University of Windsor
Scholarship at UWindsor

Electronic Theses and Dissertations | Theses, Dissertations, and Major Papers

2012

Online Devanagari Handwritten Character Recognition

Shruthi Sreedhar Kubatur
University of Windsor

Follow this and additional works at: https://scholar.uwindsor.ca/etd

Recommended Citation

Kubatur, Shruthi Sreedhar, "Online Devanagari Handwritten Character Recognition" (2012). *Electronic Theses and Dissertations*. 4822.
https://scholar.uwindsor.ca/etd/4822

This online database contains the full-text of PhD dissertations and Masters' theses of University of Windsor students from 1954 forward. These documents are made available for personal study and research purposes only, in accordance with the Canadian Copyright Act and the Creative Commons license—CC BY-NC-ND (Attribution, Non-Commercial, No Derivative Works). Under this license, works must always be attributed to the copyright holder (original author), cannot be used for any commercial purposes, and may not be altered. Any other use would require the permission of the copyright holder. Students may inquire about withdrawing their dissertation and/or thesis from this database. For additional inquiries, please contact the repository administrator via email (scholarship@uwindsor.ca) or by telephone at 519-253-3000ext. 3208.

Online Devanagari Handwritten Character Recognition

by

Shruthi Kubatur

A Thesis
Submitted to the Faculty of Graduate Studies
through Electrical and Computer Engineering
in Partial Fulfillment of the Requirements for
the Degree of Master of Applied Science at the
University of Windsor

Windsor, Ontario, Canada

2012

© 2012 Shruthi Kubatur

Online Devanagari Handwritten Character Recognition

by

Shruthi Kubatur

APPROVED BY:

_____

Dr. Tirupati Bolisetti
Department of Civil and Environmental Engineering

_____

Dr. Rashid Rashidzadeh
Department of Electrical and Computer Engineering

_____

Dr. Maher Sid-Ahmed, Co-Advisor
Department of Electrical and Computer Engineering

_____

Dr. Majid Ahmadi, Co-Advisor
Department of Electrical and Computer Engineering

August 27th, 2012

DECLARATION OF ORIGINALITY

I hereby certify that I am the sole author of this thesis and that no part of this thesis has been published or submitted for publication.

I certify that, to the best of my knowledge, my thesis does not infringe upon anyone's copyright nor violate any proprietary rights and that any ideas, techniques, quotations, or any other material from the work of other people included in my thesis, published or otherwise, are fully acknowledged in accordance with the standard referencing practices. Furthermore, to the extent that I have included copyrighted material that surpasses the bounds of fair dealing within the meaning of the Canada Copyright Act, I certify that I have obtained a written permission from the copyright owner(s) to include such material(s) in my thesis and have included copies of such copyright clearances to my appendix.

I declare that this is a true copy of my thesis, including any final revisions, as approved by my thesis committee and the Graduate Studies office, and that this thesis has not been submitted for a higher degree to any other University or Institution.

ABSTRACT

This thesis proposes a neural network based framework to classify online Devanagari characters into one of 46 characters in the alphabet set. The uniqueness of this work is three-fold: (1) The feature extraction is just the Discrete Cosine Transform of the temporal sequence of the character points (utilizing the nature of online data input). We show that if it is used right, a simple feature set yielded by the DCT can be very reliable for accurate recognition of Devanagari handwriting, (2) The mode of character input is through a computer mouse – training the system with which will lead to jitter-robustness, and (3) We have built the online handwritten database of Devanagari characters from scratch, and there are some unique features in the way we have built up the database. Lastly, after comprehensive testing of the algorithm on 2760 characters, recognition rates of up to 97.2% are achieved.

ACKNOWLEDGEMENTS

I am truly indebted and thankful to my advisors Dr. Maher Sid-Ahmed and Dr. Majid Ahmadi, whose encouragement and guidance at every step enabled me to develop an understanding of the subject. I am grateful for all their patience and support that they showered on me throughout the length of my research. Dr. Maher Sid-Ahmed introduced me to this exciting field of research and I would always remember the lengthy and useful discussions we have had about various tools and techniques and the vastness of knowledge in this area. I have gained immensely from the numerous critical analyses and constructive suggestions made by Dr. Majid Ahmadi, who also taught me the art of presenting my work. I feel truly blessed to have had supervision from these two distinguished researchers and true experts in my field.

I would also like to thank all my friends who were kind enough to provide me with their handwriting samples – a collective effort without which this research would not be possible.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

Even to this day and age, humanity's concentrated efforts continue for the perfect machine/computer that can emulate the immaculate sensory abilities of the humans that have been perfected over centuries of evolutionary trial-and-run mutations. This daunting task includes, but is not restricted to, conceiving a machine that is able to sense and understand its surrounding like how we humans do and produce some kind of a turnout that is useful for us. The most basic of all human communications – writing – happens to hold an essential key in bringing about this effect, and any machine, worth its salt, should at least be able to recognize basic human writing on a script level, if not its implications and nuances. Any undertaking that aims to engender such an intelligent computer that recognizes human handwriting comes under the broad purview of what is known as handwriting recognition.

1.1 Foundation

Handwriting recognition comes in two flavors – offline or online – subject to the availability of scanned/digitized version of handwriting or handwriting trajectory data, respectively. In both cases, the handwriting is analysed, understood, and uniquely mapped to a digital representation of the original handwriting – thereby eliminating ambiguity and subjectivity for all further machine processing. Online handwriting recognition was effectuated by the advent of pen computing devices that consisted of a sensor which could track the position as well as the pen-up/pen-down information of a

stylus (that mimicked a pen) while it was being used to write on a tablet (that mimicked a sheet of paper). The fact that all humans are comfortable and well versed in writing with a pen and a paper makes the employment of pen computing devices along with online handwriting recognition, a far more justifiable human computer interface, in terms of usability and ergonomics alike. This is also proved by the fact that pen based devices were conceptualized as early as 1950s and 1960s – around two decades before the *mouse* and other graphical user interfaces came into existence.

Compared to early 1950s, we now have compact, more powerful and less resource-consuming tablet PCs and pen-based devices. This has resulted in a renewed interest in developing new algorithms that accurately recognize the alphabet set of any given script – thus propelling us towards envisioning a future where high performance online handwriting recognition is a routine inclusion in the standard feature set of the modern day tablet PCs.

1.2 Devanagari, the script

Devanagari is an ancient Indian script that is used to write languages such as Sanskrit, Hindi, Marathi and several others – Hindi being the official language of 1.2 billion people worldwide. Algorithms that are aimed at providing high recognition rates for online Devanagari script recognition will prove beneficial to 17.5% of world's population. Although a lot of work has been reported for online handwriting recognition in English and Asian languages such as Japanese and Chinese, there have been very few attempts at online Devanagari handwriting recognition. Thus, the need for more efficient

online handwriting recognition algorithms and the under-represented status of Devanagari script set the premise for the work done under this thesis.

As part of this partly unprecedented undertaking – the lack of firm precedence thereof being set in the details of the methodologies adopted to cater to various steps involved – several well-known concepts, formulations, and tools are brought together, keeping abreast with the subtle geometrical evolution of the Devanagari script. While the major components of the complete development cycle that defines present day online handwriting recognition are retained to a large extent, the techniques and tools that go into the workings of each of these components are carefully picked to result in the minimum resource consumption while not having to compromise on the final character classification accuracy. The salient point in this framework pivots around the fact that a fairly simple feature set has been effectively put to use in that the essence of the geometric distinction between characters (in a 46-alphabet-strong set) is captured remarkably well, before being fed to the final classification stage – which itself is laid out elegantly along with classifier ranking and fusing techniques, leading to a higher recognition rate than seen ever before.

1.3 Outline of the thesis

The thesis is organised according to a logical structure and flow in such a way as to best present the different aspects of the research conducted. Chapter 2 contains certain preliminaries that a reader needs to get acquainted with in order to better understand the work. Chapter 3 focuses on the history of handwriting recognition and also contains the

literature survey conducted as part of this research. Chapter 4 on design and methodology provides the reader with a clear picture of how existing tools are intelligently used, combined and modified in a sequence of rigorous steps to solve the research problem defined in the scope of this master's thesis. The results of each of the step detailed in chapter 4 are then presented in a logical and pictorial form in chapter 5. The conclusions that could be drawn based on these results and the possibilities and ideas for future research in this area are discussed in chapter 6.

CHAPTER II

PRELIMINARIES

2.1 Devanagari Script

Devanagari is an Indian, syllabic alphabetic type of script that is used to write several languages like Sanskrit, Hindi, Marathi, Bhojpuri, Nepali, Konkani, Sindhi, Marwari, Pali, Maithli and many languages that are spoken in various parts of India. The word "Devanagari" is a combination of two words – "deva" which means *God* and "nagari" which means *urban establishment*. Put together, these words mean "*Script of the Gods*" or "*Script of the urban establishment*" [1].

Some salient features of this script [2]:

1. The script is written from left to right and after the completion of each word; a horizontal line is placed on top.
2. Each letter represents a consonant with an inherent schwa vowel a [ə] which can be killed by a diacritic or matra.
3. Vowels can occur independently or in conjunction with diacritics.
4. In contemporary Devanagari script, there are 13 vowels and 33 consonants. (See Fig. 2.1 and Fig. 2.2)

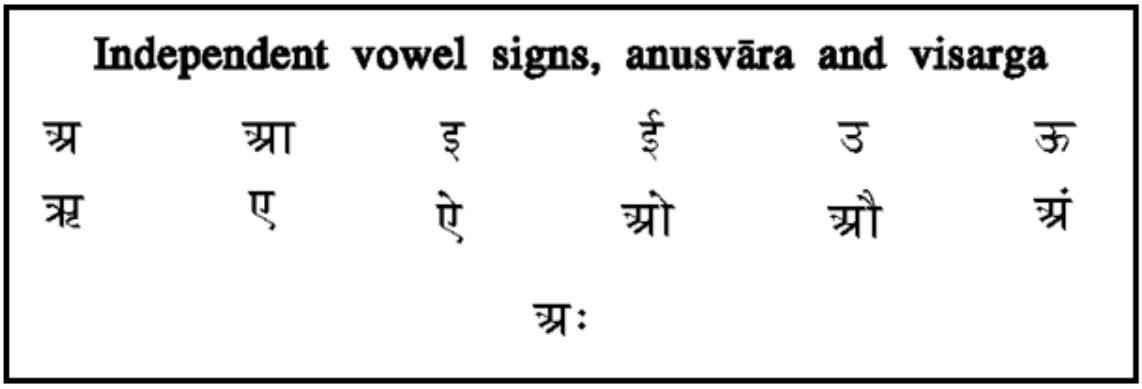

Fig. 2.1: 13 Vowels of the contemporary Devanagari script

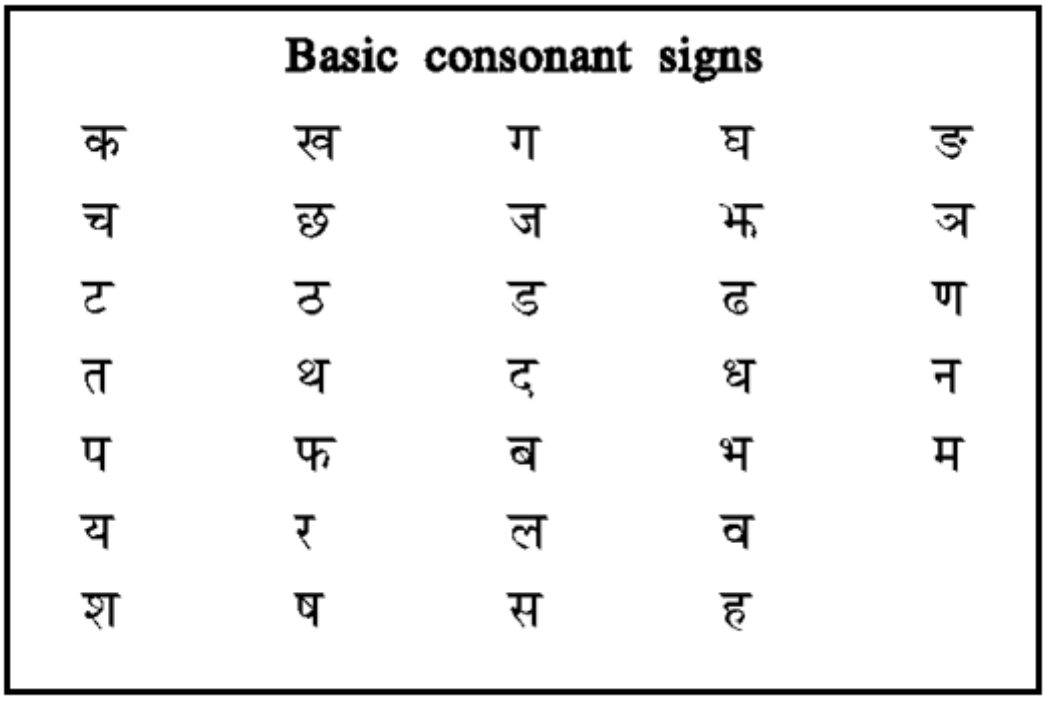

Fig. 2.2: Consonants of the contemporary Devanagari script

2.2 Smoothing and Filtering

Smoothing is the process that involves removing noise from a data set by averaging the data points with their neighbours. Smoothing and filtering are two terms that are used interchangeably, since smoothing brings about a filtering effect by reducing the high frequencies in the data and strengthening the low frequency content of the data set.

Before we go deeper into the processes and techniques employed in modern systems (that process live signals or stored data), it would be appropriate to induce to the lay reader some sense of the nature of the very problem dealt to the processing system – which in this case for a recognition system is the presence of unwanted randomness in data points, which lumped together are termed "noise". Stemming from the more common usage of the word *noise* in the scenario of auditory perception, one might be compelled to draw the analogy (much rightly in this case however) of the hampering caused by the unexplained disturbances to the audibility of the more important pieces of sounds, which could range anywhere from important conversation, a lecture, all the way up to music and just about anything that is generated for a purpose, and not by a flicker of randomness.

Now, the conceptual similarity of noise in the realms of any statistical data is not different at all. It is the same random variation caused *amongst* the data which makes it extremely hard to tell the portion of data that was meant to exist in the record – which could be a 2D capture (image) or a raster of coordinates – from the randomly added data points by a process of variation that can have only a statistical existence and never a

rational one. It is this feature that is thoroughly undesirable, leading to all the data muffling, and has to be removed before any processing can take place – just to ensure that the effort of the processing system works on the right data (that is meant to be processed in the first place) and not on the muffled sections.

A very logical way of dealing with these statistical variations in the data points is to get rid of all conspicuous peaks and dents in the smooth flow of the data, by averaging each data point with its neighbouring points and replacing the data point with this average, where the number of neighbouring points is defined by a term called "span". This definition of the moving average filter ensures that the response of the smoothing is equivalent to a low pass filter and it can be described by the difference equation [3]

$$X_{smoothed}(i) = (2N+1)^{-1} * [x(i+N) + x(i+N-1) + \ldots + x(i) + \ldots.. + x(i-N)] \quad \ldots. \quad (2.1)$$

Where, $X_{smoothed}(i)$ is the smoothed value that replaces the i^{th} data point x (i) and (2N+1) is the length of the span (L_{span}). There are rules that need to be followed while applying the moving average filter.

1. The length of span (2N+1) should be an odd number to make sure there are an equal number of points (N) on either side of the data point to be smoothed.
2. The span is adjusted for the points that cannot accommodate the specified number of points on either side. These are the few initial and end points of the data set and the span for these points is adjusted using the formula,

New $L_{span} = 2 (\min (N,a,b)) + 1$

…..(2.2)

Where, *a* is the number of points on the left side of the data points and *b* is the number of points on the right side of the data points.

3. The end points are not smoothed as the length of span cannot be defined for these points.

2.3 Discrete Cosine Transform (DCT)

The Discrete Cosine Transform (DCT) is a transform that represents any signal as a summation of cosines that oscillate at different frequencies. The algorithm for DCT was developed by N. Ahmed et. al [4] and was posed as solutions to the problems of dimensionality reduction in pattern recognition and Wiener filtering, the two most popular areas that are catered to by a class of orthogonal transforms such as Discrete Fourier Transform (DFT), Haar Transform, Karhunen-Loeve transform (KLT) and the Slant Transform (ST).

In the area of machine learning or pattern recognition, the dimensionality of the data needs to be reduced. The orthogonal transforms provide a method of transforming the data in pattern space to a space with reduced dimensionality which is known as the feature space of the transform. These transforms are usually noninvertible since they retain the important features of the data in pattern space by discarding the redundant data.

The Wiener filter is used to filter out the noise from a signal by comparing the signal to an estimated noise-free version of the same signal. Orthogonal transforms are used to calculate the filter matrix that needs to be convoluted with the noisy signal in order to obtain the filtered output, since the filter matrix calculated using these transforms results in a substantial number of elements having values close to zero. Those elements in the matrix can be replaced with zeros thus reducing the number of multiplications and additions in the convolution process. We will not go further into the application of DCT in the field of Wiener filtering and turn our attention to the definition of DCT and how it fares compared to other popular orthogonal transforms.

Given N real numbers x_1, x_2, ..., x_N, the DCT of these numbers are defined in different ways with slight modifications, producing the transformed DCT co-efficients $X_1, X_2, \ldots\ldots, X_N$

DCT – 1

$$X_k = \frac{1}{2}(x_1 + (-1)^k x_N) + \sum_{n=2}^{N-1} x_n \cos\left[\frac{\pi}{N-1} nk\right], \quad k = 1, \ldots, N \qquad \text{....(2.3)}$$

This definition of DCT of N real numbers makes it equivalent to the DFT of 2N-2 real numbers with even symmetry.

DCT – 2

$$X_k = \sum_{n=1}^{N} x_n \cos\left[\frac{\pi}{N}\left(n + \frac{1}{2}\right)k\right] \qquad k = 1, \ldots, N \qquad \text{....(2.4)}$$

This is the most popular definition of DCT. DCT of N real numbers is equal to half of DFT calculated on a series of 4N real numbers which is obtained by replacing every even indexed element by zero and extending the series to make it even symmetric.

DCT – 3

$$X_k = \frac{1}{2}x_1 + \sum_{n=2}^{N} x_n \cos\left[\frac{\pi}{N} n \left(k + \frac{1}{2}\right)\right] \qquad k = 1, \ldots, N \qquad \text{....(2.5)}$$

This definition of DCT – 3 is the inverse of the DCT defined in DCT – 2.

DCT – 4

$$X_k = \sum_{n=1}^{N} x_n \cos\left[\frac{\pi}{N} \left(n + \frac{1}{2}\right) \left(k + \frac{1}{2}\right)\right] \qquad k = 1, \ldots, N \qquad \text{....(2.6)}$$

This DCT is also known as Modified Discrete Cosine Transform (MDCT)

As can be seen from these definitions, DCT of any series of real numbers can be calculated based on DFT of the same series with slight modifications. There are numerous algorithms that make use of this property of DCT, thus reducing the complexity of DCT computation by employing Fast Fourier Transform (FFT) algorithms such as Cooley-Tukey algorithm, Prime-factor algorithm or Rader's FFT algorithm. The FFT algorithms, themselves, have a complexity of O (N log N) and the algorithms for fast computation of DCT using FFT have an added complexity of O (N) for pre- and post - computations. However, there are other fast efficient algorithms that compute DCT

directly – without the aid of FFT, which employ the classic method of factorization of N for reducing the computational complexity to O (N log N).

Before we wrap up our section on DCT, we need to look at why DCT acts as a good signal compression tool and how good the tool is compared to other tools available out there. A thorough mathematical comparison of DCT with other orthogonal transforms such as KLT, Haar, Fourier and Walsh–Haddard based on variance and rate distortion has been made by N. Ahmed et. Al [4]. If we have a look at the comparison (See Fig. 2.3 and Fig. 2.4), it becomes clear that DCT is closer to KLT in terms of its variance distribution of transform co-efficients and rate distortion criterion. The definition used by the authors is DCT – 2.

From the definition and performance of DCT with respect to other transforms, we can see that DCT not only acts as a good feature extraction method and dimensionality reduction technique, it also performs better than most other transforms and is as close to the optimal transform KLT as could be possible [6]. However, when the complexity of computations is taken into account, DCT fares better than KLT as it is very simple to calculate [5].

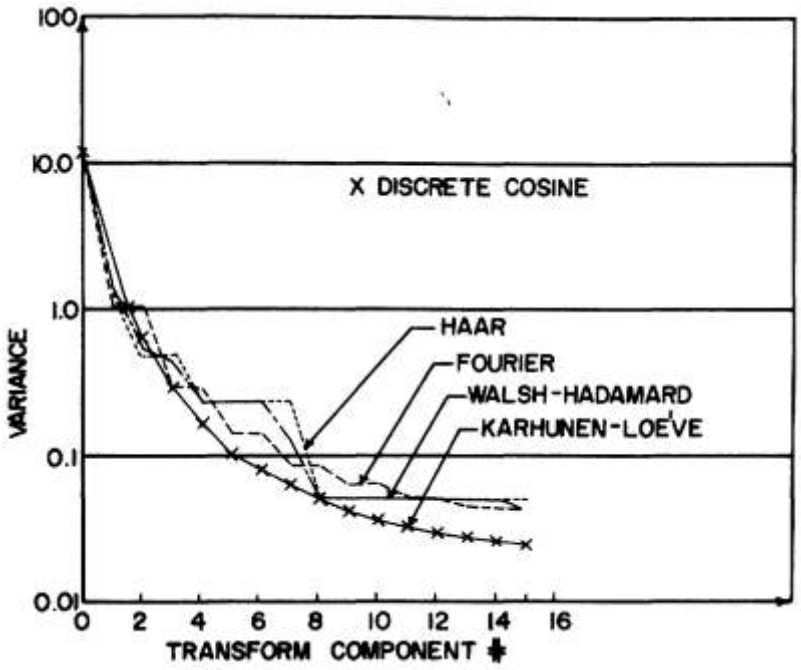

Fig. 2.3: Transform domain variance; M = 16, ρ = 0.95. [4]

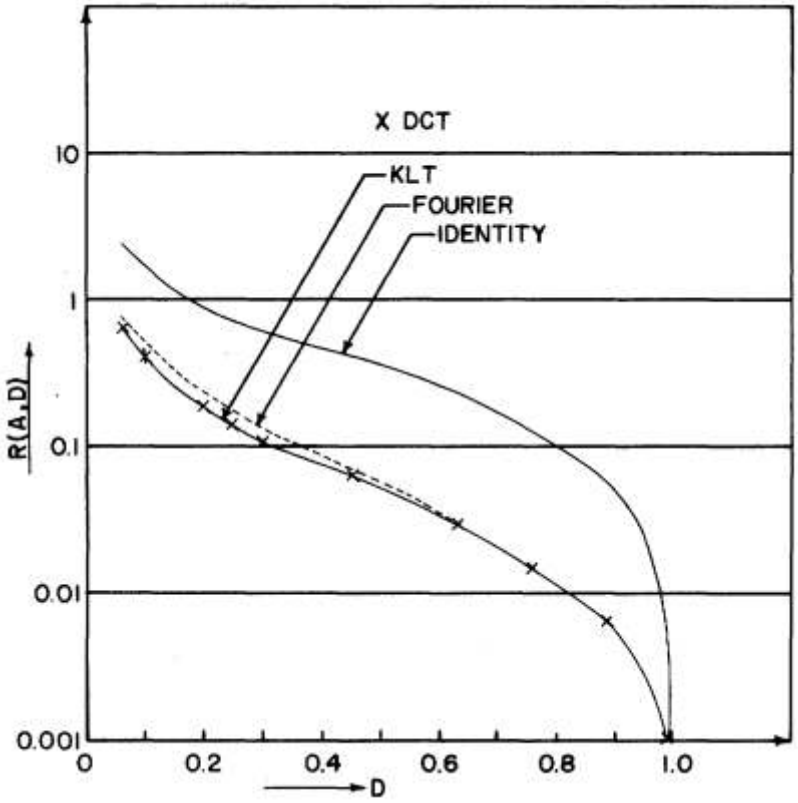

Fig. 2.4: Rate versus distortion for M = 16 and ρ = 0.9 [4]

2.4 Artificial Neural Network (ANN)

An Artificial Neural Network (ANN) is a network of neurons that acts as a statistical tool to relate inputs to outputs and recognise patterns in data .It was initially designed to emulate the various processes of the biological neural system and a neuron model that is an abstraction of the complexity of a real neuron was developed. However, over a period of time it has evolved to be employed in a variety of applications such as cognitive psychology, regression analysis and pattern recognition. In this section, we discuss the role of neurons as building blocks of ANN, the mathematical model of ANN and its application in the field of pattern recognition. As a pattern recognition tool ANN helps in classifying classes that are not linearly separable as will be explained later.

An artificial neuron, which is the building block of ANN, consists of inputs, a summing junction and an activation function; in its most basic and popular form (Fig. 2.5).

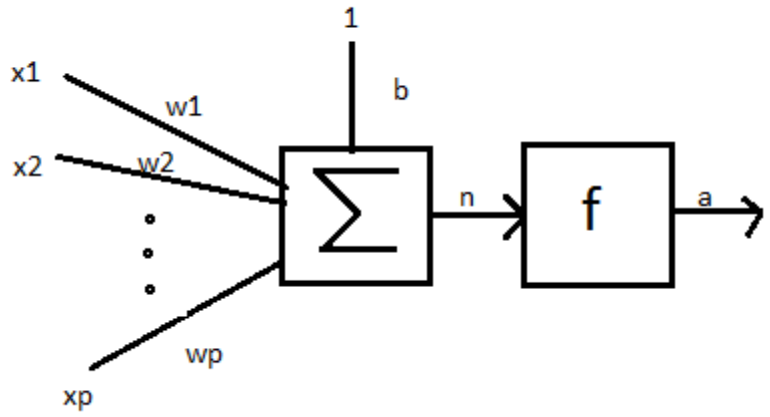

Fig. 2.5: An abstract model of a biological neuron: An artificial neuron

The inputs are x1, x2… xp which get multiplied by their respective synaptic weights w1, w2, …. wp. The summing junction has a bias of value b and its input is always 1. The summing junction sums all its inputs to give the input (n) to the activation function. The activation function is a non-linear function that computes the output (a) of the neuron. The output of the neuron can be expressed mathematically as

$$a = f(n) \quad \text{....(2.7)}$$

$$n = \sum_{i=1}^{p} x_i w_i + b \quad \text{....(2.8)}$$

The transfer function can be seen as a limiting function that dictates the output of the neuron based on the input to the transfer function (n). The most basic transfer function is the threshold function which gives an output of 0 if n is less than a threshold (usually 0) and an output of 1 if n is greater than or equal to the threshold. (Fig.)

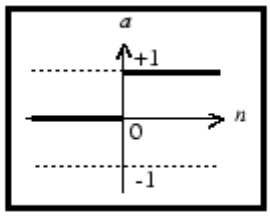

Fig. 2.6: Threshold transfer function (with threshold value = 0)

There are other different types of transfer functions and the most popular ones are linear, log-sigmoid and tan-sigmoid. The activation-output characteristics of these transfer functions are as shown in Fig 2.7.

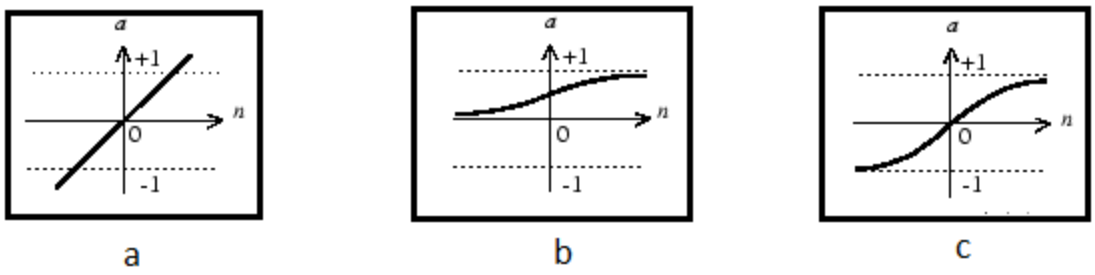

Fig. 2.7: Transfer functions: a). Linear transfer function, b). log-sigmoid function, c). tan-sigmoid function

The weights and the bias are the adjustable parameters of the neuron, and we can obtain a particular output for a specific combination of weights given a specific set of inputs. The neuron can be made to behave in some desirable and interesting way by varying these parameters. However, the limitation of an artificial neuron is its ability to classify only linearly separable classes. The difference between linearly and non-linearly separable classes is best shown in Fig. 2.8.

.

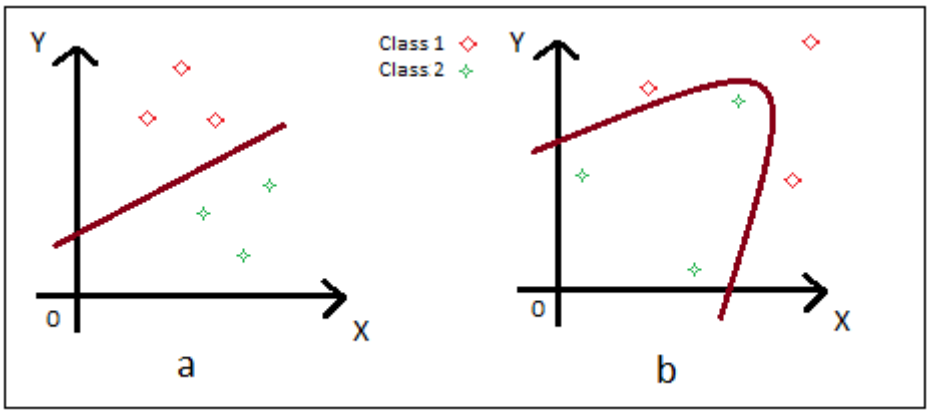

Fig. 2.8: Example of a) Linearly separable classes, b) Non-linearly separable classes

Ever since the inception of artificial neuron by McCulloch and Pitts [33], various simple and complex networks consisting of multiple neurons were developed. A network of neurons (ANN) was necessary for the classification of non-linearly separable classes and the deployment of these ANNs lead to the development of algorithms that could compute the weights and biases of the multiple neurons involved. ANNs were unique since they provided the possibility of learning and in order to realize the learning functionality in ANNs, special learning algorithms were engineered that could not only adjust the weights and biases but also could do so taking into account the relationship between desired outputs in response to a set of inputs.

The ANNs differ in their choice of transfer function, topology and learning algorithms. In this section, we will be focussing on one particular kind of ANN called the feed-forward network since it is the most common network and other models are based on it. This is in contrast with the other type of neural network topology which is the recurrent neural network with feedback loop. The learning algorithm discussed is the resilient backpropagation algorithm. An ANN may have one or more of layers with layer containing multiple neurons. Most neural networks have two layers of neurons with the first layer called hidden layer and the second layer called output layer. These two layers are preceded by the input layer which consists of just inputs and no neurons (Fig. 2.9). The network receives inputs from the input layer and the output of the neural network is available at the output layer.

These feed forward neural networks are also known as multilayer perceptrons (MLP). The backpropagation algorithm that was popularised by Paul Werbos and D. E.

Rumelhart [34] accomplishes the design of a feed forward neural network. Before we dig deep into the workings of the back propagation algorithm, it becomes imperative at this point that we discuss a little bit more about the concept of learning. The learning process is necessary for the network to acquire desired knowledge and apply this process on its inputs and provide us with the correct outputs. As mentioned earlier, this necessary knowledge is stored in the set of parameters (weights and biases) of all the neurons in the network. On a broad level, the learning process has been classified into two types:

- *Supervised learning* that involves training the network with a set of training samples
- *Unsupervised learning* that involves classifying models based on unlabelled data.

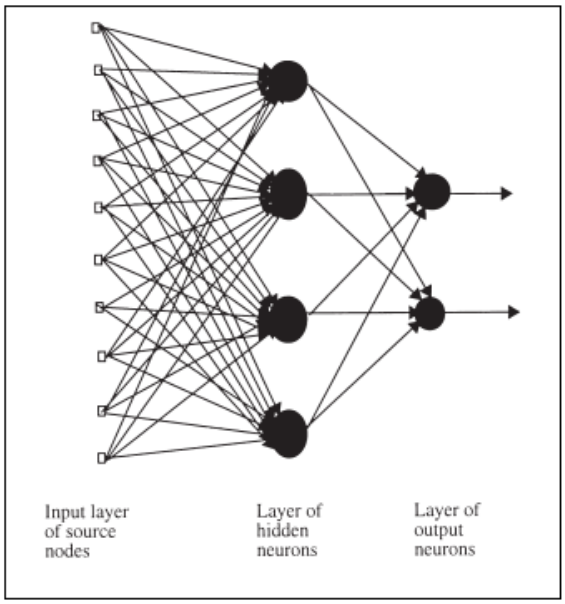

Fig. 2.9: A feed forward neural network with one hidden layer and one output layer

Let the set of training samples in supervised learning be denoted by P, which is a set of N pairs of samples.

$$T = \{(\mathbf{x}_1, \mathbf{y}_1), (\mathbf{x}_2, \mathbf{y}_2) \ldots (\mathbf{x}_N, \mathbf{y}_N)\} \quad \ldots.(2.9)$$

Where,

$\mathbf{x}_1, \mathbf{x}_2 \ldots, \mathbf{x}_N$ are all input vectors and belong to input vector space **X**

$\mathbf{y}_1, \mathbf{y}_2 \ldots \mathbf{y}_N$ are all desired output vectors for the respective input vectors and belong to output vector space **Y**

N is the number of such samples

As part of supervised learning, the neural network is trained with these sample pairs either on a one-by-one basis or in a batch and after each input-output pair pass ($\mathbf{x}_i$, $\mathbf{y}_i$), the parameters of neurons in hidden and output layer are computed so that the actual output ($\mathbf{a}_i$, where i = 1,…, N)of the neural network is close to the desired output ($\mathbf{y}_i$, where i = 1,…, N) based on a performance measure such as the mean square error defined as

$$E = \frac{1}{N} \sum_{1=1}^{N} (y_i - a_i)^2 \quad \ldots(2.10)$$

The mean square error is the designated cost index that needs to be minimized over the complete set of training samples and the resulting neural network represents the function f : **X** -> **Y**, which could be described as the optimised classifier function. For minimizing the mean square error (E), weight of each neuron in a neural network needs

to be moved through all possible values in proportion to the gradient of the error (E). The *gradient descent* algorithm that finds out the local minimum of any function is used for this very purpose. The *delta rule* employs the gradient descent algorithm for obtaining the weights of neurons in a single layer perceptron. If, however, the network involved is a multilayer perceptron then the generalised version of the delta rule – backpropagation algorithm is employed. Thus, we can note that by being an algorithm that is used by multilayer perceptron training, backpropagation algorithm needs to have the capability to handle adjustments in a large number of neurons in very complicated network topologies.

Backpropagation algorithm has 2 broad steps –

1. Forward phase: In this step, parameters (weights) of the neurons in the network are fixed and inputs are applied at the input layer and the corresponding outputs are calculated. This step also includes calculation of the error

   $$\mathbf{e}_i = \mathbf{y}_i - \mathbf{a}_i \quad \ldots.(2.11)$$

   where, $\mathbf{y}_i$ is the desired output vector and $\mathbf{a}_i$ is the actual observed output vector for a given input vector $\mathbf{x}_i$.

2. Backward phase: The error calculated in the forward phase is propagated in the backward direction through all the neurons in both output and hidden layers. It is in this stage that the weights of the neurons are adjusted by using a generalised version of the delta rule, so as to minimize the error $\mathbf{e}_i$.

In sequential mode, the forward phase is implemented on each input-output pair individually and in batch mode, the forward phase is carried out on a sizeable batch of input-output pairs. In both cases, the error function E is determined using Eq. (2.10) and is treated as the cost function to be minimized.

Let us consider the j^{th} neuron in the hidden layer of Fig. . The input to this neuron is the actual input vector to the neural network $\mathbf{x}_i$ (i = 1, …, N) and let the weight matrix of the synapses connecting the inputs to this neuron be $\mathbf{w}_{ij}$. The error function E needs to be minimized with respect to each of the weights in the weight matrix and in order to accomplish that, we need to consider the partial derivative of the error function with respect to each weight in the weight matrix. A simpler figure (Fig. 2.10) is provided for the convenience of the reader.

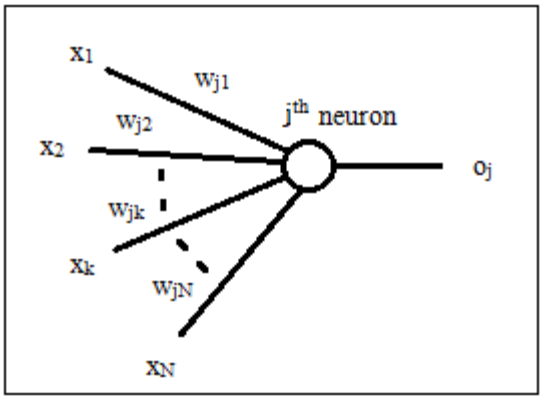

Fig. 2.10: A magnified view of the j^{th} neuron

Let us consider the partial derivative of E with respect to the weight that connects the k^{th} element in the input vector to the j^{th} neuron (w_{jk}). Applying the chain rule, we obtain the gradient

$$\frac{\partial E}{\partial w_{jk}} = \frac{\partial E}{\partial z_j} \cdot \frac{\partial z_j}{\partial w_{jk}} \quad \text{....(2.12)}$$

$$\frac{\partial z_j}{\partial w_{ji}} = x_k \quad \text{....(2.13)}$$

Where, z_j is the weighted sum of inputs for the j^{th} neuron and x_k is the k^{th} element of the input vector. Employing the general rules of differentiation, it can be shown that

$$\frac{\partial E}{\partial z_j} = -(t_j - o_j)(1 - o_j)o_j \quad \text{....(2.14)}$$

Where, t_j = desired output for j^{th} neuron

o_j = actual output obtained for j^{th} neuron

Substituting Eq. (2.13) and Eq. (2.14) in Eq. (2.12), we obtain

$$\frac{\partial E}{\partial w_{jk}} = -(t_j - o_j)(1 - o_j)o_j \,.\, x_k \quad \text{....(2.15)}$$

Deriving the delta adjustment to weight w_{jk} from Eq. (2.15), we obtain

$$\Delta w_{jk} = -\eta \frac{\partial E}{\partial w_{jk}} = \eta\ (t_j - o_j)(1 - o_j)o_j \,.\, x_k \quad \text{....(2.16)}$$

The term η is called the learning rate of the backpropagation algorithm. Thus, the weight w_{jk} needs to be updated according to the equation

$$w_{jk} + \Delta w_{jk} \Longrightarrow \text{updated } w_{jk} \qquad \text{....(2.17)}$$

As we can see, the weights are updated in the opposite direction to the sign of the gradient, making sure the error decreases in steps of η with every weight update. The choice of learning rate η determines how fast the local minimum is reached and also affects the quality of learning. The forward and backward steps are repeated for all training samples or till the network has met satisfactory performance standards.

Many algorithms have been proposed for adapting weights in neural networks that are based on backpropagation algorithm. The resilient backpropagation (Rprop) is one of the best methods that could be used for achieving batch learning in MLPs. The Rprop considers only the sign of the partial derivative of the error function and not on its exact value, thus making it an appropriate, fast and easy algorithm for implementing in the case of noisy error function.

In Rprop algorithm, the weight adjustment equation is given by

$$\Delta w_{jk} = - \text{ sign } \left(\frac{\partial E}{\partial w_{jk}} \right) \Delta_{jk} \qquad \text{....(2.18)}$$

Thus the direction of the weight update is dependent only on sign of the partial derivative and not on its absolute value. Also, the weight is updated in steps of Δ_{jk}, which is based on changes in the sign of the partial derivative. Depending on whether there is change in the sign of the partial derivative, Δ_{jk} is multiplied by a value of η^+ ($\eta^+ > 1$) or a value of η^- ($0 < \eta^- < 1$). Thus, Rprop ensures that the local minimum is reached in a more elegant way with as few numbers of iterations as possible.

In each iteration (t) and for each weight w_{jk}

{

if $\text{sign}\left(\frac{\partial E}{\partial w_{jk}}(t-1)\right) = \text{sign}\left(\frac{\partial E}{\partial w_{jk}}(t)\right)$

{

Updated $\Delta_{jk} = \Delta_{jk} \cdot \eta^{+}$

Updated $w_{jk} = w_{jk} - \text{sign}\left(\frac{\partial E}{\partial w_{jk}}(t)\right) \cdot \text{Updated } \Delta_{jk}$

}

else if $\text{sign}\left(\frac{\partial E}{\partial w_{jk}}(t-1)\right) \neq \text{sign}\left(\frac{\partial E}{\partial w_{jk}}(t)\right)$

{

Updated $\Delta_{jk} = \Delta_{jk} \cdot \eta^{-}$

Updated $w_{jk} = w_{jk} - \text{sign}\left(\frac{\partial E}{\partial w_{jk}}(t)\right) \cdot \text{Updated } \Delta_{jk}$

}

}

Fig. 2.11: An easy-to-follow algorithm of Rprop method is provided for enhanced readability.

This chapter on preliminaries touched upon all necessary concepts required to understand the premise of the thesis. This section was aimed at familiarizing the novice reader with relevant mathematical background to appreciate the forthcoming sections.

CHAPTER III

REVIEW OF LITERATURE

3.1 Basics

Handwriting recognition (HWR) is the process of obtaining handwritten material, either in the form of scanned images of handwritten documents or in the form of spatial co-ordinates of the hand movement of users, and developing it into a form that can be recognized by a computer or any text based application. Early developments in the area of handwriting recognition were mainly aimed at aiding the blind and also improving interface between human and the computer, thus bringing about increased functionality and user-friendliness in computers.

3.2 Offline Vs Online Handwriting Recognition

As discussed, handwriting recognition can be either offline or online depending on whether the input is scanned and digitized copy of handwritten documents or x-y co-ordinates of the hand movement - for example through a motion based sensory screen of a computer, respectively. Offline handwriting recognition is a specific case of "Optical Character Recognition" (OCR). There are positive and negative points associated with both methods of recognition. OCR is well suited in situations where the handwritten material is already available in document form or when the range of characters is already known.

Online handwriting recognition, on the other hand, proves to be a better choice when the handwriting needs to be recognized as the character is being written. A deeply descriptive, objective and comprehensive comparison of offline and online handwriting recognition methods on Latin alphabet has been done by Rejean Plamondon and Sargur N. Srihari [7]. In their landmark survey, they have clearly elucidated the various differences, pros and cons of online and offline HWR. Besides having lesser data storage requirements than offline recognition, online recognition provides better recognition rates as well. The memory requirement for online recognition of an average English word is about few 100 bytes, at a sampling rate of 100 samples per word, whereas for offline recognition, it is about few 100 kilo-bytes, sampled at a rate of about 300 dots per inch. A recognition rate of 78 percent has been reported for offline word recognition with 1000 word lexicon [8], in comparison to a recognition rate of 80 percent for online word recognition with 21000 word lexicon [9]. Though better recognition rates have been reported in both offline and online HWR, online recognition algorithms outperform offline handwriting recognition methods in terms of data memory requirements and recognition rates, given the same lexicon size.

3.3 Pen Computing and Tablet PCs

An important precursor for online HWR is the availability of tablet PC or digitizer, which is collectively known as pen computing. A tablet PC or digitizer serves as the interface to collect the time ordered 2-Dimensional co-ordinates of the user's hand movement and employs a pen (stylus), and a touch sensitive screen, that collects and also stores the required co-ordinates. The extensive history of development of pen computing

technology spans over a century and a detailed discussion of it by Andre Meyer [10] and by Jayashree Subrahmonia and Thomas Zimmerman [11] present a complete and detailed picture of the then state-of-the-art. In 1888, Elisha Gray was awarded a patent for inventing a device that could capture handwriting and thus began the exploration that spewed many devices that could accurately acquire and store handwriting. All major corporations have produced their own versions of tablet PC and most commercially available ones make use of top choice handwriting recognition algorithms. A pen computing device basically consists of a tracking technology (magnetic, electric, ultrasonic or optical tracking) that determines the position of the stylus on the screen. Most modern tablet PCs use one or a combination of these tracking technologies.

This thesis tries to highlight the salient patches of the survey of pen computing in [12], such that the entire spectrum of pre-1990 research is shone light upon – without entering into the specifics of surveyed ideas that are obsolete or are close to being so. The authors point to the inception of pen-trajectory tracking in the 1950s, immediate interest, and a dip in attention in the 1970s, and revival a decade later. It is to be noted that this fluctuation in interest and attention can be attributed to the technological differences, by which I mean the 1960s and '70s definitely lacked the power of computers, tablet PCs, touch-screen hand-held devices, and efficient algorithms for quick processing. Online handwriting recognition has survived the test of time for the simple reason that a whole lot more data is available for recognition systems than an offline snapshot of handwriting can ever provide. It is needed to expand upon the ideas of pen computing and trajectory digitization introduced above.

Dimond's “Stylator” [13] has been the earliest documented tablet digitizer, the most popular one being the RAND table [14]. These spurred the initial thrust towards online handwriting recognition research. Before I move on to the actual research surveyed in [12], it is imperative to introduce the two popular technologies used to build digitizers. This will serve as a background with which to better understand the conditions prevalent for the researchers of the past. As of 1990, electromagnetic/electrostatic and pressure sensitive were the two dominant technologies.

The former category of tablets (electromagnetic/electrostatic) had regularly spaced x and y grids of conductors. The stylus tips had a loop of wire. When the grid (or the loop) was excited with an electromagnetic pulse, the loop (or the grid) would detect the induced voltage or current as a sine wave. The tablet conductors were searched to point to the pair closest to the loop, and the precise position between these two conductors was determined by interpolation. As one can expect, the recognition gets more accurate with higher density of the regularly spaced on-tablet conductors.

Pressure-sensitive tablets were a little different, in that the tablet makes use of two types of layers – one conductive, and one resistive. The spacing between any conductive layer and the corresponding resistive layer is the key for operation, and it is where the concept of vertical pressure comes in. In a pre-specified direction (usually along either the x- or the y-axis), some electric potential is applied across one of the resistive layers. By applying pressure on the stylus, the user would enable the conductive layer to make contact with the resistive layer, picking the voltage from the latter, thus picking the coordinate of the point in which pressure had been applied. As we can see, the only thing

required of the stylus is the pressure. This eliminated the need to find a special type of stylus for the tablet, and it has been a huge advantage since.

The authors also discuss briefly other sensing techniques, like the acoustic and optical sensing methods – and these will be skipped from the survey in this thesis. However, it will be mentioned that the authors wrap up the section on digitizers by alluding to the dawn of transparent tablets (the earliest of which dates back to 1968 [15]) – enabling the user to use the same tablet for input as well as output, at the same time.

With the different tablet digitizer technologies behind us, it is time to move on to have a look at the pre-1990 perception of handwriting properties and mechanisms of recognition based on these properties. It is at this point that the fact that we are dealing with online handwriting becomes more important and useful. Handwriting when recorded as written (which is of course the essence of online capture) can be viewed as a series of strokes, tagged with their time-stamps – in order to properly stack the coordinates of each stroke in a timely manner so as to preserve handwriting causality intact.

3.4 General Online Handwriting Recognition

The work done under online HWR is vastly focused on Latin alphabet and numeral set. Interest in online HWR commenced in early 1950s, largely due to the advent and advancement in pen computing. Important contributions to the field of online HWR were made in 1960s and 1970s which were a result of availability of better tablets PCs, feature extraction and classification methods. The different handwriting recognition

algorithms differed in the way the researchers chose to solve various issues that act as obstacles to the achievement of perfection in handwriting recognition.

We can easily note that all online HWR algorithms work on the basic assumption that different handwritten characters in any given script have a high degree of decorrelation. While this is a useful property, another characteristic of handwriting that is at least as useful if not more is that the handwriting samples of the same given character in any particular script are highly correlated. A deep understanding of the variability in handwriting is essential if we intend to build better and efficient HWR systems and readers are referred to [16] for a thorough discussion of the variability effects in English handwritten script. As part of the literature survey on online HWR, I would like to include significant work in all steps of HWR – starting from pre-processing all the way up to final classification. To that end, let us begin an overview of important past work pertaining to the pre-processing steps.

Though this thesis work does not require segmentation between characters as well as segmentation within each character, it is still required to showcase all crucial scholarly work on segmentation so that the continuity, discussion flow, and completeness are all kept intact. Earliest efforts in segmentation were in the form of obtaining the horizontal distance between the characters [17] or by providing a certain amount of time for the user to complete writing the character. Most of the times, users were asked to write characters in certain boxes meant for each character which is akin to character recognition. Many of the recent published works talk of two dimensional separation of characters [18] or use of both spatial and temporal information for character segmentation [19].

Smoothing and filtering of the handwriting samples are vital pre-processing steps that have been implemented in various ways. The most commonly used method for smoothing is averaging of the sample points with its neighbours [20]. As part of filtering, most researchers resample the data points so that consecutive points are all equally spaced [20]. To bring about various types of normalizations, different algorithms have been suggested. Some of the examples are: deskewing algorithms that are carried out to de-slant the characters, size normalization algorithms in order to reduce or expand characters from different users which tend to be of different sizes to the same size as the case may be, stroke length normalization equalises the number of sample points in each stroke so that all strokes have uniform length [21].

After obtaining the handwriting samples and applying all necessary pre-processing algorithms in order to obtain smoothed, filtered and normalized data points, the samples need to be reduced to a set of non-redundant, yet sufficiently informative data samples. All methodologies that are aimed at achieving dimensionality reduction are basically different forms of feature extraction methods. Anyone, who sets out to actualize handwriting recognition, has a varied collection of feature extraction approaches to choose, ranging from static features and dynamic features, a combination of both static and dynamic to those features that are representation of the data samples in some other domain. The choice that one makes is usually influenced by factors such as the language and script in question, the choice of classification method and the ease with which certain features can be extracted from the samples. Binary features are mainly used as part of a decision tree so that each branch of the tree corresponds to a particular combination of 0's and 1's, which in turn represents a character. Non-binary features are used in

conjunction with some classification method such as clustering analysis that clusters those features into a set of groups.

With each character being represented by extremely efficient feature arrays, we are faced with the challenge of developing a classification method that sorts these feature vectors into one character class among all character classes in the script. In simplest of terms, classification is a sub-problem in pattern recognition, in which, given an input vector the process involves matching this input vector with an output label. These classifiers are statistical in nature, which means these classifiers are able to detect certain characteristics, attributes or properties of classification based on a large number of observations with known output label for each input vector. This type of classification is also known as supervised learning. Linear classifiers such as Fischer Linear Discriminant, perceptron and others such as Artificial Neural Networks (ANN) [22], Support Vector Machines (SVM) [23] and k-nearest neighbour are some of the classifiers that have been used extensively by various researchers.

3.5 Devanagari Online Handwriting

After a detailed perusal of the state-of-the-art techniques in online HWR in English language, let's turn our attention to the core of this thesis topic – Devanagari script online HWR. Though a huge amount of work has been done on offline handwriting recognition for Devanagari characters, publications focused on online Devanagari handwritten character recognition are few in number.

Scott D. Connell et. al. [24] were the first few ones to work on this area and their seminal work on "Recognition of unconstrained online Devanagari characters" in 2000 brings to light the various issues involved in online Devanagari HWR. They have implemented a method which uses 5 classifiers with both online and offline features. Offline features refer to those features extracted after considering the character as whole rather than calculating the features based on the sample points that are collected from the handwriting. Their resultant accuracy rate of the five classifiers combined is reported to be 86.5 %, with the individual classifiers themselves providing much lesser accuracy rates.

In chronological order, the next work that could be mentioned as part of a survey of the developments in online Devanagari HWR is by A. M. Namboodiri et. al. [25]. The authors propose a method to classify a given online script into one out of 6 scripts, one of which is Devanagari. The method used 11 spatial and temporal features extracted from the strokes of the words and attained an accuracy of 87.1 percent on a database of 13,379 words. Although, this piece of work on classifying words into scripts has no implications and results that could be used as a reference for the work on online Devanagari HWR, it throws ample light on the various features that could be used for classifying Devanagari characters.

Niranjan Joshi et. al [26] specify a system that uses structural features such as mean (x, y) values, positional cues and directional codes at the stroke level and a test vector is developed based on these features. The system uses subspace method for classification in which each character class is represented by a basis vector which is a set

of N eigenvectors. The test vector is assigned to that class whose basis vector has the smallest orthogonal distance from the test vector. The most notable result reported is an average accuracy of 93.05% on a set of 100 frequently occurring characters.

A stroke-based recognition system using Hidden Markov Model (HMM) has been proposed by Abhimanyu Kumar and Samit Bhattacharya [27]. Their work on recognition of isolated Devanagari characters has been carried out to be implemented on the iPhone. The authors have created 42 stroke classes, and one HMM is constructed for each stroke class. The first round of stroke classification results are fed into a second round of classifiers, along with look-up tables. Six scalar features indicate the size and shape of each sub-stroke. However, the authors sign off with an end-of-paper discussion, and no actual results are presented to the reader. This makes the assessment of their approach very ambiguous and speculative in terms of accuracy and recognition rate, as the number of samples and degree of variety keep mounting.

The recognition rate, time required to train the system and number of samples used per character for testing are some of the criteria that can be utilized to compare the performances of these above mentioned scholarly work. The recognition rate which is the ratio of number of correct classifications to the total number of training samples, gives a measure of how well the system is performing in general. The next performance criterion is the training time which is the total time required to train the network to correctly classify all the training samples. The number of samples per character class used for training the system is also an essential performance criterion since it provides an insight into how well the system performs for varied handwriting samples. Apart from these

performance measures, there are parameters that take into account details such as run time and memory requirements. Any scholarly work survey on HWR is not complete without a comparison of the published algorithms based on these criteria. However, we can notice that the authors that figure in the technical survey show no form of conformance to the afore discussed performance criteria, but rather stick to exposition of the bare minimum information that is open to further inference and investigation – such as the overall recognition rate and the number of samples per character used for testing their algorithms.

This concludes our fairly extensive survey on the state-of-the-art, and the work that has led us up to it. We are now at a stage where we can fully appreciate the problem I have been working on, the complications involved, how it differs in approach and solution to the scholarly work we have just finished surveying, and finally how I solve the problem I started out with, along with the recognition rate achieved.

CHAPTER IV

DESIGN AND METHODOLOGY

4.1 Overview of the Proposed Methodology

We have seen quite a few times that handwriting recognition is a capability linked, in this context, to any programmed module that resides in a processing unit. This is the core of the system to which the recognition capability is attributed. However, much like the role the CPU has come to play in the whole computing experience, the program module is part of a bigger system. It is, more often than not; wrong to focus on the core unit sidelining the peripheral units. In fact, it is not uncommon at all for the performance and design of the peripheral units of a typical recognition system to bear a rather direct impact on the overall ability to recognize handwriting correctly and efficiently.

It is with this background that we need to look at the overall development of our system, going sequential – as if we are tailgating the data to be processed (or more specifically in this case, the handwriting sample to be recognized). This way, it is easy for the reader – and more importantly for the designer – to expect and be prepared for the alteration that awaits the data next.

Once the nature of the system is decided upon, one of the first things to be acted upon is the nature of the data. In this case, the system that is being built is a recognition system for *online* handwriting characters belonging to the ancient Indo-European heritage script, Devanagari. This immediately dictates the input mechanism, and the policies to be employed thereof. While mechanism entails the procedure and process of taking in the handwriting samples, policies refer to certain discretionary measures to be adopted to aid

in the process of data acquisition – in order to bring about numerous features to the system being built. These features can range from ethnic invariance, robustness, speed invariance, angular invariance, and immunity to distortion, all leading the system toward one ultimately desirable state of user-independence. In more explicit terms, what this means is that the data has to be *tuned* in a way that the system would have no difficulty in recognizing characters from a secondary data set (usually termed the "test set") which has no intersection with the primary data set (commonly referred to as the "training set") – regardless of the variation that finds expression as a function of several attributes of the database contributors – leaving the legibility of the written script as the only concern for successful recognition.

The *process* of thus *preparing* the data in a way such that the system functioning is rendered insensitive to user-dependent variances, environment-dependent fluctuations, and randomness (whose capture happens only in posteriori measurements, leading to well-developed statistical quantifications) is an art – that goes by the name *pre-processing*.

The various stages of readying a potential handwriting recognition system goes far beyond the core module, often starting at data acquisition (either afresh or via an accredited database), routing through storage and pre-processing, right to the identification of feature cruxes and their extraction, all the way up to the design of the classification unit at the core – which, akin to a two year-old kid, learns to tell one class of data from another, based on the featured identified in the previous step.

This cycle of developing a new recognition system is summarized below:

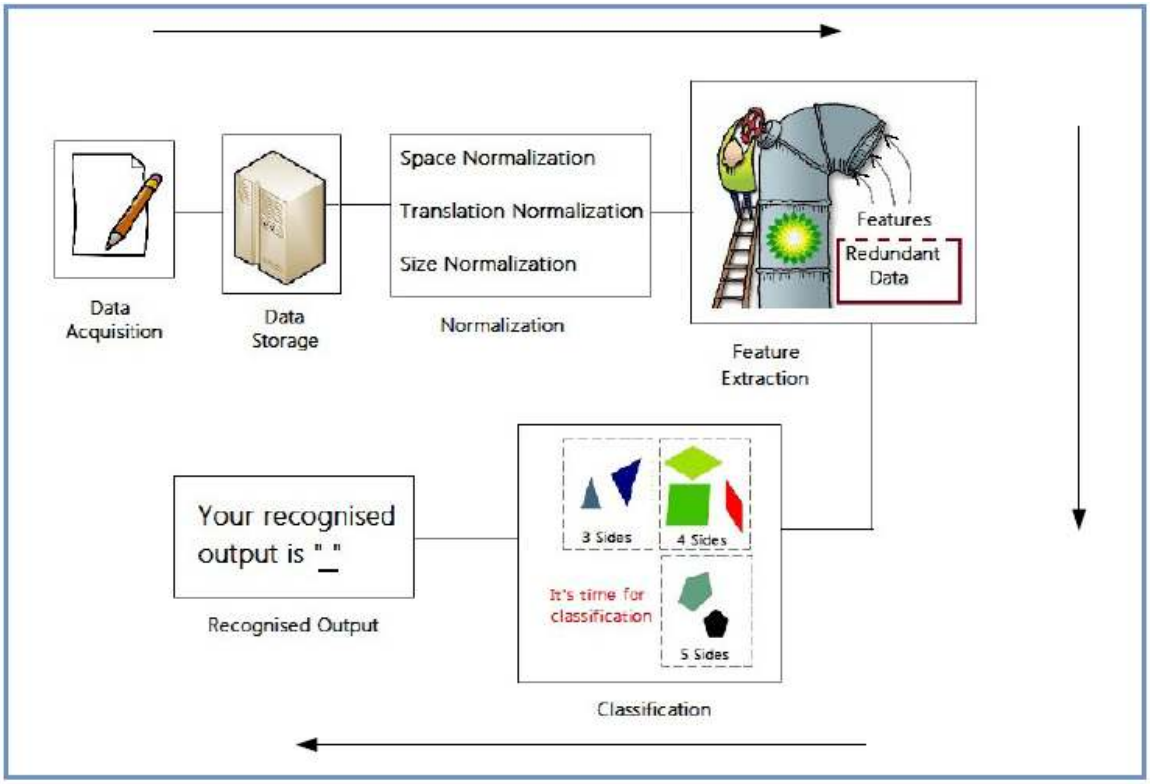

Fig. 4.1: The developmental stages of the handwriting recognition system – adopted in the work.

The various stages in the HWR systems are

1. Data Acquisition
2. Data storage
3. Normalization
4. Feature extraction
5. Classification and Recognition

4.2 Nature of Data, Acquisition and Storage

The above steps are common to any recognition system. Specifically, in our case, steps 1 and 2 are merged into one block. It is only natural to do this since the data acquired has to be taken care of, either in terms of processing or storage. Immediate processing of the acquired data forces the system to turn into a real-time entity, which is not what this work of mine is intended for. In handwriting recognition systems, there are steps that typically need to be followed before the actual processing can take place. The data accumulation can be overwhelming for any real-time system (with minimum memory) while the system readies one batch of data for actual processing. It is due to this simple reason that handwriting recognition (and many other pattern recognition and image analysis) systems are seldom made real-time.

Once we come to the decision that the system is not real-time, then we need to take care of the data accumulation, such that the integrity and content of the data – in its entirety – is not lost. This calls for a proper storage arrangement. Therefore, it is vivid as to why any handwriting recognition system designer would naturally want to merge the steps of data acquisition with the imminent and inevitable step of data storage.

Before discussing data acquisition at any length, it is a good idea to understand the type of recognition system – as the nature of the system has a direct impact on the nature of the data being fed. Most of the work, as we have seen during the literature survey chapter, focuses on the recognition of script which is printed or handwritten *some time ago*, on a *non-electronic* medium. Now, what that means is that the there is no information available about the *way* in which the data was written. This type of data is

usually termed “offline” data. The most widely found examples of offline data are images/pictures of written/typed script. Often in the world, offline data is what we have easier access to – be it pages from a book, an old handwritten document, an unsigned letter, or an ancient manuscript. We don’t have control over the type of data. But in the modern age, the electronic presence is growing, allowing one to think about having more control over the data being acquired.

Offline data samples have their share of disadvantages. Perhaps the best way to drive home the major disadvantage of offline data is through the following example: Consider two persons – Ronald and Donald. Ronald is an influential man, with the real authority to sign an important document of a reasonably high impact. Due to discretionary reasons, Ronald wouldn’t sign a particular document. In Ronald’s absence, Donald, who is a fraudster, carefully signs the document (which Ronald had declined to sign). After Donald finishes signing the document, it looks exactly identical to Ronald’s signature. Now suppose this signature of Donald’s signature is captured offline, then it is nearly impossible to bring Donald to justice. However, had Donald’s forgery been captured electronically, one could have access to the *temporal* features of Donald’s signature. The pressure of the strokes, the timing, and the speed of sub-strokes are all collected in an electronic acquisition of data. This type of data is termed *online*.

The above example illustrates the advantage of dealing with online data as opposed to offline data, due to the availability of temporal characteristics in the former type of data sets.

The example of Ronald and Donald has been made up for the sake of illustration. However, the problem described herein is real. It has been a recurrent problem through the course of history, as we shall see with one real example below.

The forgery shown in Fig. 4.2 below is not the best forgery one could imagine, but it comes pretty close to the original. The concern grows in cases where the visual distinction between two writings (signatures and regular writing alike) is harder to make than the example below.

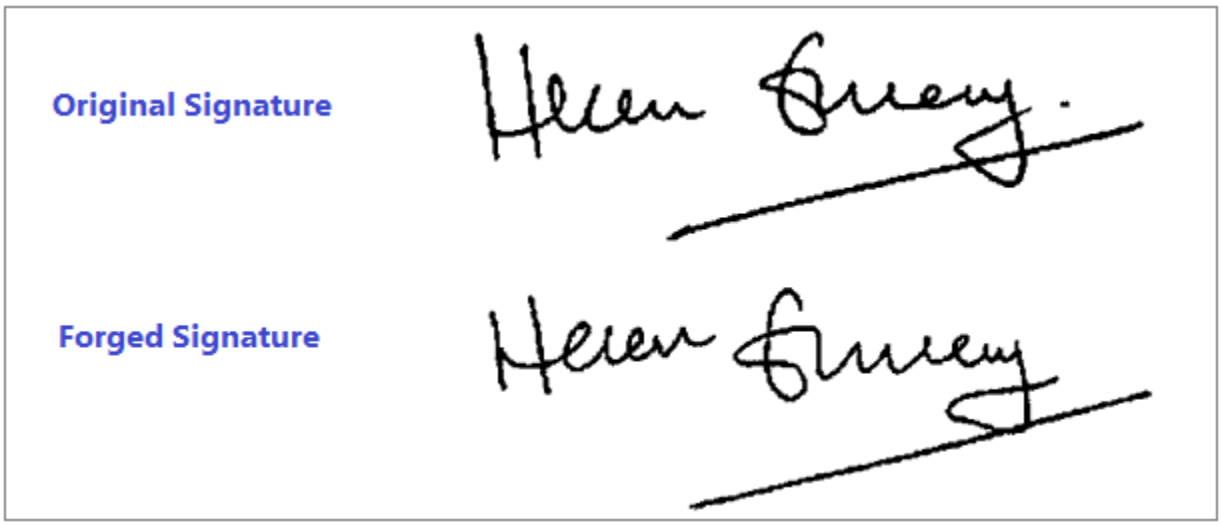

Fig. 4.2: A real signature, and its forged version: offline recording.

Tom Davis (of the University of Birmingham) has an interesting analysis of this forgery [28]. Tom opines, “The forgery is quite good, both in line quality and letter formation, and at first sight looks very convincing. But look at it closely. Firstly, the laudable concentration on line quality has led to a gross error, in that an extra minim has been inserted. Then close examinations reveals that the line quality also lets it down. The capital /E/ of the surname is slightly less assured, with a tendency for curves to turn into

straight lines and corners, which is very characteristic of forgery. And, in the case of the lower loop of the final /y/, the reverse has occurred: a curve in place of an angle and straight line. Then look at the end of that stroke: in the genuine signature the pressure of the pen reduces gradually, while in [the forged version] the line ends abruptly, no doubt with relief; a failure of concentration at the end of the job."

The case has been made, intuitively at the least, for the choice of using online data for training and testing the HWR system. The logical continuation of the choice (of the nature of data) is the specification of the acquisition module of the system.

4.2.1 Need for building a database

It may seem to the reader that the obvious choice for acquiring data is to use a database of handwriting samples – along with the knowledge of the relevant methodology behind the build-up of the database being considered. What adds an extra level of challenge to this work is that there is no readily available database for Devanagari characters, unlike the Yale B/Extended Yale B/CMU-PIE databases that exist for human faces, or unlike the USPS/MNIST digit databases. The direct, unavoidable, implication is that it becomes a responsibility to populate a decent sized database for isolated Devanagari characters, from scratch.

4.2.2 Building the database: considerations and methodology

Recognizing the importance of this step, we worked out a comprehensive way of creating this database, in a way that a variety of writing styles – in the right number – are incorporated in a *layered* fashion. We will revisit the idea of layering in a while.

A quick detour is essential to set the premise for the strategic development of our unique comprehensive database. The protagonist, resorting to personification for lack of a better inanimate qualifier, of this thesis is the script, Devanagari. Devanagari, as explained earlier, is a rich script of the Indo-European belt, now finding residence mainly in the Indian languages. The language-script relationship evolves over time, causing the language to get control over how the script changes to suit the needs of the development of the language itself. In other words, what sets off as a common script for say, five languages, is sculptured by time and the respective language into five similar scripts of common ancestry.

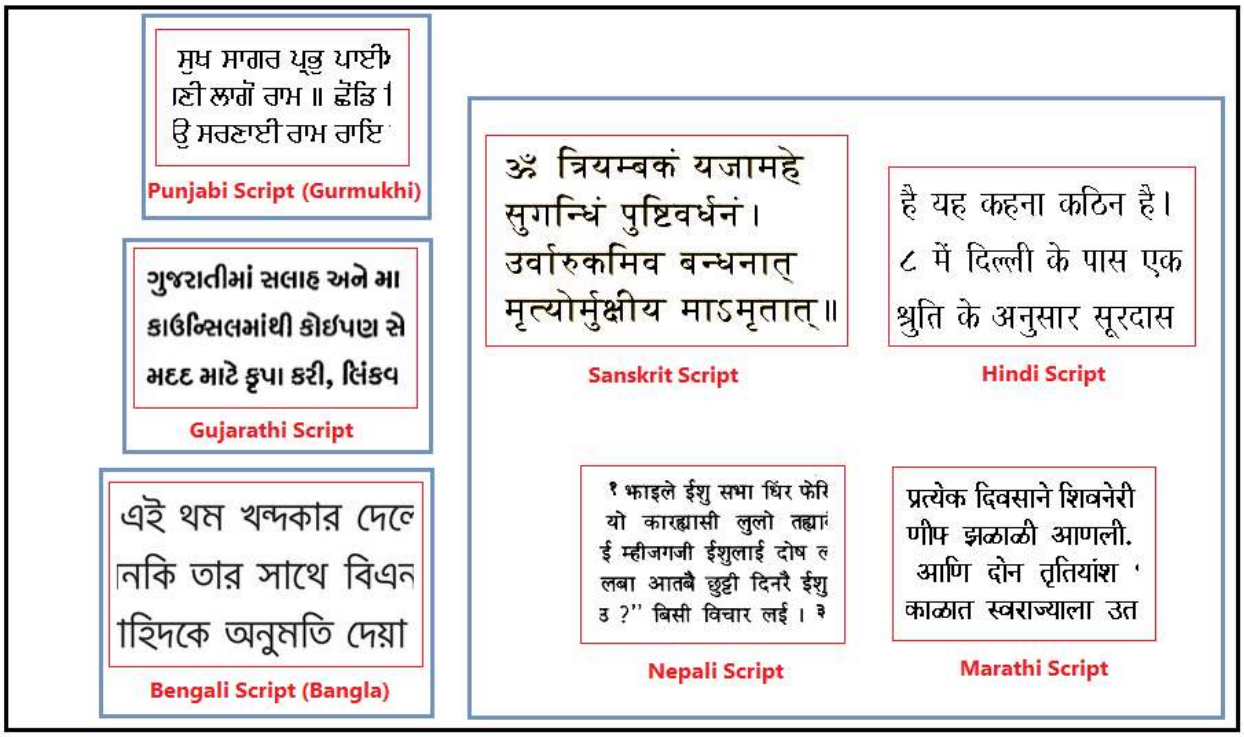

Fig. 4.3: Relative evolution of the common Devanagari script – based on individual languages.

The original Devanagari script is known to most closely resemble the four scripts in the right big box in Fig. 3.3. Within this big box, the four scripts are enormously similar, with just very minor variations. However, the other boxes carry scripts (Punjabi, Gujarathi, and Bangla) that have drifted farther off from the parent Devanagari script. The unique thing incorporated in the database is that we have ensured that the influence of all these script-drifts is taken into account. The people from whom handwriting samples are obtained for building the database have varied backgrounds. Their linguistic origins include Hindi, Gujarathi, Marathi, Punjabi, and Bangla. The scripts of these languages, as we saw above, are off-shoots of Devanagari. We have also taken Devanagari samples from people whose native script has no connection with Devanagari at all (like Kannada, Tamil, Telugu, and Malayalam).

The variety that has been created in the database makes sure that the handwriting recognition system is immune to the influence of the native script (of the handwriting sample contributor).

Let us quickly touch upon the concept of layering, which gains relevance in this context. How entries are arranged in the database is that the samples are spread from contributors of various linguistic origins, in recurring layers, so to say.

Let us pictorially see what layering means.

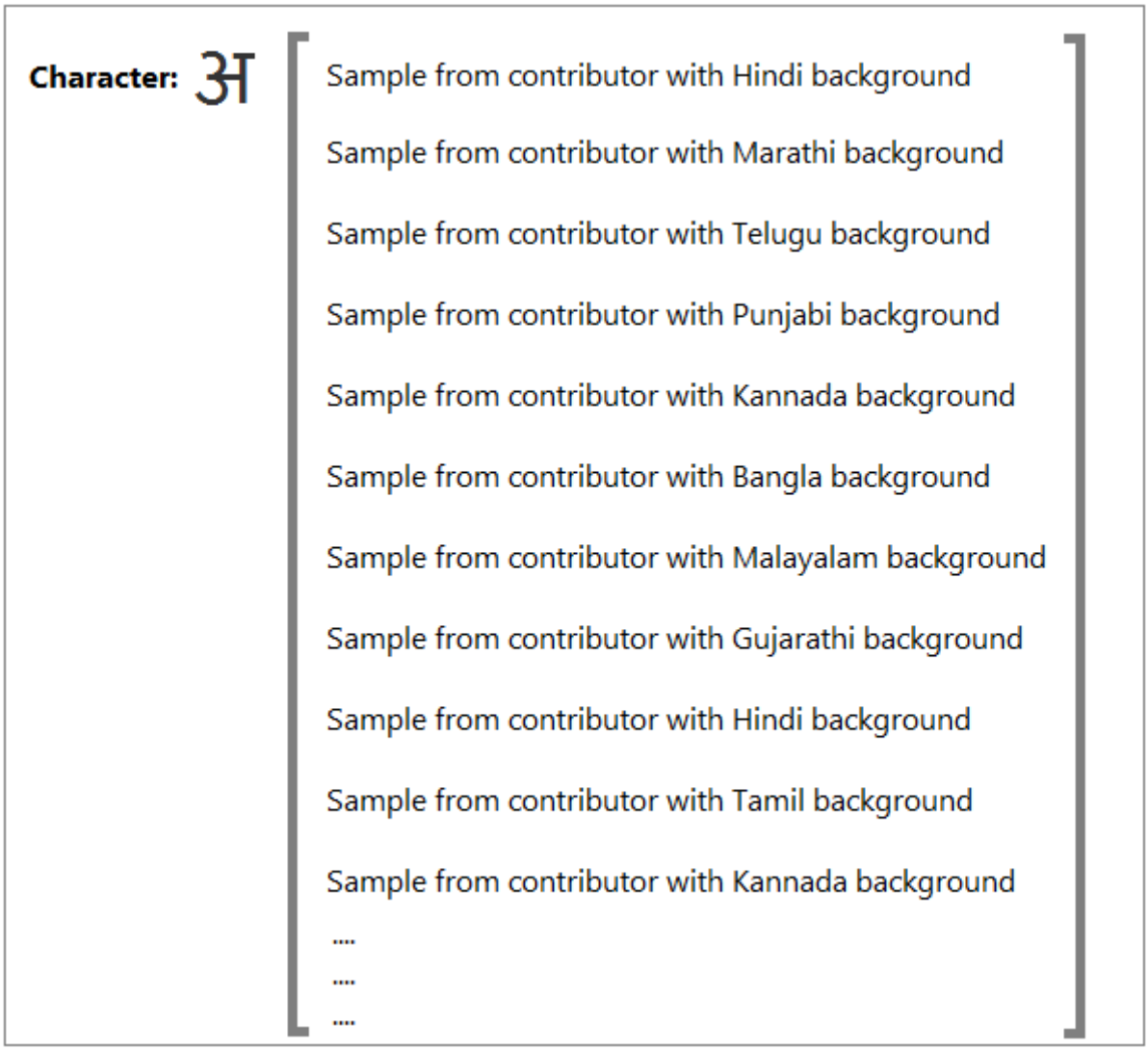

Fig. 4.4: Layering of handwriting samples from contributors of various linguistic backgrounds, making the database uniform and randomly interspersed.

The rationale behind adopting such a scheme as layering is to make sure that even a small section of the database is not biased towards any particular script-influence on the writing style of the Devanagari characters. This way, the results of testing is uniform and largely independent of the section of the database upon which the testing is carried out. This layering is a novel contribution of mine to make the system robust.

A subtle point to be made here is that we have internally divided the database into two big sections – one for training the system, and the other for testing. The contributors for each section are different, meaning that the contributors from whom we have obtained the handwriting samples to populate the training section have not contributed their samples to the testing section, and vice-versa. However, within the training and testing section, the samples are layered (as described above). Therefore, we have managed to make the system writer-independent too.

4.2.3 Acquiring the handwriting samples

Next up, we need to look at how we have actually acquired the handwriting samples from these varied contributors. The choice that has been rather popular in recent times is the use of tablet PCs (more so with the advent of Apple iPad, HP tablets, BlackBerry PlayBook, and the like) or digitizers.

At this point, there is something about the technology that has an influence on the robustness of the system under construction. Let us start with an analogy. Let us suppose the job at hand is to teach a bunch of college students how to crack an entrance examination. The better way to train the students is to get them to solve a variety of problems – tricky, straightforward, trap, and tough – in class, in order to have them prepared for any kind of situation they might have to face on the entrance examination. The training cold be made easier by using just good-to-be-with or easy-to-solve problems, but it will do little good on the examination itself. Going by the same logic, we chose to present this handwriting recognition system with not-so-nice handwriting for the

training routine. It would have been way easier, as a designer and trainer, to have handwriting contributors come into the office and write neatly on a sophisticated tablet PC. It was the easier and simpler option for everyone – but just for the duration of the training. However, we have been far-sighted to recognize the problem with the easier approach in the long run: lack of rigorous training for the real-world testing.

The solution for this problem has been both far-sighted and economically cooperative. I wrote a program (in C++) to capture the movement of the mouse. The program successfully tracks the abscissa (x-coordinates) and the ordinate (y-coordinates) of the mouse position on the screen, along with the time stamps. This special and temporal recording of the mouse movement has to be sampled, without which the data acquired per recording session may be enormous, beyond handling capacity. To take care of that, I introduced a sampling factor into the program – by way of polling the status of the left-click button on the mouse, as a pre-requisite for recording the x, y, and time coordinates. By controlling the frequency of polling, one can control the amount of sample points per stroke of the character.

Recording the handwriting through mouse movements is a jittery procedure, introducing a lot of imperfections into the writing (and uneasiness into the writer). It is this jitter that goes into the training process, making the handwriting recognition system less prone to wrong classifications later on.

The C++ program takes in all the data from the contributor, and exports the data in the form of a simple text file, and saves it automatically in the resident folder of MATLAB. MATLAB in turn opens this file, reads in each triple-valued coordinate entry (x, y, t) for the purposes of computing the Discrete Cosine Transform (specifically DCT – Type II, the choice of which will be elaborated upon in the section on feature extraction). The DCT of the data samples would be a compressed set of 20 values (excluding the DC component), and it is this compressed DCT set which is eventually stored. The original coordinate data set (which occupies a much larger space) is of lesser significance – now that we have derived the DCT from it – and is consequently not stored, thereby setting up a novel circumstance to cut down on memory requirements of the system as a whole.

```
 1  986,198,0
 2  986,198,0.0065
 3  986,198,0.013
 4  986,198,0.0195
 5  986,198,0.026
 6  986,198,0.0325
 7  986,198,0.039
 8  986,198,0.0455
 9  986,198,0.052
10  986,198,0.0585
11  986,198,0.065
12  986,198,0.0715
13  986,198,0.078
14  986,198,0.0845
15  986,198,0.091
16  986,198,0.0975
17  986,198,0.104
18  986,198,0.1105
19  986,198,0.117
20  986,198,0.1235
21  986,198,0.13
22  986,198,0.1365
```

Fig. 4.5: Initial storage of the (x, y, t) coordinates of the mouse movement data for a particular handwriting sample – temporarily saved as a text document in the MATLAB folder.

The text file on the left is opened from within MATLAB, which is why the numbering of the coordinates is seen in the left margin. This image portrays the very essence of online data – the ordering of the coordinates. Therefore, unlike in offline images, online data presented here tells the reader the exact chronological order in which the coordinates were recorded. With this data, one can re-trace the trajectory of the mouse almost exactly (subject to a fairly decent polling frequency applied within the C++ program).

The first two values in each row are the x and y coordinates respectively, expressed in terms of the on-screen pixel values. The last item in each row is the time-stamp, expressed in seconds.

4.3 Pre-processing

4.3.1 Establishing the need

While every effort is being made to make the recognition system as flexible and robust (to variations) as possible, machine capability is still a far cry away from human sensory versatility and efficiency. To put it in perspective would be to compare the machine (which is our recognition system) to a two year old kid – and this too could be demeaning to the kid. What the computer is good at is tireless computation, and any display of real intelligence is a lofty, if not unfair, expectation of the computer from our side. A specific, exaggerated example is not hard to describe. Let us assume that the computer is trained to recognize the character “A” of font size 12 (Times New Roman). The main question framed in the example is: Will the system correctly recognize “A” for

the following three configurations: (1) Font size: 14 (Algerian's equivalent of Times New Roman's size 12), Font face: Algerian (or basically any font face except Times New Roman – which it is trained for), (2) Font size: 20 (again, any font size not close to 12), Font face: Times New Roman, and (3) Font size: 30, Font face: Algerian? See Fig. 3.6.

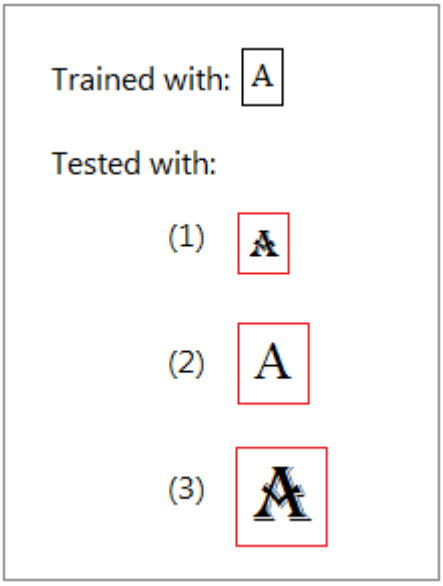

Fig. 4.6: Character A in various configurations. Top to bottom: Times New Roman, 12; Algerian, 14; Times New Roman, 20; Algerian, 30.

The figure above illustrates this thought experiment.

To even a child, recognition of all these characters as "A" is not of any challenge at all. But can we say the same of the computer system? It borders on several AI and cognition topics, digression into which we shall avoid here. For the purposes of the thesis, it suffices to state that the learning mechanisms of a computer are quite rigid, and thus it becomes the responsibility of the designer to ensure that the data upon which the

computer trains, and upon which the testing happens, are to be of some uniform nature. The expectation is not identity (which diminishes the role of the system to pattern matching at best). All that I am striving to accomplish is to have some regulation on the data, before using it for training or testing.

These regulations are essential for the successful performance of the system, but the sample contributors cannot be subjected to regulations while they write the character samples for the database. The last thing we want is impose constraints on the writing. Since the sample contributors enjoy unconstrained writing, the onus of regularizing the samples shifts to the design of the system itself. The steps that take care of this very problem are collectively termed “data normalization”.

Before we get into the specifics of the normalization procedures adopted in this work, a mention should be made about another equally crucial step that has to take place before normalization can be enforced.

4.3.2 Smoothing and De-noising

In the sub-section where we discussed the way the data is entered and stored, we have observed that there is jitter, to say the least, in the data. Now we need to look at what has to be done to cut this jitter and noise out, restoring data integrity in the process.

First, let us look at *smoothing* out the jitter, by making use of any one among an array of filters just waiting to smooth our data. The entire setup of the data acquisition as explained here in the thesis is not known to introduce much noise (the sort that creeps

into the system without notice or welcome – thermal, shot, and Gaussian are examples of which). This does not necessitate the use of the toughest or the most convoluted or the best filter out there. Any filter that does not consume too much of the available resources, and does not take forever, and does a considerably good job of smoothing and de-noising the data, will perfectly fit the bill.

With that view, we have used the Moving Average filter, whose endorsement is chosen from [29]: "The moving average is the most common filter in DSP, mainly because it is the easiest digital filter to understand and use. In spite of its simplicity, the moving average filter is optimal for a common task: reducing random noise while retaining a sharp step response. This makes it the premier filter for time domain encoded signals. However, the moving average is the worst filter for frequency domain encoded signals, with little ability to separate one band of frequencies from another. Relatives of the moving average filter include the Gaussian, Blackman, and multiple-pass moving average. These have slightly better performance in the frequency domain, at the expense of increased computation time."

Since the Moving Average filter is an optimal filter (neither the most convoluted nor characterized by exemplary performance), there needs to be one more *just-in-case* step, to take care of any noise that may be left behind in the data. The back-up de-noising mechanism that we have put in place is a simple block of MATLAB code. The purpose of this block of code is to make L_2-spacial distance measurements between consecutive data points, within each stroke. If we consider one single stroke (comprising the data points between pen-down and pen-up, or specifically in this case, between mouse-click-

hold and click-release), it does not make sense if there is an outlier – especially beyond a certain pre-specified threshold distance. The points beyond the threshold distance (from its neighbour) are removed from the data coordinate set by this block of code. It is very reasonable to treat the outlier beyond the threshold to be noise. The value of the threshold controls how accurately we remove these noise points from our data. If the threshold is set to a very low value (like 2-3 pixel distance), we run the risk of removing several to many points which are genuine data points (which could have resulted from fast writing, among other reasons). Hence, a low value of the threshold eats away many genuine data points along with noise. At the other extreme, a high value of the threshold (like 15-20 pixel distance) may leave most of the outlier (noise) points in the data. We do not desire either consequence. Based on some trial-and-error runs, we could see that the perfect threshold for the system would be an L_2-space separation of 5-7 pixels. Specifically, for the code block, we have used a 7-pixel threshold. The net effect of the moving average filter, coupled with the outlier removal code is very impressive smoothing and de-noising.

4.3.3 Normalization

The data is now smooth and jitter free. De-noising is complete. But it still is not ready to be recognized. The reason for the data still not being ready is the lack of any particular *format*. To understand the meaning of lack of format, see Fig. 4.7.

The job at hand is to bring all the data samples to one format. To achieve this simple-yet-essential goal, three types of normalization routines were used, whose explanations will follow.

a. Translation Normalization

To understand the need for this type of normalization, first we need to illustrate the manner in which the database contributors wrote on the screen (with the mouse). They wrote ten samples (of the same character) at one go, on various positions on a portion of the screen. Though the C++ program was programmed to read the mouse coordinates just based on the state of the mouse buttons (clicked or un-clicked), for the purposes of visual feedback for the writer, they were made to write their samples on the portion of the screen having the MS Paintbrush window open.

As seen in Fig. 4.7, the writers were given full freedom, and were not constrained to write into one particular box. The pixel values of each of the ten samples are dependent on *where* the writer writes the samples. To take care of this variable positional offset, I have applied translation normalization.

Let the minimum value among all x-coordinates (for each sample) be denoted by X_{min}, and the minimum y-coordinate by Y_{min}. Translation normalization is achieved by simply subtracting X_{min} from each x-coordinate, and Y_{min} from each y-coordinate. The effect of this operation is that the whole sample is pushed (without skewing/stretching/compressing) leftward such that the

leftmost pixel of the sample touches the x=0 line, and the sample is then pushed downward (again, without deforming) such that the bottommost pixel touches the y=0 line.

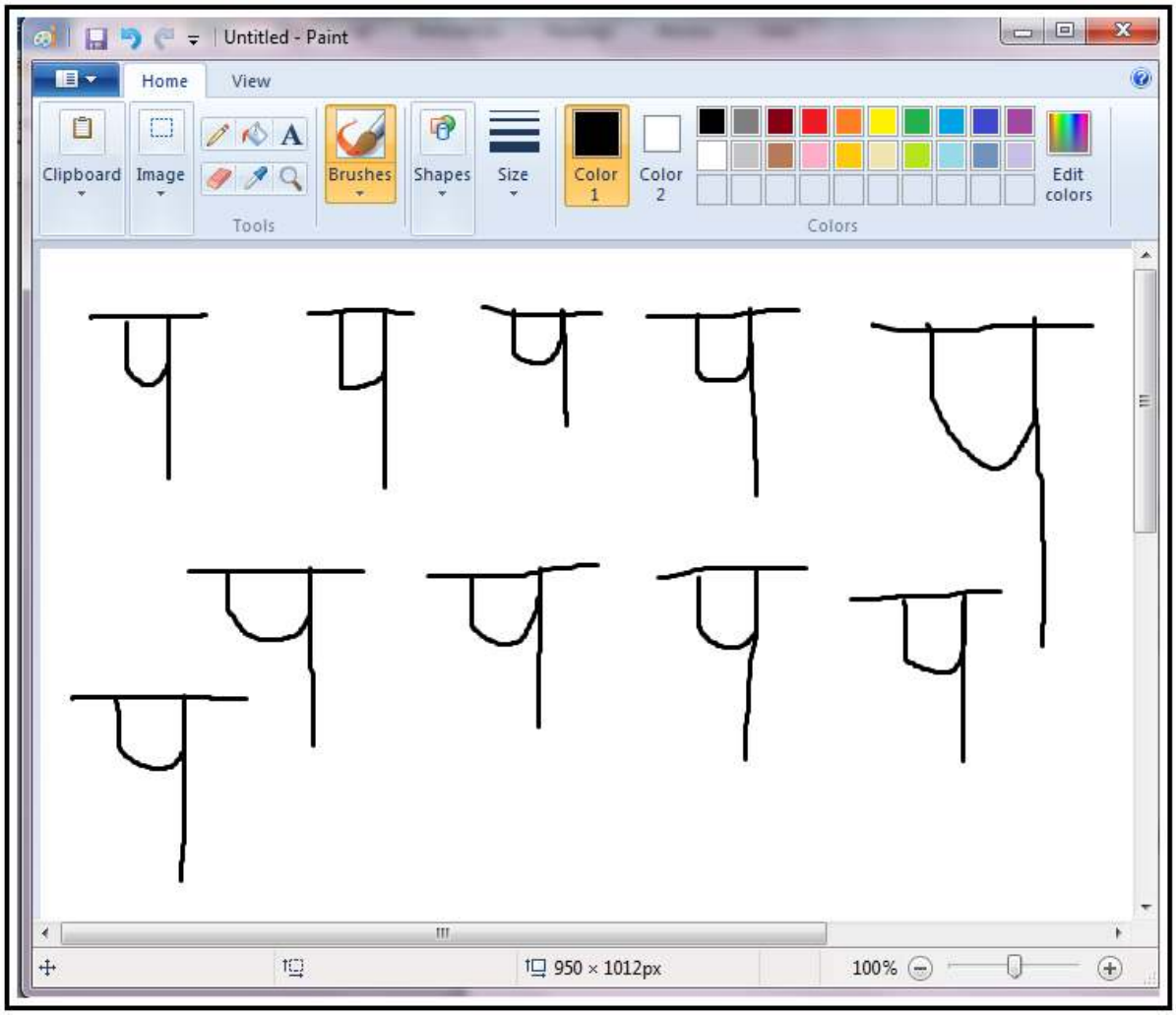

Fig. 4.7: A cropped screen-shot of a typical handwriting sample recording (with feedback provided on MS Paint).

The whole concept behind this type of normalization is that the system gets to have an origin-vertexed box within which to contain the samples. At the end of

this step, though we would have achieved positional uniformity, the size of the dotted box (tightly wrapping the character sample) is a variable. Let us look at how we bring uniformity to that parameter – under size normalization.

b. Size Normalization

Like other forms of freedom the writers have had, there was absolutely no constraint on how big they could write the character samples. This discussion too borders on the rhetorical questions introduced in the explanatory passage of Fig. 4.6.

Without making assumptions on the capability of the system's robustness to variation in size, it is far easier and safer to re-size the samples to some standard measure. It is to be noted that relationship (or proportion) between the vertical and horizontal components of the character samples, is one of the fundamental characteristics of the character. It may even be treated as part of the character feature, known to bear an influence over the quantification of the feature-set (more about which will be discussed later, in the section on feature extraction). This is the main reason why it is a bad idea to skew/stretch the samples in an attempt to re-size them.

The direct implication of this non-skew/stretch condition is that the dotted box (see Fig. 4.8) cannot be of pre-set dimensions. The exception to this rule comes into play only when the formation of the dotted box does not involve tight

wrapping of the sample. But, in our considerations, the dotted box is always the tightest possible wrap.

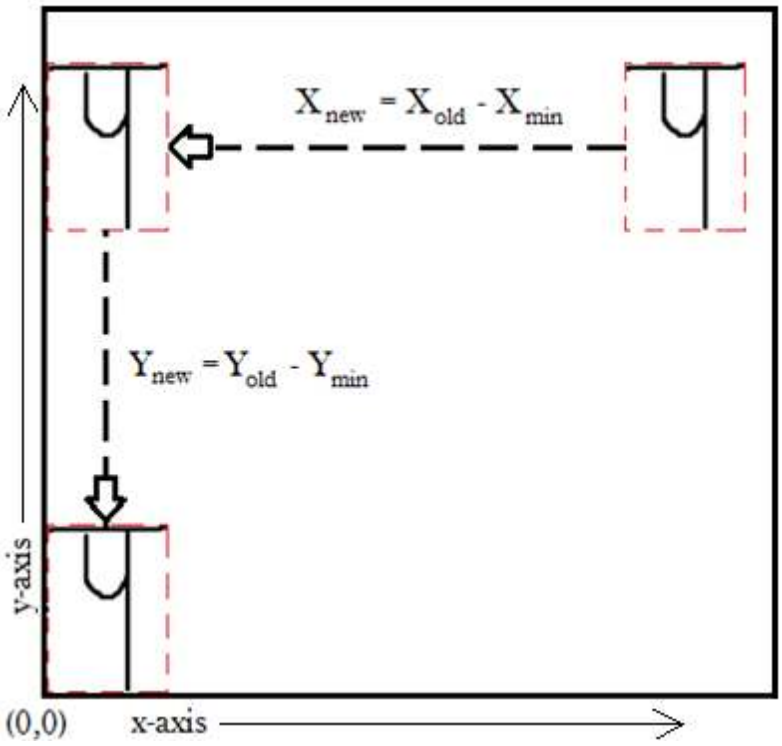

Fig. 4.8: Representation of the general idea of translation normalization.

A well accepted solution to this problem is to put a cap on either the height or the width of the dotted box, but not both. The procedure followed to bring down (or in very few cases where the written sample is too small, to bring up) the size of the samples is represented by the following algorithm:

Step 0: Get the maximum y-coordinate (Y_{max}).

Step 1: The new value of Y_{max} is forced to go to 50. Call this Y'_{max}.

(Discussion: In the Devanagari script, all members of the 46-strong in-use alphabet have nearly the same height, while the width of each character varies from a proper fraction of the height to almost twice the height. It is this observation that places sense in forcing the height of the samples to a constant – which, in our case, is 50 pixels.)

Step 2: Define vertical scaling factor, $S = 50/Y_{max}$

Step 3: $Y_{new} = Y_{old}*S$

$X_{new} = X_{old}*S$

(Discussion: The vertical scaling factor, S, although derived primarily to scale the sample height-wise, is not limited to the y-axis scaling. The same factor, S, is employed to scale the sample width-wise as well. The reason for this is that the x-coordinates are scaled by exactly the same factor as the y-coordinates are scaled with, lest there be any skewing/stretching effect on the samples.)

The net effect of implementing this algorithm is that the sample is now brought down to a tightly wrapped box of height 50, with a variable width.

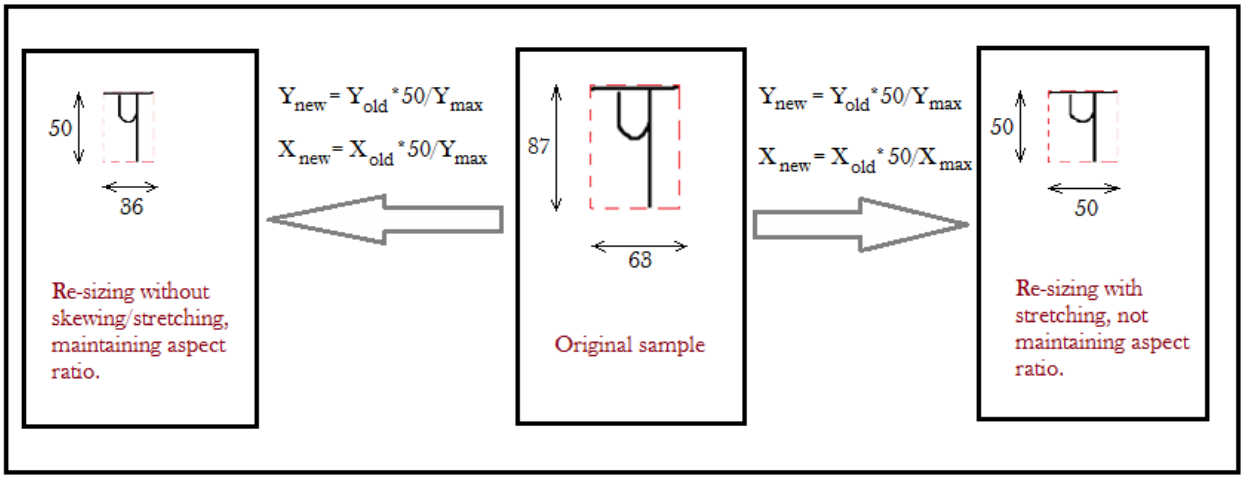

Fig. 4.9: Two sets of re-sizing transformations. The one to the left of the original sample is the one applied in this work.

c. Time Normalization

The last step, before the data samples are fully ready to go on to feature extraction and finally to classification, is time normalization. This is achieved in more than one way. We have used re-sampling, due to its effectiveness and low computational effort.

The main need for having to resort to re-sampling is that different database contributors write at different speeds. The way the C++ program works is that the number of data points acquired per character sample is inversely proportional to the writing speed. Since the program polls to check the state of the mouse click, and then the coordinates, a slower writer will gather more data points for essentially the same stroke than will a faster writer. To make the data independent of the writing speed, we have re-sampled the data points in a novel manner.

One common way to time normalize data is via the usage of interpolation. This needs the computation/fitting of the curve – in a dynamic sequence – and is pretty complex. It is not just about the complexity we need to think about. The critical question is: Is it required? Well, the method we have employed works just as good, but with far less computational complexity and cost.

Consider a case where a particular writer gathers 1204 data points on one of her samples. The aim is to reduce it to 500 points, which are then compressed again by the application of the Discrete Cosine Transform. Reducing 1204 points to 500 points can be done with interpolation. We do not want that. Fortunately, we could derive an innovative alternative that could circumvent interpolation. The thought process ran somewhat like this:

- Ignoring the last 204 points will result in1000 points.
- Picking every second point will result in 500 points.

The method works, but at the cost of a non-negligible truncation loss at the end of the sample. But since the idea was right in principle, I improvised on it by varying it a bit. What I realized was that if the number of data points that I ignore (at the end of the character sample) is too small compared to the number of data points that remain, the truncation loss should be quite negligible and innocuous. In the example that is introduced above, let us suppose we had 10,204 points to start with. In that case, leaving out 204 points (to arrive at a multiple of 500) would still leave us with 10,000 points. The truncation loss would be much lower. For a given handwriting speed, the number of data points we start with directly

depends upon the polling frequency set in the program. Therefore, in order to minimize the truncation loss, we can simply increase the program polling frequency.

There is no way to have a prior knowledge of the number of data points a particular writer will accumulate for a particular character sample. Therefore, a simple algorithm to carry out the time normalization has been adopted:

Step 1: Count the number of data points in the character sample. Call it N.

Step 2: Compute T = N mod 500. T would be the number of data points (at the end) to be left out.

Step 3: Compute P = $\lfloor N/500 \rfloor$. Every P^{th} data point is to be picked from the data point set remaining after the completion of Step 2.

Completion of Step 3 would leave us with 500 data points, regardless of the character sample size, shape, or the writer's speed. The data is said to be time normalized.

4.4 Feature Extraction

After smoothing, de-noising, and three types of normalizations, the raw data (resulting from unconstrained, occasionally haphazard, writing) is made to fit some format. Now that all the character samples are in a position to be compared with each other, in a similar structure, let us see how we have recognized the best feature for the

sample set. The selection and extraction of the feature form the core step before the final classification routine can be put in place.

In order to better appreciate the role of feature extraction, it may be a good idea to go one step backward – by starting at dimensionality reduction (which is a higher level topic). Dimensionality reduction is a concept that frequently appears in the broad area of pattern recognition. Simply put, it deals with the reduction in the number of independent random variables in the system. Two cases of dimensionality reduction are *feature selection* and *feature extraction*.

Martin Sewell captures the essence of feature selection as follows [30]: "Feature selection (also known as subset selection) is a process commonly used in machine learning, wherein a subset of the features available from the data are selected for application of a learning algorithm. The best subset contains the least number of dimensions that most contribute to accuracy; we discard the remaining, unimportant dimensions."

In the case of supervised learning (which is what has been employed in the work), feature selection is known to require a complete listing of all possible subsets of features. Thus, feature selection is rendered impractical as the number of features gets large. To retain practicality, the listing is limited to a satisfactory set of features instead of an optimal/ideal set of features.

A further look into feature selection reveals the two big categories that make up the technique – *feature ranking* and *subset selection*. Feature ranking, as the name

indicates, is merely a ranking of the features by a performance measure. The features that don't perform to a pre-set standard are crossed out. Subset selection searches for the optimal/ideal set from within the (narrowed down) set of possible features. However, in this work, we have not made use of feature selection.

Moving on to the technique that is actually used, let us understand what feature extraction is all about. Feature extraction involves transforming the input data into a *feature vector*. A feature vector is just a reduced representation of the data, which is sufficient to recognize the data. The reason for the sufficiency claim is that most forms of data have a large chunk of redundancy. This redundancy carries no extra information that helps recognize the character sample. So the main idea is to eliminate as much redundancy as we can, and use the skeleton left for the classification purposes.

Feature extraction involves a fairly accurate description of the data with just a few points. The Discrete Cosine Transform (Type II), which has had a sufficient introduction in the chapter on preliminaries, is used for creating the feature vector for each character sample. The DCT produces a DC component, which is not much helpful in differentiating samples (which is essentially what is done in the classification step). Apart from the DC component, the DCT compresses the x- and y-coordinates of the data. Specifically in this case, what this means is that a sample which is originally represented by 500 data points, is now squished to a 10 data-point plot. The plot is converted to a 20x1 vector of the 10 x-coordinates followed by the corresponding y-coordinates. Each character sample gets to have its own 20x1 feature vector, which it can hang on to during the process of differentiation and identification, better known with the domain jargon, as *classification*.

Apart from the DCT, online handwriting recognition offers a choice of several temporal features too (which is where the bulk of the advantage of resorting to online recognition comes from). The speed of writing (and variations thereof), angular velocity, and the order of strokes – just to name a few – are excellent features that can complement the DCT, should the situation demand extra feature-extraction (usually as a feedback from the classification step). A similar supporting role can also be played by spatial features such as shape contexts [31], and results from Principal Component Analysis.

4.5 Classification

The next logical step after feature extraction is the classification of the feature vector that represents an input character into one of the 46 character classes as outlined in chapter 2. For most practical purposes, this is the final step and although the choice of feature/(s) has a larger bearing on the overall accuracy and feasibility of the HWR algorithm, the selection of an appropriate classifier ensures low misclassification and rejection rates.

In this thesis on online Devanagari handwriting recognition, we are dealing with isolated characters, each of which is represented by the feature vector –

$F = \{X_{d1}, X_{d2}, \ldots, X_{d10}, Y_{d1}, Y_{d2}, \ldots, Y_{d10}\}$ ....

This feature vector with its 20 elements acts as the input to the classifier. The classification rule can be summarised as follows – given a set of input-output pairs $S = \{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\}$, the classifier should compute a function (f), based on

these input-output pairs, which when applied to an unknown input x should give an output y' = f(x), where y' is as close to the actual output y as possible. The idea is that the input-output pairs in S portray a behaviour that can be captured statistically and therefore such classifiers are called statistical classifiers.

These statistical classifiers can be broadly classified into types – those that are based on supervised learning paradigm and those that are based on unsupervised learning (see section on ANN for more on these learning paradigms). It is very clear that supervised learning is the one we need to be looking at, for solving our very particular case of pattern recognition with well- defined character classes as labelled outputs. Unsupervised learning based algorithms are made of use in the cases where either the target outputs are unknown or where only a few numbers of inputs have been labelled with their target outputs.

A representative (but not exhaustive) list of the popular approaches for contriving supervised learning as a pattern recognition tool are neural networks, Support Vector Machine (SVM), Hidden Markov Model (HMM), nearest neighbour algorithm and decision tree learning. There is no particular study that documents the effectiveness of these algorithms to specific problem areas. Considering a straight forward case as ours, we can safely assume that any of the above mentioned classifiers would do a good job of classifying a given input character based on DCT coefficients.

Even after narrowing down to the usage of Artificial Neural Networks as the sole classifier, there is still scope for a lot of freedom and experimentation. One of the primary decisions to be taken is the number of neurons to be used in each layer – input, hidden,

and output. In keeping with the output of the Discrete Cosine Transform, which is the feature extractor used in our work, the number of neurons used in the input layer has to be fixed to 20. Each element of the feature vector (F) is an input to the neural network.

This leaves us with the hidden and output layers. First, let us look at the output layer. There are 46 characters in the Devanagari alphabet set. This means that any neural network output must have at least 46 different combinations, one for each unique character. Every output neuron can have a value between -1 and +1 (for the *tan-sigmoid*), but is trained to go to either 0 or 1 depending upon the character being trained for, and the character whose result is being displayed by the neuron. The net effect is that each output neuron is trained to behave like a binary digit (bit). With this background, we can notice that the minimum number of neurons required to represent 46 combinations is $\lceil log_2(46) \rceil$ = 6. A network with just 6 output neurons showed a mediocre-to-dissatisfactory performance during testing.

The next thing that was tried was an output layer with six groups of three neurons each. It can be seen that a group of three neurons can represent up to 8 combinations. Six such groups would represent 48 combinations (which is two more than what we need). Totally, this approach would cost us 6*3 = 18 neurons. The performance of the network improved with this configuration, but not by a great extent. Along the same lines, three groups of four neurons each – again capable of 48 combinations were used. This approach cost 4*3 = 12 neurons, but the performance level did not even match the 6x3 configuration.

Finally, one neuron was set aside for each combination, thus using 46 neurons in the output layer. It is this output layer configuration that yielded the least classification errors, showing promise for all configurations of the hidden layer.

The main engine responsible for classification is the hidden layer, making it extremely crucial that we design the hidden layer with great care and responsibility. We started with the general idea (based on several empirical tests) that the performance of the system increases monotonically with increase in the number of neurons in the hidden layer of our neural network. With this assumption, we set out with 20 neurons in the hidden layer. The performance (by performance, we mean accuracy of classification) of the neural network was not much impressive. The test process was repeated with 30 neurons, and the performance did not improve significantly. However, with 40 hidden layer neurons, the accuracy rate climbed over 90%. 50 hidden layer neurons (with 46 output neurons) gave an accuracy rate of more than 94.5%. So far, the monotonic behaviour of performance (versus number of hidden layer neurons) was confirmed. However, this pattern was broken when we further stepped up the number of hidden layer neurons to 60.

To further better the performance of the network, we created one more neural network with identical input and output layer configurations. The only thing I changed was the number of hidden layer neurons, from 50 to 55. I, then, used the sum of the outputs of the two networks to classify the data. The keen reader can observe that this is not a ranking scheme or an explicit voting methodology. Though the outputs of both networks are considered to make the final classification, since there is no decision being

taken at each network, this is different from standard voting schemes. It turns out that this performs better than elementary forms of voting. The performance with this setup crossed 96%.

Just to experiment a bit more, one more neural network was created with 45 hidden layer neurons. The outputs of all three neural networks were summed, and the summed output vector was used for classification. This three-network configuration performed best, yielding an accuracy rate of more than 97%. More on the results (with specific comparisons) will be presented in the chapter on results.

CHAPTER V

ANALYSIS OF RESULTS

It is only logical to have a numerical, quantitative, visual backing for the method proposed in the previous chapter, without which it would all be dry theory. A whole lot of character samples have been rigorously trained and tested(whose details would follow in this chapter), but it would be impossible to capture all the individual results here. Therefore, what we will attempt to do here is to show how each step in the proposed method actually works (with relevant data and visual aid), along with some consolidated numbers that indicate the impact these steps jointly have over the recognition rate of the system built.

Thus, this chapter is organized to go over two broad items in a mostly serial fashion:

1. The working of individual steps in this method, to evince the correctness/effectiveness of the sub-methods employed,

2. The overall effectiveness brought about by these procedures, measured in terms of the accuracy of recognition of the test-set samples.

5.1 Pre-processing

If you recall from the previous chapter, the first operation to be done on the input data is smoothing and de-noising. As mentioned earlier, noise is *normally* introduced in mechanical and physical processes more than just software data transfers and processing. It is due to this reason that in the several examples that have been reviewed to include in the current part of the thesis, not in a single one could we spot a random entry of noise.

As a consequence, in Fig. 5.1, below, the reader can see that the first operation, though titled “smoothing and de-noising”, shows only smoothing, since there wasn’t any noise to be removed – though the perfectly crafted de-noising routine was there in place.

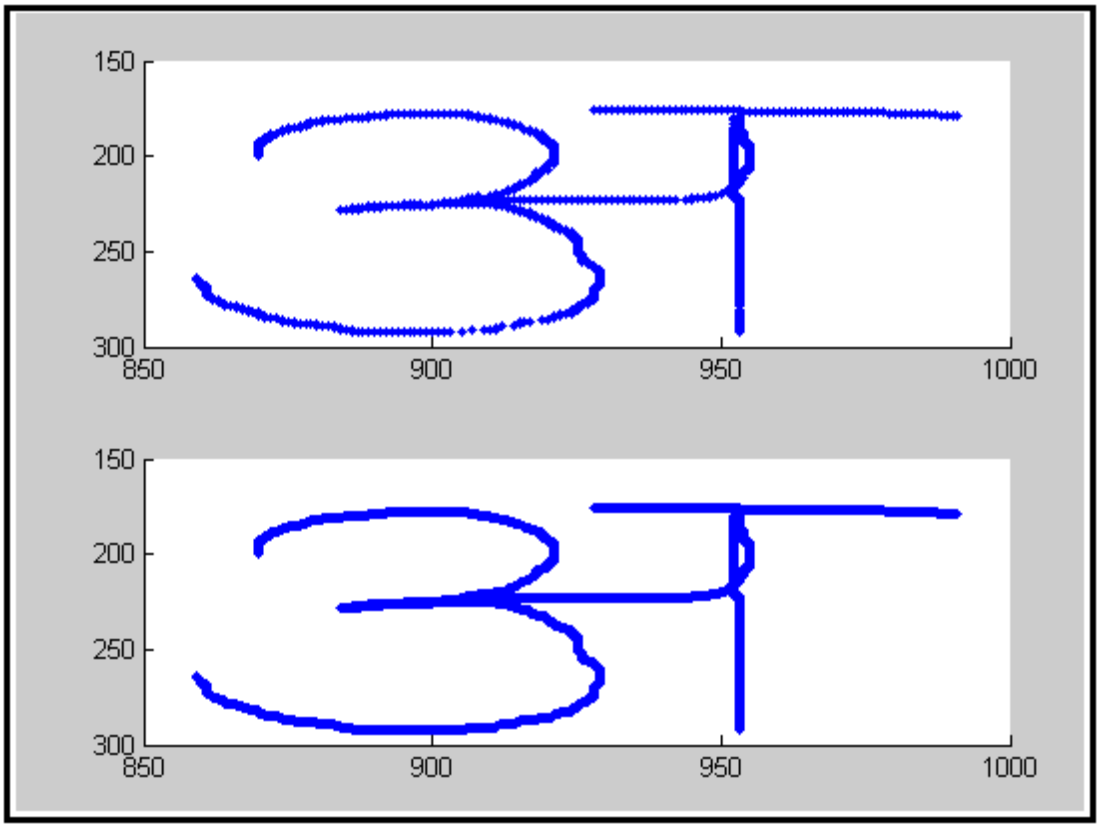

Fig. 5.1: Top: Input character sample; Bottom: The smoothed version of the character sample.

It is clear that the smoothed version of the data sample has a lot more data points than the original sample, which is not entirely desirable. However, the sample in bottom portion of the above figure is not the final data sample that goes into the recognition engine later on.

Next up, let us see how the smoothed data sample is re-positioned to push it the tiny corner vertexed at the origin. The same character example as introduced above, when translation-normalized is in Fig. 5.2.

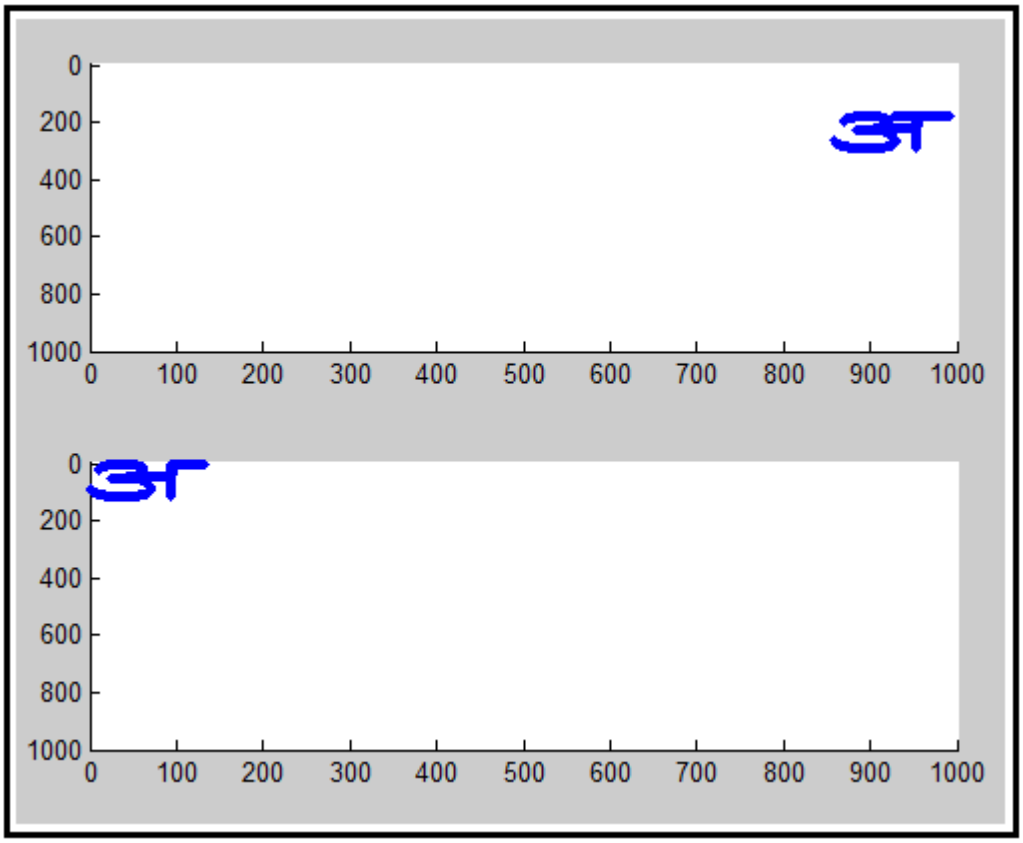

Fig. 5.2: Effect of translation normalization on the smoothed character sample.

Next in line, waiting impatiently to squeeze or stretch the character samples, is the routine for size normalization. Statistically speaking, almost all of the samples written in the database are taller (allowing for some degree of personification) than the stipulated 50 pixels. So, most of the samples have been squeezed rather than stretched.

In the following example figure, as the reader can see, the character which originally measured more than 120 pixels in height, is squeezed down (not *cropped*

down) to exactly 50 pixels in height. As mentioned in the previous chapter, the width of image, though, is not bound by limits.

The result of size normalization on the smoothed and translated character sample is captured in the following figure. In order to preserve the clarity of the image, especially when we are sizing it down by a factor up to 3-4, images have not been plotted in standardized axes.

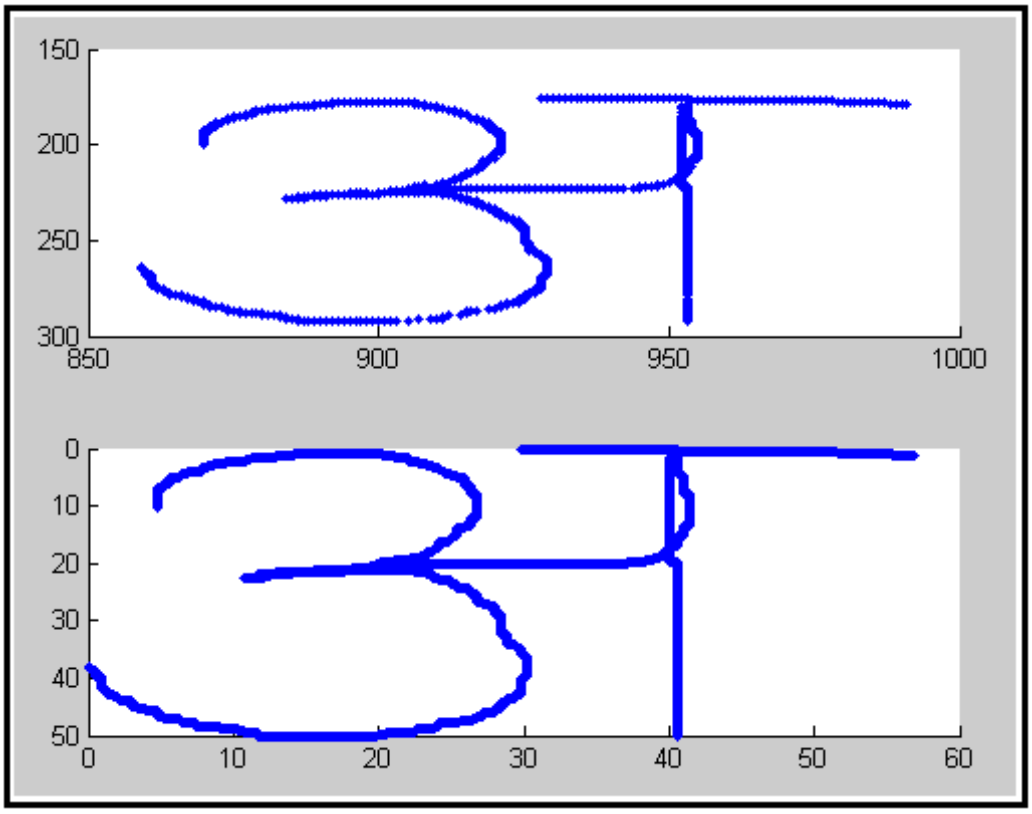

Fig. 5.3: The character sample – after smoothing, translation-normalized, and size-normalized.

Now comes the part which could leave the reader a little perplexed regarding the whole philosophy and rationale behind initially smoothing followed later on by re-sampling. The figure, that is about to follow, might not be of any help either – just going by visual reasoning. However, what should make thorough sense is that mechanism of reconstructing (smoothing) and tearing down the reconstruction by sampling it again proves to be a very simple way of ensuring the character samples have exactly a pre-set number of data points – which, in our case, is 500.

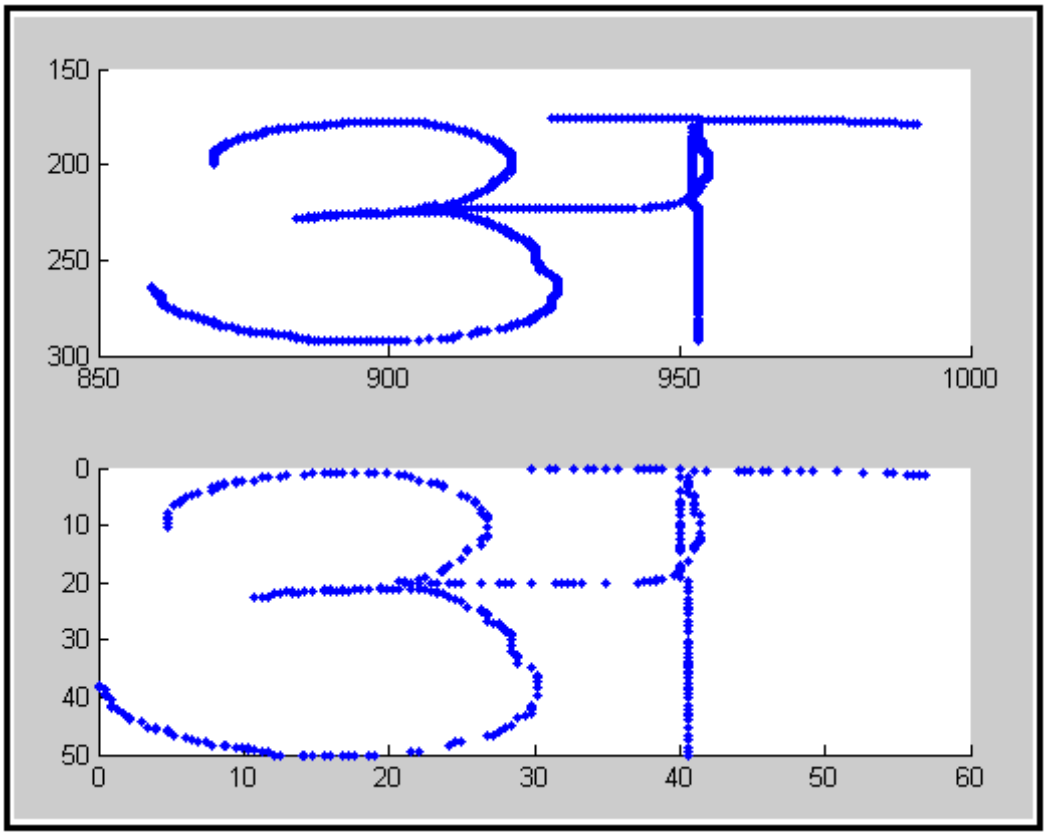

Fig. 5.4: The smoothed, translation-normalized, size-normalized, and re-sampled version of the original character sample.

The data is now ready to proceed to the penultimate step: feature extraction. Let us see what the DCT of the pre-processed data looks like.

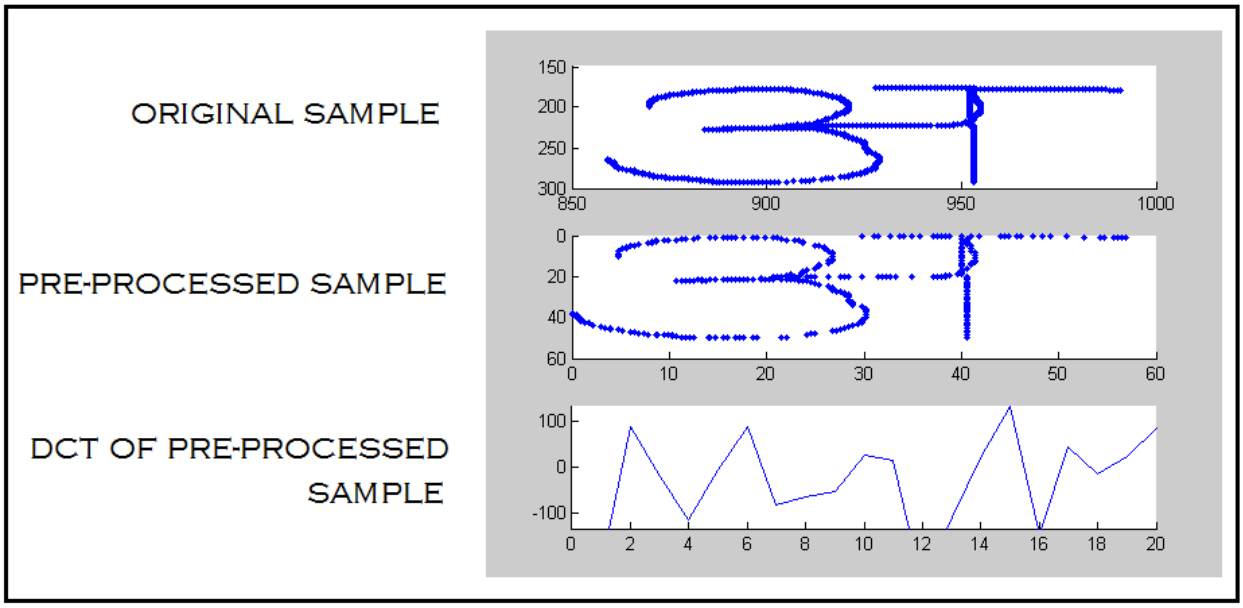

Fig. 5.5: Consolidated view of the original character sample, its pre-processed version, and the Discrete Cosine Transform (first ten x-coordinates appended by their respective y-coordinates – excluding the DC component) of the pre-processed sample.

The DCT is what is finally stored in the database, thereby reducing the memory consumption from 500 points per sample to just 20.

5.2 Performance Measurement

We have seen how each step in the pre-processing routine set affects the appearance of the character sample, all the way to the dimensionality reduction afforded by the computation of the DCT. Now, let us see how well the system has been built and trained in terms of the accuracy of recognition of 2760 characters – that come from writers who did not contribute to the training data set. The effect of *layering* on

recognition accuracy can be verified by testing smaller subsets of data (in lieu of the entire testable set of 2760 character samples). The intended effect is that the accuracy rate should be pretty close to the overall value, indicating a sense of homogeneity within the character sample spread. However, I should clarify that we have not resorted to such an extrapolation of accuracy tests, and have tested each of the 2760 character samples in the test set for *hit* and *miss*.

The first neural network used for the system comprised 50 neurons in the hidden layer, and worked better (in terms of accuracy) than the preceding work – whose formal comparison shall be made shortly.

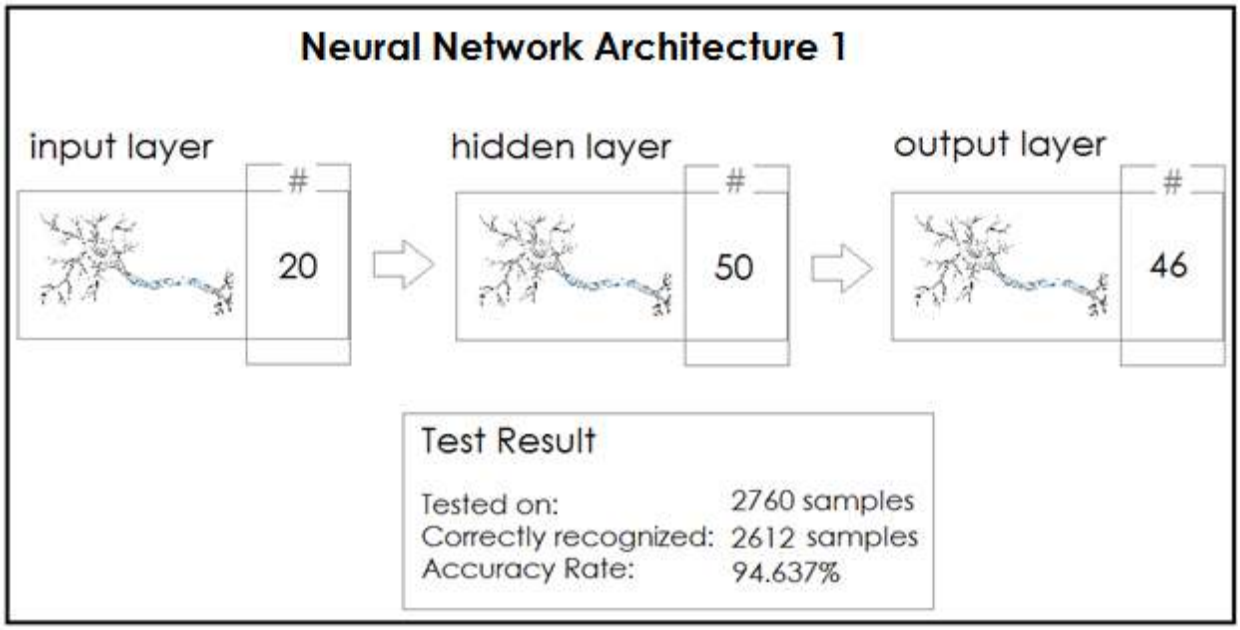

Fig. 5.6: The basic, single neural network architecture

I noticed that, though the neural network with 55 neurons in the hidden layer does not singly perform better than the one with 50 neurons in the hidden layer, a merger of

the topologies (by means of taking the average of the 46 outputs of each network) would pave way to better accuracy than the individual networks.

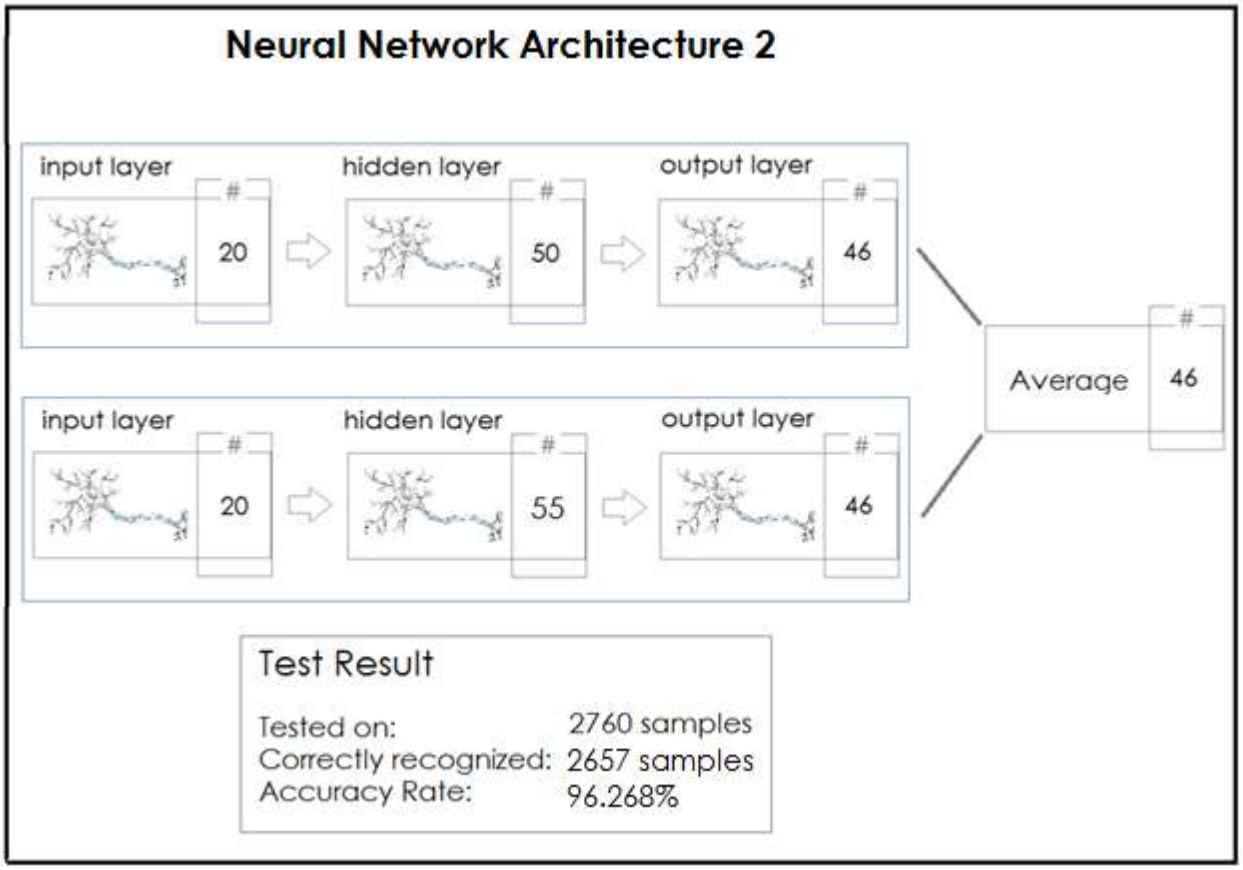

Fig. 5.7: An architecture with 2 neural networks

Next, we tried to extend the reasoning to include one more neural network topology with 45 neurons in the hidden layer, and the whole setup and performance are summarized in the figure below.

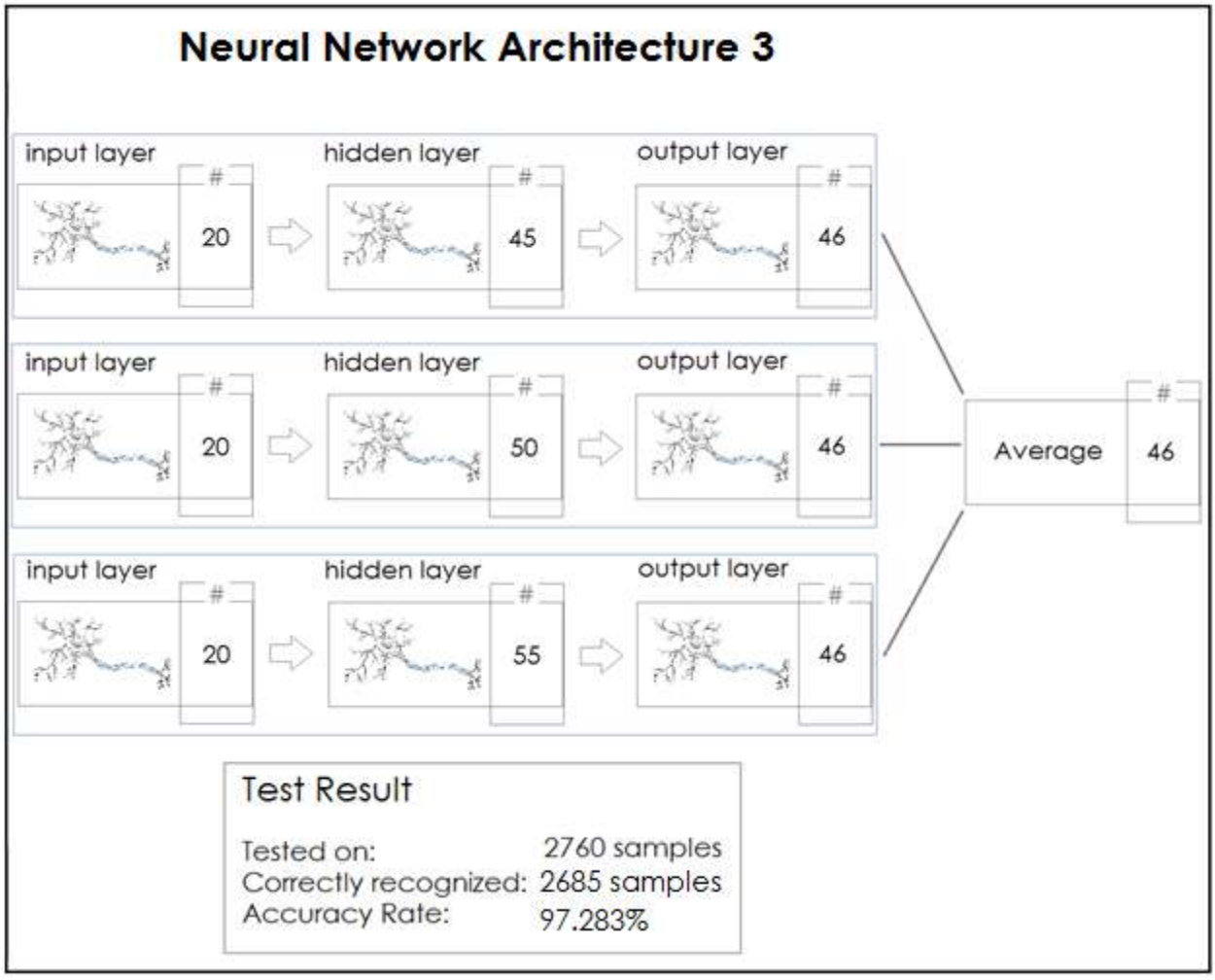

Fig. 5.8: An architecture with 3 neural networks

We did not want to continue the growth of the total number of hidden layer neurons (between networks), and decided to experiment a bit with grouping neurons in the output layer. Instead of using 46 neurons to cater to an output vector space of identical cardinality, we decided to group output neurons in smaller numbers, so as to create as many combinations (assuming binary behaviour of neurons) as there are classes – 46, in our case. If the number of combinations exceeds 46 – which may be possible while grouping – the excess combinations indicate misclassification.

First, an architecture with 6 groups of 3 neurons each in the output layer was tested, then with 3 groups of 4 neurons each, and finally with just one group of 6 neurons.

The former two architectures have only 2 excess combinations than needed, while the last architecture (labelled "Neural Network Architecture 6") has 18 excess combinations.

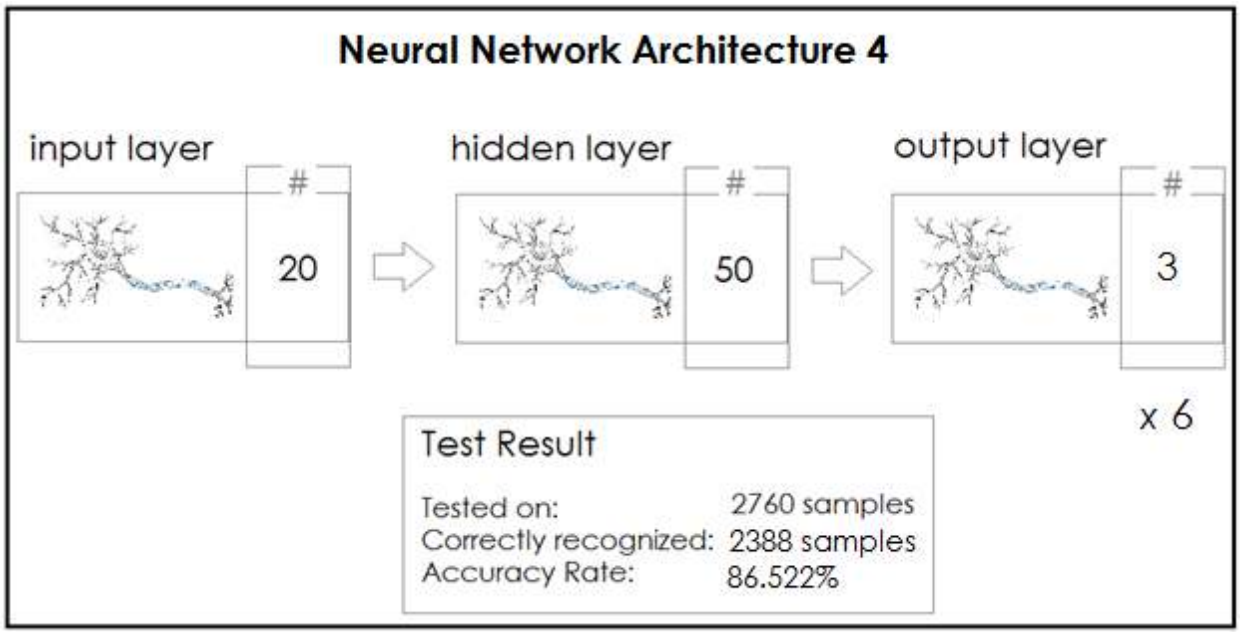

Fig. 5.9: An architecture with neuron grouping – 6 groups of 3 neurons each.

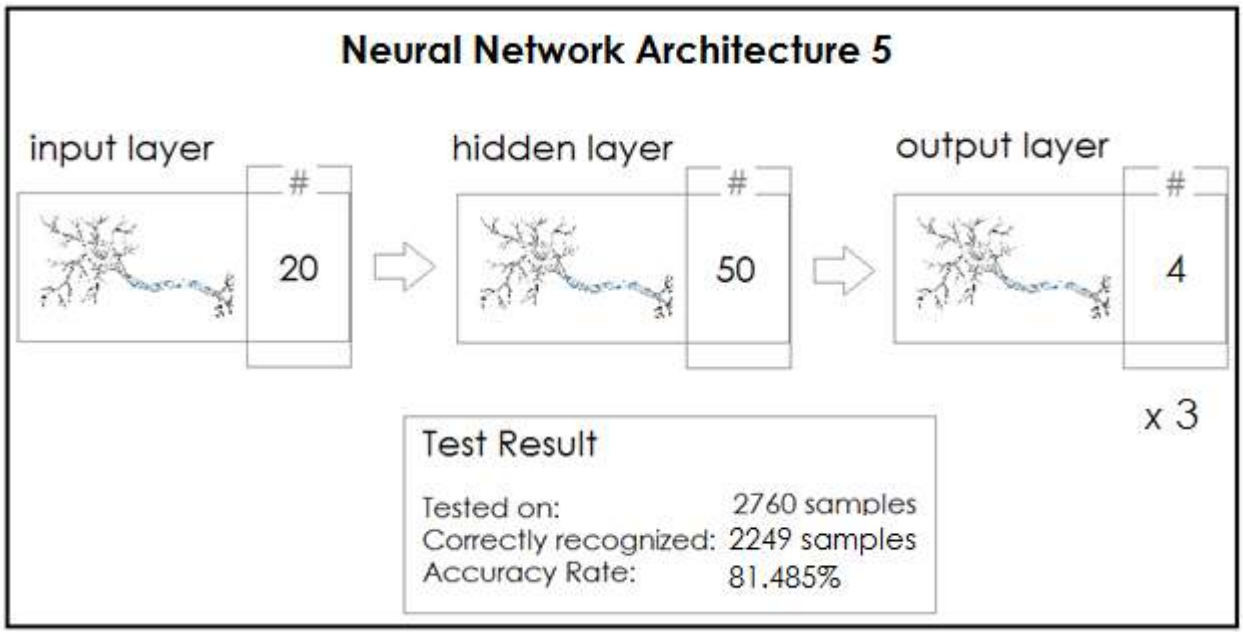

Fig. 5.10: An architecture with neuron grouping – 3 groups of 4 neurons each

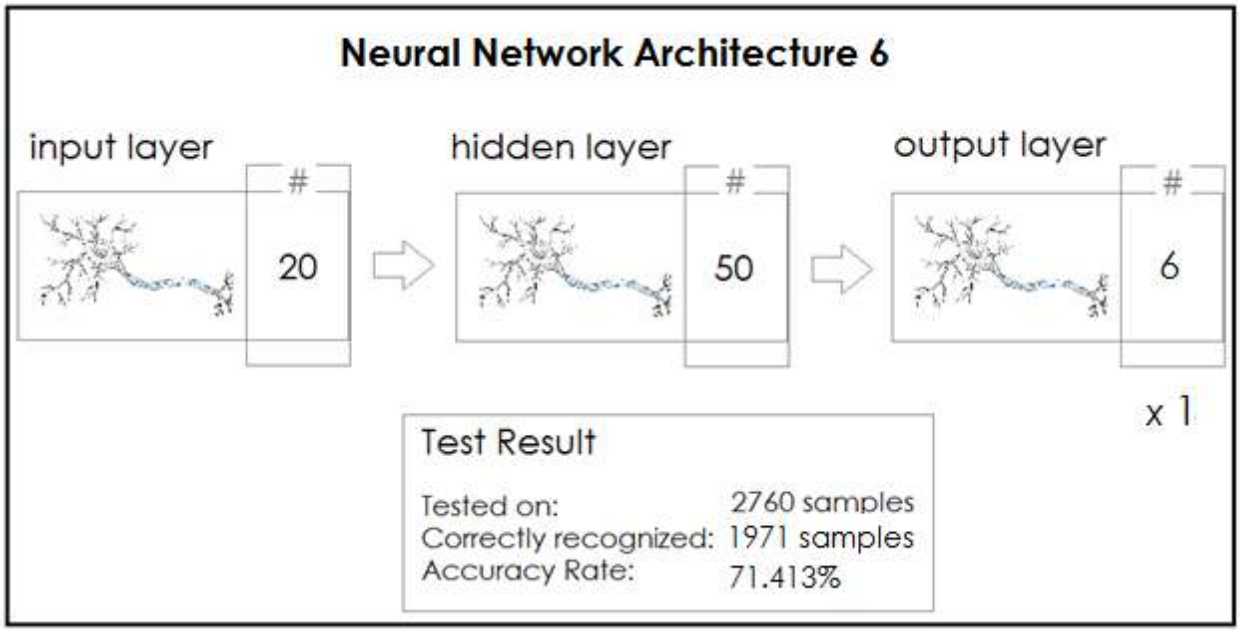

Fig. 5.11: An architecture with neuron grouping – 1 group of 6 neurons

We can see that the best performing architecture is the third one, labelled "Neural Network Architecture 3", with a whopping recognition rate of 97.28%. Next in line is the

neural network architecture with two neural networks, achieving over 96.2% accuracy. Though the neuron grouping experiment cuts down the number of neurons in the output layer, the performance takes a pretty severe hit.

Here is a comparison of how the proposed method stacks up against other methods, which we reviewed as part of the literature survey.

TABLE I: A comparison of published works and the proposed methods based on methodology, number of samples used per character for testing and accuracy rate.

Author(s)	Year	Method(s)	No. of samples/character	Accuracy Rate
Scott D. Connell, et al. [24]	2000	Offline and online features; HMM and Nearest Neighbour classifiers.	40	86.5
Anoop M. Namboodiri, et al. [25]	2004	11 features; 6 classifiers.	NA (since testing is done on test *pages*)	95.5%
Niranjan Joshi, et al. [26]	2007	Structural recognition; Subspace method.	25.71	94.49%
Abhimanyu Kumar, et al. [27]	2010	Curvature feature vector; 42 HMMs.	10	NA
Proposed	2011	DCT as feature extraction; 1 Neural Network	60	94.64%
Proposed	2011	DCT; 2 Neural Networks	60	96.27%
Proposed	2011	DCT; 3 Neural Networks	60	97.28%

It is to be noted that when neural networks of accuracy η are fused, an accuracy rate of $1-(1-\eta)^{ks}$ is approached, where k is the number of such fused neural

networks, and s is the saturation index, which becomes more prominent as k grows bigger.

CHAPTER VI

CONCLUSIONS AND RECOMMENDATIONS

6.1 Summary of Contributions

This thesis set out to implement a unique recognition system for handwritten Devanagari characters taking into account the geometrical variations in the script, the primary motive being the accomplishment of higher recognition rate than what were previously achieved. The method delineated in this thesis that helped in achieving our motive and the actual results can be summarized as follows

- We captured the time stamped co-ordinates of the characters and build a unique database. The database consisted of 5520 character samples (2760 training samples and 2760 testing samples) collected from 12 people. We wish to clarify that none of these 2760 characters used for testing purposes were part of the training set.
- The feature set that consisted of DCT co-efficients of the time stamped co-ordinates was then used to train and test the neural network – based on resilient backpropagation algorithm.
- The first neural network we used for our recognition system comprised 50 hidden layer neurons, and this by itself was more accurate than the preceding work.
- We then fused 2 neural networks (with 50 and 55 hidden layer neurons), and lastly we fused 3 neural networks with 45, 50, and 55 hidden layer neurons. The fusing methodology is not a voting/ranking mechanism. It is

using the average of the corresponding output neurons in all the networks. This scheme has made our system more accurate.

- Based on these experimental results, we can observe that the best recognition rate of 97.28% occurred when we fused 3 neural networks averaging 50 neurons per hidden layer. However, even when one single neural network is considered, the recognition rate is better than any previously published work

6.2 Potential for Future Research

The research undertaken as part of this master's program, as presented in this thesis, is a complete, uncompromised and an accurate solution to the problem originally defined at the beginning of the program. Due to the very nature of any master's program, the critical parameter which governs the quantity of work that can be done *beyond* solving the original work is time. While the quality has in no way been compromised, we fully understand that there can be a few additions in scope to the solution presented here in this thesis, and that the area of handwritten character recognition is far from being saturated.

With the presence of a looser time bound, an interested researcher may consider implementing several useful add-ons to this work:

1. Handwritten word recognition: The easiest extension would be to recognize one whole word - a task which would need character segmentation. The add-on could be as simple as a word recognizer, or could go on to incorporate a simple word-level translator (with the aid of a look-up table). While the latter implementation is a contained solution in itself, one could make heightened sense of the idea if the ultimate aim is to go beyond the word-level, onto sentence- and manuscript-levels.

2. Handwritten complete sentence recognition and translation: This extension would appear more complicated than the previous one. While it is partially true, the task is simplified when the script under consideration is Devanagari. Fortunately, Devanagari stipulates the use of a Vinculum (termed "*Shirorekha*" in Sanskrit) over the word. Therefore, the end of the Vinculum would signal the end of the word, making inter-word segmentation a rather simple procedure.

However, the problem would go beyond recognition of each word. A series of recognized words, more often than not, make little sense. Since the main purpose of recognizing an entire Devanagari sentence could be assumed to be an eventual translation into a more widely used language (say English), the arrangement of the recognized words pose a bigger problem. This problem would arise mainly due to the differences in grammatical structuring between Devanagari and the terminal language. At this point, the ambitious researcher might make use of a dedicated algorithm to solve the grammatical mismatch problem. One such algorithm is presented in [32].

3. Manuscript translator: If the previous problem is solved, building a full-fledged manuscript translator would not be a daunting task. The grammatical ordering is a feature that is bounded within each sentence, and it poses no problem across sentences. A manuscript, at its most rudimentary description, is a well-defined collection of sentences. Thence, it can be assumed without loss of generality that the lexical and semantic integrity of any prose-based manuscript is well preserved.

These add-ons could be culturally or situationally influenced, and hence it is impossible to provide a complete list of such possibilities. However, the above listed extensions would have been the primary concern of the present work in a wider blanket of time and freedom.

REFERENCES

[1] http://ancientscripts.com/devanagari.html

[2] http://www.omniglot.com/writing/devanagari.htm

[3] Goodall, C. "A Survey of Smoothing Techniques." *Modern Methods of Data Analysis (J. Fox and J. S. Long, eds.) Newbury Park, CA: Sage Publications, pp. 126–176*, 1990.

[4] N. Ahmed, T. Natarajan and K. R. Rao, "Discrete Cosine Transform", *IEEE Transactions on Computers, pp. 90-93*, January 1974.

[5] Chellappa, R., Wilson, C., and Sirohey, S. "Human and machine recognition of faces: A survey", *In Proc. IEEE, 83(5):705–740*, 1995.

[6] Pratt, W, "Digital Image Processing", *2nd edition. Wiley: New York, NY*, 1991.

[7] R. Plamondon and S. N. Srihari, "On-Line and Off-Line Handwriting Recognition: A Comprehensive Survey," *1EEE Transactions On Pattern Analysis And Machine Intelligence. Vol. 22, NO. 1., pp.63 – 84*, January 2000.

[8] S.N.Srihari, "High Performance Reading Machines", *Proc. IEEE, Vol. 80., pp.1,120-1,132*,1992.

[9] G.Seni, R.K.Srihari, and N.Nasrabadi, "Large Vocabulary Recognition of On-Line Handwritten Cursive Words", *IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol 18, no.7, pp 757-762*, July 1996.

[10] A. Meyer, "Pen Computing : A Technology Overview and Vision", *SIGCHI Bulletin, Vol.27, No.3.*, July 1995.

[11] J. Subrahmonia and Thomas Zimmerman, "Pen Computing: Challenges and Applications", *15th International conference on Pattern Recognition, Vol 2, pp 60 – 66*, September 2000.

[12] C. Tappert, C. Suen, and T. Wakhara, "The State of the Art in On-Line Handwriting Recognition," *IEEE Transactions on pattern analysis and machine intelligence, Vol. 12, No. 8*, August 1990.

[13] T. L. Dimond, "Devices for reading handwritten characters," *Proceedings of Eastern Joint Comput. Conference, pp.232-237,* Dec. 1957.

[14] M. R. Davis and T. O. Ellis, "The Rand tablet: A man-machine graphical communication device," *Proceedings of FJCC, pp. 325-331,* 1964.

[15] J. F. Teixeira and R. P. Sallen, "The Sylvania data tablet: A new approach to graphic data input," *Proceedings of FJCC, pp. 315-321*, 1968.

[16] J. R Ward and T. Kuklinski, "A Mode l for Variability Effects in Handprinting with Implications for the Design of Handwriting Character Recognition System", *IEEE Transactions on Systems, Man and Cybernetics, Vol. 18, No.3,* May/June 1988.

[17] G Gaillat, "An on-line recognizer with learning capabilities", *Proc. 2nd Int. Joint Con& Pattern Recognition, pp. 305-306,* August 1974.

[18] J. M. Kurtzberg and C. C. Tappert, "Segmentation procedure for handwritten symbols and words", *IBM Tech. Disclosure Bull., vol. 25, pp. 3848-3852*, Dec. 1982.

[19] A. S. Fox and C. C. Tapper, "On-line external word segmentation for handwriting recognition", *Proc. 3rd Int. Symposium on Handwriting Comput. Appl.*, July 1987.

[20] H. Arakawa, K. Odaka and 1. Masuda, "On-line recognition of handwritten characters Alphanumeric, Hiragana, Katakana, Kanji", *Proc. 4th Int. Joint Con$ Pattern Recognition, pp. 810-812*, Nov 1978.

[21] W.Guerfali and R. Plamondon, "Normalizing and restoring online handwriting", *Pattern Recognition, Vol. 26, No. 3, pp. 419 - 431*, 1993.

[22] L. S. Yaeger, Brandyn J. Webb and R. F. Lyon, "Combining Neural Networks and Context-Driven Search for On-Line, Printed Handwriting Recognition in the Newton", *Association for the Advancement of Artificial Intelligence, Vol.19, No.1, pp. 73-89*, 2011.

[23] A. R Ahmad, M Khalid, C Viard-Gaudin and E Poisson, "Online Handwriting Recognition using Support Vector Machine", *Tencon 2004.2004 IEEE Region 10 Conference, Vol. 1, pp. 311-314,* Nov 2004.

[24] S. D. Connell, R.M.K. Sinha and A. K. Jain, "Recognition of Unconstrained On-Line Devanagari Characters", *Proceedings of 15th International Conference on Pattern Recognition, Vol 2*, 2000.

[25] A. M. Namboodiri and A. K. Jain, "Online Handwritten Script Recognition", *IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 26, No. 1*, January 2004.

[26] Joshi, N., Sita G., Ramakrishnan A.G., Deepu V., Madhvanath S., "Machine Recognition of Online Handwritten Devanagari Characters", *8th International Conference on Document Analysis and Recognition, Vol. 2, pp 1156-1160*, January 2005.

[27] A. Kumar and S. Bhattacharya, "Online Devanagari isolated character recognition for the iPhone using Hidden Markov Models", *IEEE Students' Technology Symposium, pp 300-304,* May 2010.

[28] http://www.csad.ox.ac.uk/csad/Newsletters/Newsletter10/Newsletter10c.html

[29] S. W. Smith, "The Scientist and Engineer's Guide to Digital Signal Processing," Chapter 15.

[30] http://machine-learning.martinsewell.com/

[31] S. Belongie and J. Malik, "Matching with Shape Contexts," *IEEE Workshop on Contentbased Access of Image and Video Libraries (CBAIVL)*, 2000.

[32] S. Kubatur, S. Sreehari and R. Hegde, "An Image Processing Approach to Linguistic Translation", *2nd International Conference on Methods and Models in Science and Technology, Vol. 1414, pp. 172-177,* November 2011.

[33] M. W. and Pitts W., "A logical calculus of the ideas immanent in nervous activity*", Bulletin of Mathematical Biophysics*, *1943.*

APPENDIX

Matlab code common to training and testing phase

```
load output.txt;
T = output(:,3);%extracting only time values
maxT = max(T);
r = maxT/20 ;
T = T./r;

[m,n]= size(output);
%disp(m);
%disp(output);
X = output(:,1);%extracting x co-ordinates
Y = output(:,2);%extracting y co-ordinates

subplot(3,1,1)
set(gca,'YDir','reverse');%to reverse y-axis
hold on
plot(X,Y,'.');%plotting the input data
hold off

windowSize = 9;                                    %smoothing(averaging)

X1 = filter(ones(1,windowSize)/windowSize,1,X); %the x and y
Y1 = filter(ones(1,windowSize)/windowSize,1,Y); %co-ordinates

%[s,t] = size(X1);
%disp(s);
outlier_count = 0;

%the following code is for deleting all outliers
for i = 1:m-1
    %if X1(i-1) == 0
      % continue
    %end
    if sqrt((X1(i)-X1(i+1))*(X1(i)-X1(i+1)) +  (Y1(i)-Y1(i+1))*(Y1(i)-
Y1(i+1)) )  > 7
        outlier_count = outlier_count + 1;
        A(outlier_count) = i;
    end
end

for j = 1:outlier_count
    n = A(j) - j +1;
    X1(n) = [];
    Y1(n) = [];
end

%translation and size normalization
MinX1 = min(X1);
```

```
MinY1 = min(Y1);


X1 = X1 - MinX1;
Y1 = Y1 - MinY1;
MaxX1 = max(X1);
MaxY1 = max(Y1);

AR = MaxY1/MaxX1;

r = MaxY1 / 50;
Y1 = Y1./r;
X1 = X1./r;
%X1 = (X1.*50)./(AR*MaxX1);

p=1;
k = mod(m,500);
k = m-k;
step = k/500;
 for l = 1:step:k
     X2(p) = X1(l);
     Y2(p) = Y1(l);
     p = p+1;
 end

[c,d] = size(X2);
disp(d);


subplot(3,1,2)
set(gca,'YDir','reverse');%to reverse y-axis
hold on

plot(X2,Y2,'.');%plotting the translated and normalised character
hold off

Xd2 = dct(X2);
ndct = size(Xd2);
Yd2 = dct(Y2);
Xd2 = Xd2(2:11);
Yd2 = Yd2(2:11);
Xd = [Xd2 Yd2];
test = Xd';
ymax = max(Xd);
subplot(3,1,3)
set(gca,'xlim',[0 20], 'ylim',[-ymax ymax]);
hold on
plot(Xd);%plotting the dct
hold off
```

VITA AUCTORIS

Shruthi Kubatur was born in Belgaum, Karnataka, India. She graduated from the National College – Jayanagar, Bangalore, India in 2006. She received her Bachelor of Engineering degree in Telecommunications Engineering at B.M.S. College of Engineering affiliated to Visvesvaraya Technological University, India, in July 2010. She is currently a candidate for the Master of Applied Science degree in Electrical and Computer Engineering at the University of Windsor and hopes to graduate in September 2012.