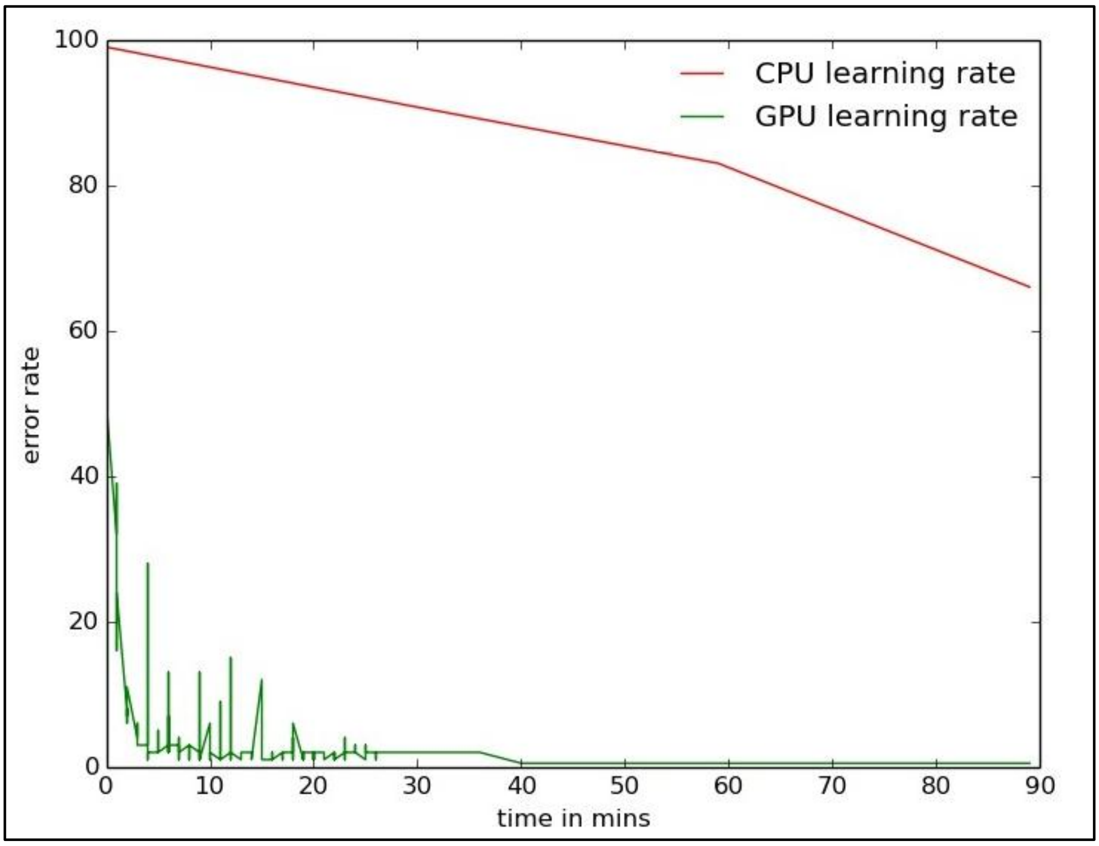


Fig. 6.1: *Comparison between learning rates of CPU and GPU driven learning algorithms*

As mentioned earlier in Section 5.7, each column of the proposed architecture work in parallel, each acting as a separate classifier model. Once all three columns have finished their respective classification, an aggregation operation is done on the results of the columns and a final class label is assigned to the test image. In the present work, we have taken a majority voting based approach towards aggregating the results from each column. From our previous experience with other datasets we have found that the voting based approach provides the best performance compared to other aggregation approaches. A comparison of the proposed approach with two other aggregation methods i.e. average and weighted average, suggested by researchers like Ciresan et al.[98][97] has also been provided. The database indexes are same as shown in

Database index	*Average aggregation scheme (%)*	*Weighted average aggregation scheme (%)*	*Majority voting aggregation scheme (%)*
D1	95.39	95.61	98.12
D2	99.35	99.39	100.00
D3	96.32	96.07	98.52
D4	99.73	99.73	100.00
D5	98.56	98.609	99.50
D6	98.83	98.861	99.50

Table 6.1: *Successrate achieved by the proposed MMCNN based architecture on different datasets*

From the results shown in Table 6.1, it is clear that the proposed majority voting scheme to combine the results from the columns performs much better than other contemporary aggregation schemes. As it can be seen from the results that the proposed method provides promising results across all of the datasets used in our experimental setup. In fact, the recognition accuracy achieved by the proposed method are state-of-the-art for all of the datasets used in the present work. Such promising results proves the efficiency of the method and hence helps prove our hypothesis that the proposed MMCNN architecture provides script independent, ubiquitous feature set.

A comprehensive table of average recognition times taken for each character/numeral by the proposed system is given in Table 6.2.

Database index	*Average per character recognition time (milliseconds)*
D1	19.82
D2	18.66
D3	19.09
D4	18.21
D5	18.48
D6	18.57

Table 6.2: *Average per character/numeral recognition time for the proposed system*

To prove the superiority of the proposed method, performance of the proposed system is compared with some of the popular, contemporary works. The comparative analysis is given in Table 6.3 as below.

Database type	*Work reference*	*Recognition rate (%)*
Bangla Compound Characters	Das et al.[138]	75.05
	Das et al.[3]	87.50
	Sarkhel et al.[16]	78.38
	Sarkhel et al.[17]	86.64
	Pal et al.[129]	93.12
	The present work	98.12

Table 6.3.1

Database index	*Work reference*	*Recognition rate (%)*
Bangla Basic Characters	Roy et al.[57]	86.40
	Das et al.[138]	80.50
	Basu et al.[18]	80.58
	Sarkhel et al.[16]	86.53
	Bhattacharya et al.[146]	92.15
	The present work	100.00

Table 6.3.2

Database index	*Work reference*	*Recognition rate (%)*
Tamil Characters	Sigappi et al.[147]	89.74

Database index	***Work reference***	***Recognition rate (%)***
	Raj et al.[148]	89.00
	Abirami et al.[149]	85.00
	Subashini et al.[150]	87.00
	Bhattachraya et al.[151]	89.66
	Ramakrishnan et al.[152]	95.86
	The present work	98.52

Table 6.3.3

Database index	***Work reference***	***Recognition rate (%)***
Bangla digit	Das et al.[45]	97.80
	Sarkhel et al.[17]	98.23
	Basu et al.[29]	96.67
	Roy et al.[28]	95.08
	Roy et al.[153]	97.45
	Singh et al.[25]	99.30
	The present work	100.00

Table 6.3.4

Database index	***Work reference***	***Recognition rate (%)***
Hindi digit	Acharya et al.[154]	98.47
	Sarkar et al.[155]	98.20
	Das et al.[138]	90.44
	Singh et al.[25]	99.50
	Roy et al.[153]	96.50
	The present work	99.50

Table 6.3.5

Database index	***Work reference***	***Recognition rate (%)***
Telugu digit	Rajasheraradhya et al.[56]	96.0
	Rajasheraradhya et al.[156]	99.0
	Singh et al.[25]	98.8
	Roy et al.[153]	87.2
	Sarkhel et al.[16]	97.5
	The present work	99.5

Table 6.3.6

Table 6.3: *A comparative analysis of the proposed system against some of the contemporary OCR methods*

Conclusion

A non-explicit, script independent, ubiquitous feature extraction technique for the recognition of handwritten Indic characters and numerals has been proposed in the present work. A multi-scale, multi-column convolutional neural network (MMCNN) based architecture has been proposed for this purpose. A mini-batch gradient descent based global learning algorithm with RMSProp for local learning of the parameters is adopted to train the proposed deep architecture. GPU based apis which utilize vectorized libraries is used for training the network. The proposed method leads to faster convergence, better generalization and allows for effective usage of labelled data. A staggered or percolated prediction model has been taken up to assign class labels to a test image. The experimental results performed on six publicly available benchmark datasets of isolated, handwritten Indian characters and numerals show that the proposed method provides state-of-the-art results for all of the datasets. Superiority of the proposed method against some of the other competitive contemporary methods has also been statistically proved.

The proposed deep architecture based non-explicit, script independent approach towards largely varying, complex, handwritten Indian characters and numerals detection technique marks the contribution of the present work. The proposed script independent generic approach towards the recognition of handwritten, multi-lingual Indian characters opens up a new avenue in research. Significantly low error rate and better performance than its contemporaries marks it as an important contribution towards the OCR research community. It get rids of proposing pesky features, tailored to particularities of a specific language and provides an all-encompassing method for the recognition of a wide array of handwritten characters and numerals belonging to different Indian languages. In future, the proposed architecture can be tested for more datasets. This avenue can also be explored further as deeper and wider networks can be employed to further increase the performance of the OCR system. However, it should be remembered that extracting features from larger networks take a long time. Hence, a perfect balance should be maintained between the increase in average per character prediction time and the gain in recognition performance in result.

References

[1] B. . Chaudhuri and U. Pal, “A complete printed Bangla OCR system,” *Pattern Recognit.*, vol. 31, no. 5, pp. 531–549, Mar. 1998.

[2] U. Pal and B. B. Chaudhuri, “Indian script character recognition: a survey,” *Pattern Recognit.*, vol. 37, no. 9, pp. 1887–1899, Sep. 2004.

[3] N. Das, R. Sarkar, S. Basu, P. K. Saha, M. Kundu, and M. Nasipuri, “Handwritten Bangla character recognition using a soft computing paradigm embedded in two pass approach.,” *Pattern Recognit.*, vol. 48, no. 6, pp. 2054–2071, Dec. 2014.

[4] R. Plamondon and S. N. Srihari, “Online and off-line handwriting recognition: a comprehensive survey,” *Pattern Analysis and Machine …*, vol. 22, no. 1. pp. 63–84, 2000.

[5] F. Coulmas, *The writing systems of the world*. B. Blackwell, 1989.

[6] R. Plamondon, “A renaissance for handwriting,” *Mach. Vis. Appl.*, vol. 8, no. 4, pp. 195–196, 1995.

[7] J. Plaice, “‘The Unicode® Consortium,” 1995.

[8] Mantas J., “An overview of character recognition methodologies,” *Pattern Recognit.*, no. 19, pp. 425–430, 1986.

[9] S. Mori, C. Y. C. Y. Suen, and K. Yamamoto, “Historical review of OCR research and development,” *Proc. IEEE*, vol. 80, no. 7, pp. 1029–1058, Jul. 1992.

[10] S. Mori, K. Yamamoto, and M. Yasuda, “Research on machine recognition of handprinted characters,” *Pattern Anal. Mach. Intell. IEEE Trans.*, no. 4, pp. 386–405, 1984.

[11] W. Stallings, “Approaches to Chinese character recognition,” *Pattern Recognit.*, vol. 8, no. 2, pp. 87–98, Apr. 1976.

[12] A. Ul-Hasan, S. Bin Ahmed, F. Rashid, F. Shafait, and T. M. Breuel, “Offline Printed Urdu Nastaleeq Script Recognition with Bidirectional LSTM Networks,” in *2013 12th International Conference on Document Analysis and Recognition*, 2013, pp. 1061–1065.

[13] G. Nagy, “Chinese character recognition: a twenty-five-year retrospective,” in *Pattern Recognition, 1988., 9th International Conference on*, 1988, pp. 163–167.

[14] L. O’Gorman and R. Kasturi, *Document image analysis*, vol. 39. IEEE Computer Society Press Los Alamitos, CA, 1995.

[15] J. Rocha and T. Pavlidis, “Character recognition without segmentation,” *Pattern Anal. Mach. Intell. IEEE Trans.*, vol. 17, no. 9, pp. 903–909, 1995.

[16] R. Sarkhel, A. Saha, and N. Das, “An Enhanced Harmony Search Method for Bangla Handwritten Character Recognition Using Region Sampling,” in *The 2nd IEEE International Conference on Recent Trends in Information Systems (ReTIS-15)*, 2015, p. (accepted for publication).

[17] R. Sarkhel, N. Das, A. K. Saha, and M. Nasipuri, “A multi-objective approach towards cost effective isolated handwritten Bangla character and digit recognition,” *Pattern Recognit.*, 2016.

[18] S. Basu, N. Das, R. Sarkar, M. Kundu, M. Nasipuri, and D. K. Basu, “A hierarchical approach to recognition of handwritten Bangla characters,” *Pattern Recognit.*, vol. 42, no. 7, pp. 1467–1484, Jul. 2009.

[19] S. Wendling and G. Stamon, “Hadamard and Haar transforms and their power spectrum in character recognition,” in *Joint Workshop on Pattern Recognition andArtfIcial Intelligence, Hyannis, USA*, 1976, pp. 103–112.

[20] S. A. Mahmoud, “Arabic character recognition using Fourier descriptors and character contour encoding,” *Pattern Recognit.*, vol. 27, no. 6, pp. 815–824, 1994.

[21] J. S. Huang and M.-L. Chung, “Separating similar complex Chinese characters by Walsh transform,” *Pattern Recognit.*, vol. 20, no. 4, pp. 425–428, 1987.

[22] M. Kushnir, K. Abe, and K. Matsumoto, “An application of the Hough transform to the recognition of printed Hebrew characters,” *Pattern Recognit.*, vol. 16, no. 2, pp. 183–191, 1983.

[23] R. M. K. Sinha and H. N. Mahabala, “Machine recognition of Devanagari script,” *IEEE Trans. Syst. Man Cybern*, vol. 9, no. 8, pp. 435–441, 1979.

[24] H.-Y. Feng and T. Pavlidis, “Decomposition of polygons into simpler components: feature generation for syntactic pattern recognition,” *IEEE Trans. Comput.*, no. 6, pp. 636–650, 1975.

[25] P. K. Singh, R. Sarkar, and M. Nasipuri, “A Study of Moment Based Features on Handwritten Digit Recognition,” vol. 2016, 2016.

[26] S. Kahan, T. Pavlidis, and H. S. Baird, “On the recognition of printed characters of any font and size,” *Pattern Anal. Mach. Intell. IEEE Trans.*, no. 2, pp. 274–288, 1987.

[27] N. Das, S. Basu, R. Sarkar, M. Kundu, M. Nasipuri, and D. kumar Basu, “An Improved Feature Descriptor for Recognition of Handwritten Bangla Alphabet,” in *ICSIP 2009*, 2009, pp. 451–454.

[28] A. Roy, N. Mazumder, N. Das, R. Sarkar, S. Basu, and M. Nasipuri, “A new quad tree based feature set for recognition of handwritten bangla numerals,” in *AICERA 2012 - Annual International Conference on Emerging Research Areas: Innovative Practices and Future Trends*, 2012, pp. 1–6.

[29] S. Basu, N. Das, R. Sarkar, M. Kundu, M. Nasipuri, and D. K. Basu, “An MLP based Approach for Recognition of Handwritten `Bangla’ Numerals,” pp. 407–417, Mar. 2012.

[30] U. Bhattacharya, T. K. Das, A. Datta, S. K. Parui, and B. B. Chaudhuri, “A hybrid scheme for handprinted numeral recognition based on a self-organizing network and MLP classifiers,” *Int. J. pattern Recognit. Artif. Intell.*, vol. 16, no. 07, pp. 845–864, 2002.

[31] Y. LeCun and Y. Bengio, “Word-level training of a handwritten word recognizer based on convolutional neural networks,” in *International Conference on Pattern Recognition*, 1994, p. 88.

[32] J. D. Keeler and D. E. Rumelhart, “Self-organizing integrated segmentation and recognition neural network,” in *Aerospace Sensing*, 1992, pp. 744–755.

[33] A. de S. Britto Jr, R. Sabourin, F. Bortolozzi, and C. Y. Suen, “The recognition of handwritten numeral strings using a two-stage HMM-based method,” *Int. J. Doc. Anal. Recognit.*, vol. 5, no. 2–3, pp. 102–117, 2003.

[34] A. Vinciarelli, S. Bengio, and H. Bunke, “Offline recognition of unconstrained handwritten texts using HMMs and statistical language models.,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 26, no. 6, pp. 709–20, Jun. 2004.

[35] S. K. Parui, K. Guin, U. Bhattacharya, and B. B. Chaudhuri, “Online Handwritten Bangla Character Recognition Using HMM 3 . Analysis of strokes in handwritten,” *Ieee*, pp. 1–4, 2008.

[36] A. K. Datta, “A Generalized Formal Approach for Description and Analysis of Major Indian Scripts,” *IETE J. Res.*, vol. 30, no. 6, pp. 155–161, 1984.

[37] R. M. K. Sinha, “A syntactic pattern analysis system and its application to Devnagari script recognition.” Ph. D. Thesis, Electrical Engineering Department, Indian Institute of Technology, India, 1973.

[38] I. K. Sethi and B. Chatterjee, “Machine recognition of constrained hand printed Devanagari,” *Pattern Recognit.*, vol. 9, no. 2, pp. 69–75, 1977.

[39] R. Bajaj, L. Dey, and S. Chaudhury, “Devnagari numeral recognition by combining decision of multiple connectionist classifiers,” *Sadhana*, vol. 27, no. 1, pp. 59–72, 2002.

[40] H. B. Kekre, S. D. Thepade, S. P. Sanas, and S. Shinde, “Devnagari Handwritten Character Recognition using LBG vector quantization with gradient masks,” in *Advances in Technology and Engineering (ICATE), 2013 International Conference on*, 2013, pp. 1–4.

[41] K. R. Shah and D. D. Badgujar, “Devnagari handwritten character recognition (DHCR) for ancient documents: A review,” in *Information & Communication Technologies (ICT), 2013 IEEE Conference on*, 2013, pp. 656–660.

[42] A. K. Ray and B. Chatterjee, “Design of a nearest neighbour classifier system for Bengali character recognition,” *IETE J. Res.*, vol. 30, no. 6, pp. 226–229, 1984.

[43] S. K. Parui, B. B. Chaudhuri, and D. Dutta Majumder, “A procedure for recognition of connected handwritten numerals,” *Int. J. Syst. Sci.*, vol. 13, no. 9, pp. 1019–1029, 1982.

[44] U. Garain and B. B. Chaudhuri, “Compound character recognition by run-number-based metric distance,” in *Photonics West’98 Electronic Imaging*, 1998, pp. 90–97.

[45] N. Das, R. Sarkar, S. Basu, M. Kundu, M. Nasipuri, and D. K. Basu, “A genetic algorithm based region sampling for selection of local features in handwritten digit recognition application,” *Appl. Soft Comput.*, vol. 12, no. 5, pp. 1592–1606, May 2012.

[46] H. A. Khan, A. Al Helal, and K. I. Ahmed, “Handwritten Bangla digit recognition using Sparse Representation Classifier,” in *2014 International Conference on Informatics, Electronics & Vision (ICIEV)*, 2014, pp. 1–6.

[47] G. Siromoney, R. Chandrasekaran, and M. Chandrasekaran, “Computer recognition of printed Tamil characters,” *Pattern Recognit.*, vol. 10, no. 4, pp. 243–247, 1978.

[48] R. Chandrasekaran, M. Chandrasekaran, and G. Siromoney, “Computer Recognition of Tamil, Malayalam and Devanagari Characters,” *IETE J. Res.*, Jul. 2015.

[49] J. Sutha and N. Ramaraj, “Neural Network Based Offline Tamil Handwritten Character Recognition System,” *Int. Conf. Comput. Intell. Multimed. Appl. (ICCIMA 2007)*, pp. 446–450, 2007.

[50] P. Chinnuswamy and S. G. Krishnamoorthy, “Recognition of handprinted Tamil characters,” *Pattern Recognit.*, vol. 12, no. 3, pp. 141–152, 1980.

[51] S. N. S. Rajasekaran and B. L. Deekshatulu, “Recognition of printed Telugu characters,” *Comput. Graph. image Process.*, vol. 6, no. 4, pp. 335–360, 1977.

[52] M. B. Sukhaswami, P. Seetharamulu, and A. K. Pujari, “Recognition of Telugu characters using neural networks,” *Int. J. Neural Syst.*, vol. 6, no. 03, pp. 317–357, 1995.

[53] A. Srinivas, A. Agarwal, and C. R. Rao, “Telugu Character Recognition,” in *International Conference on Systemics, Cybernetics, and Informatics, Hyderabad, India*, 2007, pp. 654–659.

[54] A. Negi, C. Bhagvati, and B. Krishna, “An OCR system for Telugu,” in *Document Analysis*

and Recognition, 2001. Proceedings. Sixth International Conference on, 2001, pp. 1110–1114.

[55] D. Singh and B. S. Khehra, “Digit recognition system using back propagation neural network,” *Int. J. Comput. Sci. Commun.*, vol. 2, no. 1, pp. 197–205, 2011.

[56] S. V. Rajashekararadhya and P. Vanaja Ranjan, “Neural network based handwritten numeral recognition of Kannada and Telugu scripts,” in *TENCON 2008 - 2008 IEEE Region 10 Conference*, 2008, pp. 1–5.

[57] A. Roy, N. Das, R. Sarkar, S. Basu, M. Kundu, and M. Nasipuri, “Region Selection in Handwritten Character Recognition using Artificial Bee Colony Optimization,” in *Third International Conference on Emerging Applications of Information Technology (EAIT- 2012)*, 2012, pp. 183–186.

[58] T. T. O.D.Trier, A.K.Jain, “Feature extraction methods for character recognition - a survey,” *Pattern Recognit.*, vol. 29, no. 4, pp. 641–662, 1996.

[59] D. Impedovo and G. Pirlo, “Zoning methods for handwritten character recognition : A survey,” *Pattern Recognit.*, vol. 47, no. 3, pp. 969–981, 2014.

[60] W. K. Pratt, “Digital Image Processing, New-York,” *NY John Wiley Sons*, 1991.

[61] A. del Bimbo, S. Santini, and J. Sanz, “OCR from poor quality images by deformation of elastic templates,” in *Pattern Recognition, 1994. Vol. 2-Conference B: Computer Vision &amp; Image Processing., Proceedings of the 12th IAPR International. Conference on*, 1994, vol. 2, pp. 433–435.

[62] H. C. Andrews, “Multidimensional rotations in feature selection,” *Comput. IEEE Trans.*, vol. 100, no. 9, pp. 1045–1051, 1971.

[63] W. Gonzalez and R. E. Woods, “Eddins, Digital Image Processing Using MATLAB,” *Third New Jersey Prentice Hall*, 2004.

[64] M. Bokser, “Omnidocument technologies,” in *Proceedings of the IEEE*, 1992, vol. 80, no. 7, pp. 1066–1078.

[65] M.-K. Hu, “Visual pattern recognition by moment invariants,” *Inf. Theory, IRE Trans.*, vol. 8, no. 2, pp. 179–187, 1962.

[66] S. O. Belkasim, M. Shridhar, and M. Ahmadi, “Pattern recognition with moment invariants: a comparative study and new results,” *Pattern Recognit.*, vol. 24, no. 12, pp. 1117–1138, 1991.

[67] T. H. Reiss, *Recognizing planar objects using invariant image features*. Springer-Verlag New York, Inc., 1993.

[68] A. Khotanzad and Y. H. Hong, “Invariant image recognition by Zernike moments,” *Pattern Anal. Mach. Intell. IEEE Trans.*, vol. 12, no. 5, pp. 489–497, 1990.

[69] A. Khotanzad and Y. H. Hong, “Rotation invariant image recognition using features selected via a systematic method,” *Pattern Recognit.*, vol. 23, no. 10, pp. 1089–1101, 1990.

[70] J. D. Tubbs, “A note on binary template matching,” *Pattern Recognit.*, vol. 22, no. 4, pp. 359–365, 1989.

[71] R. Kasturi, *Image analysis applications*, vol. 24. CRC Press, 1990.

[72] S. Arora, D. Bhattacharjee, M. Nasipuri, L. Malik, M. Kundu, and D. K. Basu, “Performance Comparison of SVM and ANN for Handwritten Devnagari Character Recognition,” pp. 1–10, 2010.

[73] N. Das, S. Pramanik, R. Sarkar, S. Basu, and P. K. Saha, “Recognition of Isolated Multi-Oriented Handwritten/Printed Characters using a Novel Convex-Hull Based Alignment

Technique," *Int. J. Comput. Appl.*, vol. 1, no. 23, pp. 40–45, 2010.

[74] F. Kimura and M. Shridhar, "Handwritten numerical recognition based on multiple algorithms," *Pattern Recognit.*, vol. 24, no. 10, pp. 969–983, 1991.

[75] H. Takahashi, "A neural net OCR using geometrical and zonal pattern features," in *Proc. 1st Intl. Conf. on Document Analysis and Recognition,* 1991, pp. 821–828.

[76] I. Sekita, K. Toraichi, R. Mori, K. Yamamoto, and H. Yamada, "Feature extraction of handwritten Japanese characters by spline functions for relaxation matching," *Pattern Recognit.*, vol. 21, no. 1, pp. 9–17, 1988.

[77] T. Taxt, J. B. Ólafsdóttir, and M. Dæhlen, "Recognition of handwritten symbols," *Pattern Recognit.*, vol. 23, no. 11, pp. 1155–1166, 1990.

[78] F. P. Kuhl and C. R. Giardina, "Elliptic Fourier features of a closed contour," *Comput. Graph. image Process.*, vol. 18, no. 3, pp. 236–258, 1982.

[79] C.-S. Lin and C.-L. Hwang, "New forms of shape invariants from elliptic Fourier descriptors," *Pattern Recognit.*, vol. 20, no. 5, pp. 535–545, 1987.

[80] L. Wang and T. Pavlidis, "Detection of curved and straight segments from gray scale topography," *CVGIP Image Underst.*, vol. 58, no. 3, pp. 352–365, 1993.

[81] L. Wang and T. Pavlidis, "Direct gray-scale extraction of features for character recognition," *Pattern Anal. Mach. Intell. IEEE Trans.*, vol. 15, no. 10, pp. 1053–1067, 1993.

[82] D. J. Burr, "Elastic matching of line drawings," *IEEE Trans. Pattern Anal. Mach. Intell.*, no. 6, pp. 708–713, 1981.

[83] T. Wakahara, "Shape matching using LAT and its application to handwritten numeral recognition," *Pattern Anal. Mach. Intell. IEEE Trans.*, vol. 16, no. 6, pp. 618–629, 1994.

[84] T. Wakahara, "Toward robust handwritten character recognition," *Pattern Recognit. Lett.*, vol. 14, no. 4, pp. 345–354, 1993.

[85] A. Kundu, Y. He, and P. Bahl, "Recognition of handwritten word: first and second order hidden Markov model based approach," in *Computer Vision and Pattern Recognition, 1988. Proceedings CVPR'88., Computer Society Conference on*, 1988, pp. 457–462.

[86] S. R. Ramesh, "A generalized character recognition algorithm: a graphical approach," *Pattern Recognit.*, vol. 22, no. 4, pp. 347–350, 1989.

[87] E. Holbaek-Hanssen, K. Bråthen, and T. Taxt, "A general software system for supervised statistical classification of symbols," in *Proc. 8th Intl Conf. Pattern Recognition, Paris, France (October 1986),* 1986, pp. 144–149.

[88] J. J. Hull and S. N. Srihari, "Experiments in text recognition with binary n-gram and viterbi algorithms," *Pattern Anal. Mach. Intell. IEEE Trans.*, no. 5, pp. 520–530, 1982.

[89] T. K. Bhowmik, S. K. Parui, U. Bhattacharya, and B. Shaw, "An HMM based recognition scheme for handwritten Oriya numerals," in *Information Technology, 2006. ICIT'06. 9th International Conference on,* 2006, pp. 105–110.

[90] H. A. Al-Muhtaseb, S. A. Mahmoud, and R. S. Qahwaji, "Recognition of off-line printed Arabic text using Hidden Markov Models," *Signal Processing,* vol. 88, no. 12, pp. 2902–2912, Dec. 2008.

[91] Z. Chi, J. Wu, and H. Yan, "Handwritten numeral recognition using self-organizing maps and fuzzy rules," *Pattern Recognit.*, vol. 28, no. 1, pp. 59–66, 1995.

[92] K. Keeni, H. Shimodaira, T. Nishino, and T. Yasuo, "Recognition of Devanagari Characters

Using Neural Networks," *IEICE Trans. Inf. Syst.*, vol. 79, no. 5, pp. 523–528, 1996.

[93] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural Comput.*, vol. 1, no. 4, pp. 541–551, 1989.

[94] J. Cao, M. Ahmadi, and M. Shridhar, "Recognition of handwritten numerals with multiple feature and multistage classifier," *Pattern Recognit.*, vol. 28, no. 2, pp. 153–160, Feb. 1995.

[95] A. Dutta and S. Chaudhury, "Bengali alpha-numeric character recognition using curvature features," *Pattern Recognit.*, vol. 26, no. 12, pp. 1757–1770, 1993.

[96] S. Basu, N. Das, R. Sarkar, M. Kundu, M. Nasipuri, and D. K. Basu, "Handwritten ' Bangla ' alphabet recognition using an MLP based classfier," in *2nd National Conf. on Computer Processing of Bangla-2005*, 2005, pp. 285–291.

[97] D. Ciresan and U. Meier, "Multi-column Deep Neural Networks for Image Classification," pp. 3642–3649, 2012.

[98] D. C. Cireşan and J. Schmidhuber, "Multi-Column Deep Neural Networks for Offline Handwritten Chinese Character Classification," *arXiv Prepr. arXiv1309.0261*, no. 3755, p. 5, 2013.

[99] A. K. Jain, J. Mao, and K. M. Mohiuddin, "Artificial neural networks: A tutorial," *Computer (Long. Beach. Calif).*, no. 3, pp. 31–44, 1996.

[100] Y. Chu, *High-level language computer architecture*. Academic Press, 2014.

[101] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *Bull. Math. Biophys.*, vol. 5, no. 4, pp. 115–133, 1943.

[102] M. Minsky and S. Papert, *Perceptrons*. MIT press, 1988.

[103] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," *Proc. Natl. Acad. Sci.*, vol. 79, no. 8, pp. 2554–2558, 1982.

[104] P. J. Werbos, "Neurocontrol and supervised learning: An overview and evaluation," *Handb. Intell. Control*, vol. 65, p. 89, 1992.

[105] D. E. Rumelhart, R. Durbin, R. Golden, and Y. Chauvin, "Backpropagation: The basic theory," *Backpropagation Theory, Archit. Appl.*, pp. 1–34, 1995.

[106] "Artificial neural network - Knowino." [Online]. Available: http://www.tau.ac.il/~tsirel/dump/Static/knowino.org/wiki/Artificial_neural_network.html. [Accessed: 16-May-2016].

[107] K. Hornik, "Approximation capabilities of multilayer feedforward networks," *Neural networks*, vol. 4, no. 2, pp. 251–257, 1991.

[108] S. Hochreiter, "The vanishing gradient problem during learning recurrent neural nets and problem solutions," *Int. J. Uncertainty, Fuzziness Knowledge-Based Syst.*, vol. 6, no. 02, pp. 107–116, 1998.

[109] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," *Adv. Neural Inf. Process. Syst.*, pp. 1–9, 2012.

[110] Y. Bengio, "Learning Deep Architectures for AI," 2008.

[111] Y. Bengio and O. Delalleau, "On the expressive power of deep architectures," in *Algorithmic Learning Theory*, 2011, pp. 18–36.

[112] E. Mendelson, *Introduction to mathematical logic*. CRC press, 2009.

[113] I. Wegener, “The complexity of symmetric boolean functions,” in *Computation theory and logic*, Springer, 1987, pp. 433–442.

[114] J. Hastad, “Almost optimal lower bounds for small depth circuits,” in *Proceedings of the eighteenth annual ACM symposium on Theory of computing*, 1986, pp. 6–20.

[115] A. C.-C. Yao, “Separating the polynomial-time hierarchy by oracles,” in | *26th Annual Symposium on Foundations of Computer Science*, 1985, pp. 1–10.

[116] D. H. Hubel and T. N. Wiesel, “Receptive fields of single neurones in the cat’s striate cortex,” *J. Physiol.*, vol. 148, no. 3, pp. 574–591, 1959.

[117] D. H. Hubel and T. N. Wiesel, “Receptive fields and functional architecture in two nonstriate visual areas (18 and 19) of the cat,” *J. Neurophysiol.*, vol. 28, no. 2, pp. 229–289, 1965.

[118] K. Fukushima, “Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position,” *Biol. Cybern.*, vol. 36, no. 4, pp. 193–202, 1980.

[119] Y. LeCun and Y. Bengio, “Convolutional networks for images, speech, and time series,” *Handb. brain theory neural networks*, vol. 3361, no. 10, p. 1995, 1995.

[120] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.

[121] A. Graves, A. Mohamed, and G. Hinton, “Speech recognition with deep recurrent neural networks,” in *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, 2013, pp. 6645–6649.

[122] A. Ray, S. Rajeswar, and S. Chaudhury, “Text recognition using deep BLSTM networks,” in *2015 Eighth International Conference on Advances in Pattern Recognition (ICAPR)*, 2015, pp. 1–6.

[123] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle, “Greedy layer-wise training of deep networks,” *Adv. Neural Inf. Process. Syst.*, vol. 19, p. 153, 2007.

[124] G. E. Hinton, S. Osindero, and Y.-W. Teh, “A fast learning algorithm for deep belief nets,” *Neural Comput.*, vol. 18, no. 7, pp. 1527–1554, 2006.

[125] G. E. Hinton and R. R. Salakhutdinov, “Reducing the dimensionality of data with neural networks,” *Science (80-.).*, vol. 313, no. 5786, pp. 504–507, 2006.

[126] G. E. Hinton, M. Revow, and P. Dayan, “Recognizing handwritten digits using mixtures of linear models,” *Adv. Neural Inf. Process. Syst.*, pp. 1015–1022, 1995.

[127] H. Schwenk and Y. Bengio, “Adaptive boosting of neural networks for character recognition,” *Adv. Neural Inf. Process.*, vol. 10, 1997.

[128] H. Schwenk and M. Milgram, “Transformation Invariant Autoassociation with Application to Handwritten Character Recognition.,” *Adv. Neural Inf. Process. Syst.*, pp. 991–998, 1995.

[129] A. Pal and J. D. Pawar, “Recognition of online handwritten Bangla characters using hierarchical system with Denoising Autoencoders,” in *Computation of Power, Energy Information and Commuincation (ICCPEIC), 2015 International Conference on*, 2015, pp. 47–51.

[130] A. Pal, “Bengali handwritten numeric character recognition using denoising autoencoders,” in *Engineering and Technology (ICETECH), 2015 IEEE International Conference on*, 2015, pp. 1–6.

[131] M. Wang, Y. Chen, and X. Wang, “Recognition of Handwritten Characters in Chinese Legal

Amounts by Stacked Autoencoders," in *2014 22nd International Conference on Pattern Recognition (ICPR)*, 2014, pp. 3002–3007.

[132] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[133] T. Serre, L. Wolf, and T. Poggio, "Object recognition with features inspired by visual cortex," in *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, 2005, vol. 2, pp. 994–1000.

[134] R. Sarkar, N. Das, S. Basu, M. Kundu, M. Nasipuri, and D. K. Basu, "CMATERdb1: a database of unconstrained handwritten Bangla and Bangla–English mixed script document image," *Int. J. Doc. Anal. Recognit.*, vol. 15, no. 1, pp. 71–83, Feb. 2011.

[135] "cmaterdb - CMATERdb: The pattern recognition database repository - Google Project Hosting." [Online]. Available: https://code.google.com/p/cmaterdb/. [Accessed: 31-Jan-2015].

[136] Andrej Karpathy, "CS231n Convolutional Neural Networks for Visual Recognition." [Online]. Available: http://cs231n.github.io/convolutional-networks/#conv. [Accessed: 20-May-2016].

[137] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, 2010, pp. 807–814.

[138] N. Das, B. Das, R. Sarkar, S. Basu, and M. Kundu, "Handwritten Bangla Basic and Compound character recognition using MLP and SVM classifier," *J. Comput.*, vol. 2, no. 2, pp. 109–115, 2010.

[139] R. Sarkar, N. Das, S. Basu, M. Kundu, and M. Nasipuri, "CMATERdb1 : a database of unconstrained handwritten Bangla and Bangla – English mixed script document image," pp. 71–83, 2012.

[140] A. S. Bhaskarabhatla and S. Madhvanath, "Experiences in Collection of Handwriting Data for Online Handwriting Recognition in Indic Scripts.," in *LREC*, 2004.

[141] N. Das, J. M. Reddy, R. Sarkar, S. Basu, M. Kundu, M. Nasipuri, and D. K. Basu, "A statistical–topological feature combination for recognition of handwritten numerals," *Appl. Soft Comput.*, vol. 12, no. 8, pp. 2486–2495, Aug. 2012.

[142] R. Girshick, J. Donahue, T. Darrell, U. C. Berkeley, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," *Cvpr'14*, pp. 2–9, 2014.

[143] Y. Gong, L. Wang, R. Guo, and S. Lazebnik, "Multi-Scale Orderless Pooling of," pp. 1–17.

[144] Y. A. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller, "Efficient backprop," in *Neural networks: Tricks of the trade*, Springer, 2012, pp. 9–48.

[145] M. Riedmiller and H. Braun, "A direct adaptive method for faster backpropagation learning: The RPROP algorithm," in *Neural Networks, 1993., IEEE International Conference on*, 1993, pp. 586–591.

[146] U. Bhattacharya, M. Shridhar, and S. K. Parui, "On recognition of handwritten Bangla characters," in *Computer Vision, Graphics and Image Processing*, Springer, 2006, pp. 817–828.

[147] A. N. Sigappi and S. Palanivel, "AANN-Based Online Handwritten Tamil Character Recognition," in *Recent Advancements in System Modelling Applications*, Springer, 2013, pp. 35–42.

[148] M. A. R. Raj and S. Abirami, "Offline Tamil handwritten character recognition using statistical features," *Adv. Nat. Appl. Sci.*, vol. 9, no. 6 SE, pp. 367–375, Jun. 2015.

[149] S. Abirami, V. Essakiammal, and R. Baskaran, “Statistical features based character recognition for offline handwritten Tamil document images using HMM,” *Int. J. Comput. Vis. Robot.*, Oct. 2015.

[150] A. Subashini and N. D. Kodikara, “A novel SIFT-based codebook generation for handwritten Tamil character recognition,” in *Industrial and Information Systems (ICIIS), 2011 6th IEEE International Conference on*, 2011, pp. 261–264.

[151] U. Bhattacharya, S. K. Ghosh, and S. K. Parui, “A two stage recognition scheme for handwritten Tamil characters,” in *Document Analysis and Recognition, 2007. ICDAR 2007. Ninth International Conference on*, 2007, vol. 1, pp. 511–515.

[152] A. G. Ramakrishnan and K. B. Urala, “Global and local features for recognition of online handwritten numerals and Tamil characters,” in *Proceedings of the 4th International Workshop on Multilingual OCR - MOCR ’13*, 2013, p. 1.

[153] A. Roy, N. Das, R. Sarkar, S. Basu, and M. Kundu, “An Axiomatic Fuzzy Set Theory Based Feature Selection Methodology for Handwritten Numeral Recognition.”

[154] S. Acharya, A. K. Pant, and P. K. Gyawali, “Deep Learning Based Large Scale Handwritten Devanagari Character Recognition.”

[155] A. Sarkar, K. Singh, and A. Mukerjee, “Handwritten Hindi Numerals Recognition System.”

[156] S. V Rajashekararadhya and P. V. Ranjan, “Efficient Zone Based Feature Extraction Algorithm for Handwritten Numeral Recognition of Four Popular South Indian Scripts,” *J. Theor. Appl. Inf. Technol.*, vol. 4, no. 12, pp. 1171–1181, 2008.