OLD ENGLISH CHARACTER RECOGNITION USING NEURAL NETWORKS

by

SATTAJIT SUTRADHAR

(Under the Direction of Ionut Emil Iacob)

ABSTRACT

Character recognition has been capturing the interest of researchers since the beginning of the nineteenth century. While the Optical Character Recognition for printed material is very robust and widespread nowadays, the recognition of handwritten materials lags behind. In our digital era more and more historical, handwritten documents are digitized and made available to the general public. However, these digital copies of handwritten materials lack the automatic content recognition feature of their printed materials counterparts.

We are proposing a practical, accurate, and computationally efficient method for Old English character recognition from manuscript images. Our method relies on a modern machine learning model, Artificial Neural Networks, to perform character recognition based on individual character images cropped directly from the images of the manuscript pages. We propose model dimensionality reduction methods that improve accuracy and computational effectiveness. Our experimental results show that the model we propose outperforms previous attempts as well as current automatic text recognition techniques.

INDEX WORDS: Machine learning, Neural Network, Image Recognition, Old English

2009 Mathematics Subject Classification: 68U10, 68T10, 62H35

OLD ENGLISH CHARACTER RECOGNITION USING NEURAL NETWORKS

by

SATTAJIT SUTRADHAR

M.S., Jagannath University,Bangladesh, 2011

B.S., Jagannath University,Bangladesh, 2010

A Thesis Submitted to the Graduate Faculty of Georgia Southern University in Partial Fulfillment of the Requirements for the Degree

MASTER OF SCIENCE

STATESBORO, GEORGIA

OLD ENGLISH CHARACTER RECOGNITION USING NEURAL NETWORKS

by

SATTAJIT SUTRADHAR

Major Professor:	Ionut Emil Iacob
Committee:	Zhan Chen
	Marcel Ilie
	Kevin Kiernan

Electronic Version Approved:
June 2018

DEDICATION

I dedicate this thesis to my late Father Shital Chandra Sutradhar and all my family members for their constant support and blessings all through my life.

ACKNOWLEDGMENTS

My heartiest thanks to Dr.Emil Iacob for being such a great mentor and kind human being and all the faculty members of Georgia Southern University's Mathematics Department.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS

$\mathbb{R}$	Real Numbers
$\mathcal{I}$	The set of all character images
$\mathcal{L}$	The set of Old English letters

CHAPTER 1

INTRODUCTION

1.1 PRELIMINARIES

We describe next the problem of recognizing characters from Old English manuscript images. In order to provide a solution to this problem, we propose a model that mimics the human brain functionality, the Artificial Neural Network (ANN).

1.1.1 OLD ENGLISH MANUSCRIPTS

Our work is motivated by performing simple and accurate character recognition in Image-Based Electronic Editions of historic documents, which are important resources for humanities scholars and the general public. These editions can provide at the same time any number of researchers simultaneous first-hand access to digital images of unique and fragile material that is not otherwise widely available for study. Such electronic editions of historic materials (for instance, the Electronic Beowulf 4.0 [5]) typically combine the original manuscript images and the textual content of the manuscript (in an edited or non-edited form). The manuscript text extraction is manually performed by humanities scholars and takes a fairly large amount of time. The automatic printed text recognition (typically by using modern machine learning techniques) has advanced with the development of sophisticated methods. However, the automatic manuscripts handwritten character recognition has lagged behind. Moreover, searching for manuscript information, which makes sense in both text and image contexts (Figure 1.1), is problematic, in practice, for searching manuscription images. This is mostly because that the current and most accurate machine learning techniques for handwritten characters rely on large amounts of training data for creating a viable model. Our study aims to produce a simple but accurate model for manuscript character recognition using a smaller training set of character images. Such a model would

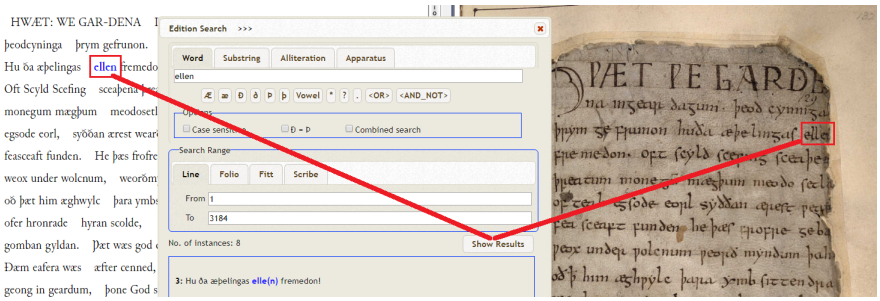


Figure 1.1: Searching for manuscript information makes sense in both text and image contexts

have important practical benefits, such as automatic manuscript text extraction combined with searching manuscript images. A natural model for our problem would be the human brain: how does the human brain perform character recognition?

In the subsequent section we introduce the idea of an Artificial Neural Network, which we choose as a model for our problem.

1.1.2 A HUMAN BRAIN MODEL

The idea of a neural network came from the way the human brain processes information. Many attempts have been made to reach the goal of training a machine to act like a human or even better. The fundamental thought is to train a machine with the already known information leading to a specific result to make its own decision for the unknown. The hopes of Artificial Neural Network (ANN) has its ups and downs. With the computational constraints and scarce resources it has always been a tough task. However, The Dreamers dreamt and the Artificial Neural network has always kept the intrigued afloat. In common ANN implementations, the signal at a connection between artificial neurons is a real number, and the output of each artificial neuron is calculated by a non-linear function of the sum of its inputs. Artificial neurons and connections typically have a weight that adjusts as learning proceeds. The weight increases or decreases the strength of the signal at a connec-

tion. Artificial neurons may have a threshold such that only if the aggregate signal crosses that threshold is the signal sent. Typically, artificial neurons are organized in layers. Different layers may perform different kinds of transformations on their inputs. Signals travel from the first (input), to the last (output) layer, possibly after traversing the layers multiple times.

The original goal of the ANN approach was to solve problems in the same way that a human brain would. However, over time, attention focused on matching specific tasks, leading to deviations from biology. ANNs have been used on a variety of tasks, including computer vision, speech recognition, machine translation, social network filtering, playing board and video games and medical diagnosis.

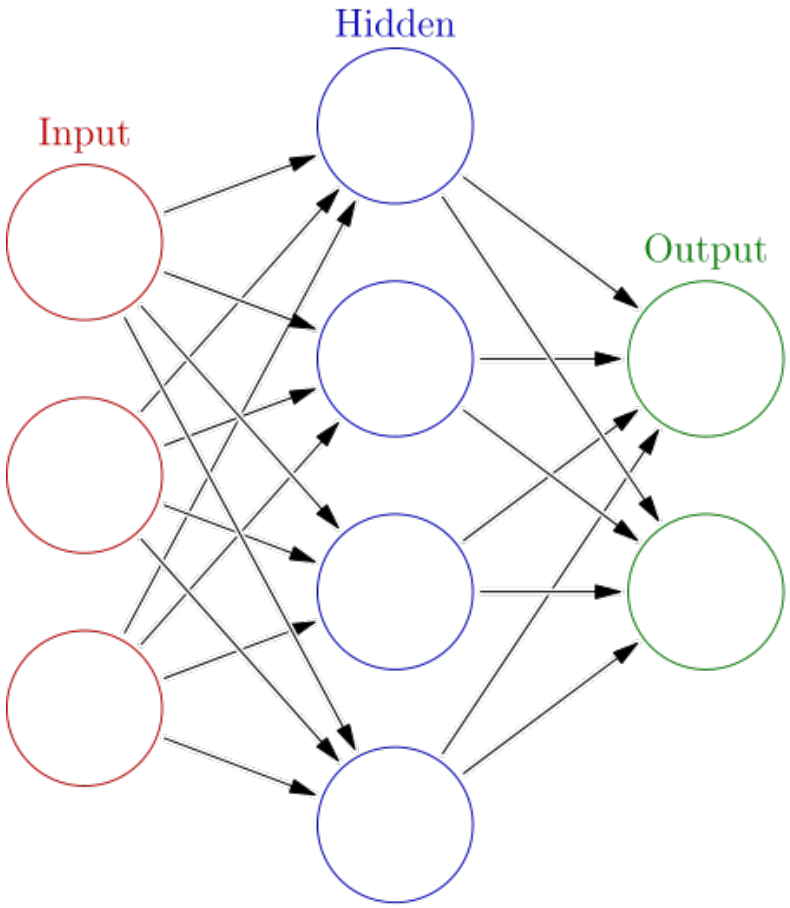


Figure 1.2: An Artificial Neural Network (ANN) with one hidden layer.

An artificial neural network is an interconnected group of nodes, akin to the vast network of neurons in a brain. Here, each circular node represents an artificial neuron and an arrow represents a connection from the output of one artificial neuron to the input of another.

We proceed next to formally define our problem and the Artificial Neural Network model.

1.2 THE CLASSIFICATION PROBLEM

Our main goal is to create a mathematical model for recognizing manuscript characters, based on the collected character image samples. We denote by $\mathcal{I}$ the set of all such character images and $\mathcal{L}$ the set of all letters they represent. For the purpose of character recognition we make the assumption that there exists a non-linear relationship between the set of such images and the set of character letters each image represents:

$$F : \mathcal{I} \to \mathcal{L} \tag{1.1}$$

The *Machine/Statistical Learning Classification* problem is the task of identifying the unknown category or class (out of a list of categories or classes) of an observation, given the known categories (classes) of a set of observations (training dataset). When the decision is between two classes, we call the problem binary classification. When there are more than two classes, it is called multiclass classification.

A *classifier* is the mathematical model (often a function) that takes as input a new observation and produces as output the class of the observation. We would like to create a classifier that takes as input a character image and produces the letter corresponding to the character image. We are therefore aiming to produce a multiclass classifier.

Let us first introduce a formal definition for the binary classification problem for the case of the input variables from the Euclidian space.

Definition 1. [Binary Classification in Euclidian Space] Let $S, T \subseteq \mathbb{R}^n$ be finite, disjoint sets. The binary classification problem consists in finding a function $f : \mathbb{R}^n \to \{-1, 1\}$ such that

$$f(x) = \begin{cases} 1, & \text{if } x \in S \\ -1, & \text{if } x \in T \end{cases}$$

An intuitive, well-known solution for this problem would be finding a hyperplane $\boldsymbol{w}^T \boldsymbol{x} - w_0 = 0$, where $\boldsymbol{w} \in \mathbb{R}^n$ is a constant vector (n components), $w_0 \in \mathbb{R}$, and $\boldsymbol{x} \in \mathbb{R}^n$

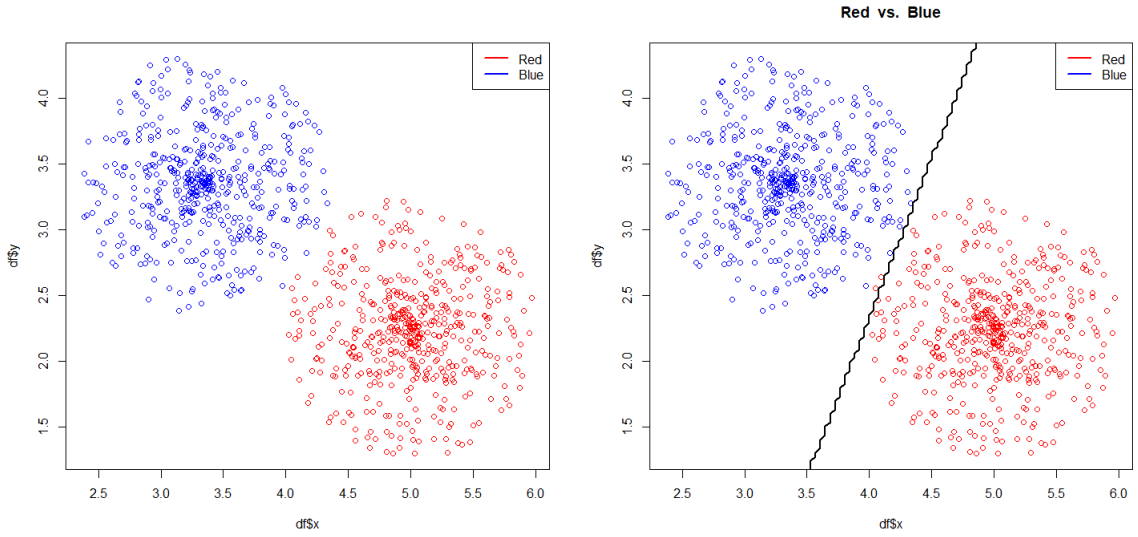


Figure 1.3: Linear solution to the binary classification problem

is a variable (n components), such that:

$$\boldsymbol{w}^T \boldsymbol{s} - w_0 > 0, \quad \forall \boldsymbol{s} \in R$$
$$\boldsymbol{w}^T \boldsymbol{t} - w_0 < 0, \quad \forall \boldsymbol{t} \in B$$

Such a solution is represented in Figure 1.3 and can found, for instance, as a solution of a LP problem [7]. Figure 1.3-left shows two sets of points in $\mathbb{R}^2$ (say S = red and T = blue) then Figure 1.3-right shows a hyperplane (the line) that separates these sets. If a new, uncolored point is given in $\mathbb{R}^2$ then the color of this new point can be easily predicted using its relative position to the line: if on the left-hand-side the new point is classified as blue, else it is classified as red. While the binary classification problem illustrates the classification problem in a very intuitive way, it is clearly not enough to model our problem. We therefore generalize Definition 1 as follows.

Definition 2. [Multi-class Classification in Euclidian Space] Let us consider C finite, disjoint sets $S_1, S_2, \ldots, S_C \subseteq \mathbb{R}^n$. The multi-class classification problem consists in finding a

function $f_C : \mathbb{R}^n \to \{1, 2, \ldots, C\}$ such that

$$f_C(x) = \begin{cases} 1, & \text{if } x \in S_1 \\ 2, & \text{if } x \in S_2 \\ \ldots & \\ C, & \text{if } x \in S_C \end{cases}$$

In other words, we want to determine a function capable of idenfying the set (or class) an x element belongs to. Our function must be determined without any prior knowledge about the data. Rather, we will be using machine learning techniques to *learn the model from data, or train the model*. We will build a model based on data for which prior class knowledge exists (image characters for which the letter is known) then subsequently use the model to identify which characters some other images represent.

1.3 Artificial Neural Networks

We propose the Artificial Neural Network based model and rely on standard machine learning techniques to train our model for performing character recognition for character images extracted from Old English manuscripts. Figure 1.4 shows a general Artificial Neural Network (ANN) model with one hidden layer and a single output. For simplicity, in this section we formally define an ANN model with a single output, but the extension to multiple outputs (predictions) follows naturally from this model and it will be presented in detail in Chapter 2. For an ANN model as in Figure 1.4 with N inputs, one hidden layer with L neurons, and a single output y, the model is described by:

$$\begin{gathered} y : \mathbb{R}^n \to (0, 1) \\ y(x_1, \ldots, x_n; w_{11}, \ldots, w_{NL}, z_1, \ldots, z_L, b_{11}, \ldots, b_{1L}, b_2) = \\ \sigma\left(\sum_{j=1}^{L} z_j \sigma\left(\sum_{i=1}^{n} w_{ij} x_i + b_{1j}\right) + b_2\right) \end{gathered} \tag{1.2}$$

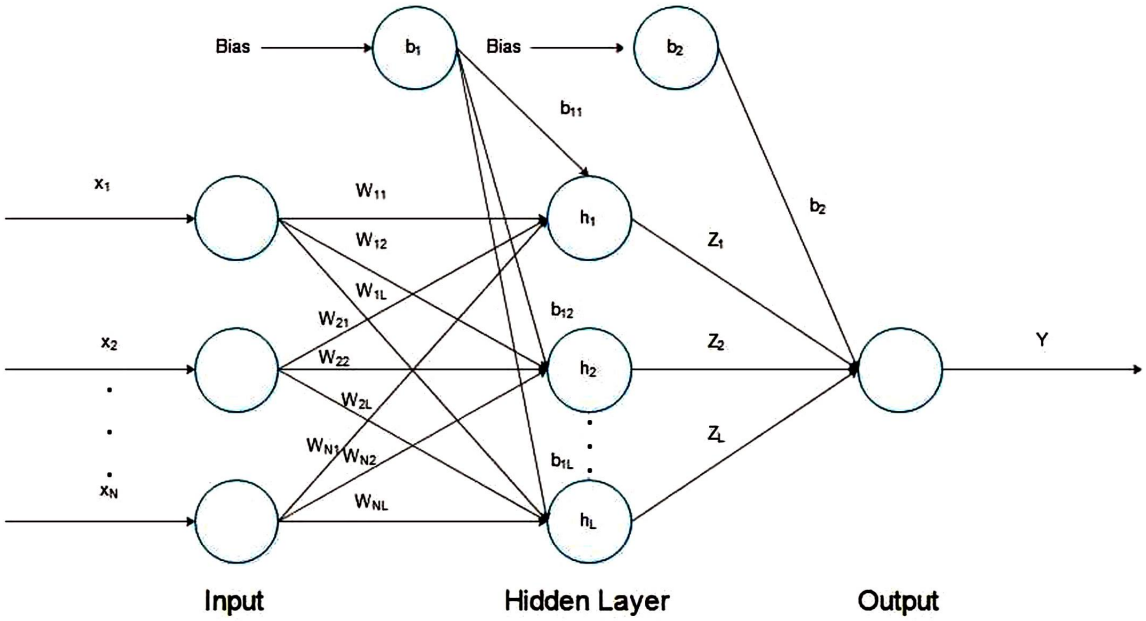


Figure 1.4: An Artificial Neural Network with one hidden layer and a single output for binary classification

where n is the input space dimension, L, $w_{11}, \ldots, w_{NL}, z_1, \ldots, z_L, b_{11}, \ldots, b_{1L}, b_2$ are the network parameters and will be determined experimentally, and $\sigma()$ is called the *activation function*, which in our model is the *sigmoid function*:

$$\sigma : \mathbb{R} \to \mathbb{R}, \quad \sigma(x) = \frac{1}{1 + e^{-x}}$$

Let us consider again two finite, disjoint sets $S, T \subseteq \mathbb{R}^n$. Using a subset of NS samples $\{(x_{i1}, \ldots, x_{i,n}, e_i) \mid i = 1 \ldots NS\} \subset (S \times \{1\} \cup T \times \{0\})\}$ (typically uniformly distributed over the sets S, T) of the experimental data (where $x_{i1}, \ldots, x_{i,n}$ represent the sample's measurement values and $e_i \in \{0, 1\}$ is the sample class), the parameters

$$w_{11}, \ldots, w_{NL}, z_1, \ldots, z_L, b_{11}, \ldots, b_{1L}, b_2$$

in (1.2) are determined as the optimal solution of the unconstrained optimization problem:

$$\begin{gathered} \min \sum_{i=1}^{NS} (-e_i \log y_i - (1 - e_i) \log(1 - y_i)) \\ (\text{where } y_i = y(x_{i1}, \ldots, x_{i,n}; w_{11}, \ldots, w_{NL}, z_1, \ldots, z_L, b_{11}, \ldots, b_{1L}, b_2)) \end{gathered} \tag{1.3}$$

We must notice that the ANN model represented by the output of function (1.2) computes a continuous value in the interval $(0, 1)$. One can interpret that value as an estimated probability that the output is zero or one. To produce a binary classification, in practice, a threshold T is introduced: if output $y \geq T$ then prediction is 0, else prediction is 1. Formally, we define our ANN prediction model as follows.

Definition 3. [Binary classification using ANN] For the given experimental data, an ANN model as in (1.2) with parameters computed using (1.3), we define the ANN binary prediction model as:

$$\begin{gathered} \widehat{F} : \mathbb{R}^n \to \{0, 1\} \\ \widehat{F}(x_1, \ldots, x_N) = \begin{cases} 1, & \text{if } y(x_1, \ldots, x_N) \geq T \\ 0, & \text{if } y(x_1, \ldots, x_N) < T \end{cases} \end{gathered} \tag{1.4}$$

where T denotes the decision threshold (typically, $T = 0.5$).

A multi-class classification ANN model is a generalization of the binary classification model. The ANN in Figure 1.4 is enhanced with multiple outputs, one output for each class in $\{1, 2, \ldots, C\}$. Each output will produce a continuous value in $(0, 1)$ as before, which again can be interpreted as a probability to have the respective class. The highest probability will win and the corresponding class is the model predicted value for a given input. The multi-class classification will be presented formally in Chapter 2.

1.4 ORGANIZATION OF THE THESIS

The rest of this thesis is organized as follows. In Chapter 2 we give a history of Artificial Neural Networks (ANN), their architecture, and how ANNs solve multi-class classification problems. We formally present the solution of the character image recognition problem in Chapter 3. The implementation and numerical results are presented in Chapter 4 and we conclude in Chapter 5.

CHAPTER 2

ARTIFICIAL NEURAL NETWORKS

2.1 HISTORY

Humankind has always been intrigued by the biological functions of the human brain. Can a machine work like the human brain and make decisions all by itself? With the development of Artificial Neural Network scientists have made the impossible a reality. The beginning of this idea started in the year 1943 when Neurophysiologist Warren McCulloch and mathematician Walter Pitts modeled a simple neural network using electrical circuits to understand the functions of neurons in the brain. In 1949, Donald Hebb wrote "The Organization of Behavior," a work which pointed out the fact that each use of the neural network pathway strengthens the network, a concept fundamentally essential and compatible to the way the human brain learns. Thus the strength of the connection between the nodes is increased when two nerves fire simultaneously. Therefore the process becomes faster and computationally more swift. However, due to the computational limitations, it was not possible to simulate the idea of a neural network until 1950. Still, the first attempt made by Nathanial Rochester from the IBM research laboratories failed. Then there came ADALINE and MADALINE in 1959. Bernard Widrow and Marcian Hoff of Stanford developed the models. The acronyms MADALINE mean Multiple ADAptive LINear Elements and ADALINE Meaning ADAptive LINear Elements. The idea of developing ADALINE was to recognize binary patterns so that if it was reading streaming bits from a phone line, it could predict the next bit. MADALINE was the first neural network applied to a real-world problem, using an adaptive filter that eliminates echoes on phone lines. While the system is as old as air traffic control systems, like air traffic control systems, it is still in commercial use. In 1962, Bernard Widrow and Marcian Hof developed a learning procedure. The goal of the learning procedure was to examine the value before the weight adjusts it. The adjust-

ment was made following the rule: Change of weight = (line value before weight) * (Error / (Total number of inputs)). It depends on the possibility that while one dynamic perceptron(linear classifier) may have a major mistake, one can alter the weight to convey it over the system, or if nothing else to adjoining perceptron. Applying this lead can still outcome in a mistake if the line before the weight is zero, despite the fact that this will in the end redress itself. The error is conserved so that all of it is distributed to all of the weights until the error is eliminated. However, with the development of the von Neumann model and Princeton architecture, the research on Neural network was left behind even though John von Neumann himself suggested the imitation of the neural functions by using telegraph relays or vacuum tubes. In the same period, a paper was written suggesting that there could not be an extension of the single-layered neural network to a multiple layered neural networks. Moreover, numerous individuals in the field were utilizing a learning capacity that was in a general sense imperfect since it was not differentiable over the whole line. Therefore, research and financing on the field went down on a big scale. This was combined with the way that the early accomplishments of some neural systems prompted a distortion of the capability of neural systems, particularly thinking about the handy innovation at the time. Guarantees went unfulfilled, and now and again more prominent philosophical inquiries prompted fear. Authors contemplated the impact that the supposed ”Thinking machines” would have on people, thoughts which are still around today.

The possibility of a PC which programs itself is exceptionally engaging. On the off chance that Microsoft’s Windows 2000 could reinvent itself, it may have the capacity to repair the great many bugs that the programming staff made. Such thoughts were engaging yet exceptionally hard to execute. Furthermore, The von Neumann model was picking up in fame. There were a few advances in the field, but for the most part, the research was few and far between.

In 1972, Kohonen and Anderson both independently developed a similar network.

They both were using matrix mathematics to describe their ideas but unaware of the fact that they were in fact, creating an array of analog ADALINE circuits. The neurons are supposed to activate a set of outputs instead of just one. In 1975, the first multi-layered network was developed as an Unsupervised network. The interest in the field of artificial neural network got a new vibe in the year 1982. John Hopfield of Caltech presented a paper to the National Academy of Sciences. His approach was to make more valuable machines by utilizing bi-directional lines, Whereas earlier, the associations between neurons were only one way. That same year, Reilly and Cooper utilized a "Hybrid Network" with various layers, each layer utilizing an alternate problem-solving technique. Additionally, in 1982, there was a joint US-Japan meeting on Cooperative/Competitive Neural Networks. In the conference, Japan announced a new Fifth Generation effort on neural networks whereas US papers generated worries. The US was concerned that they might be left behind in the field. Fifth generation computing involves artificial intelligence. The first generation used switches and wires, the second generation used the transistor, the third generation utilized solid-state technology like integrated circuits and advanced level programming languages, and the fourth generation was code generators. Thus, there was more funding, and thus more research in the field was conducted. In 1986, with multiple layered neural networks in the news, the issue was the manner by which to stretch out the Widrow-Hoff learning procedure(examining the value before adjusting the weights) to numerous layers. Three autonomous gatherings of specialists, one of whom included David Rumelhart, formerly with Stanford's psychology department, thought in similar ways about what is currently called backpropagation networks. It was given this name, because it circulated pattern recognition errors all through the network. Hybrid Neural Networks utilized only two layers; these back-propagation networks use many. The outcome is that back-propagation networks are "slow learners," therefore, requiring conceivably a large number of repetitions to learn. Presently, Neural Networks are utilized as a part of a few applications. The

principal thought behind the idea of Neural Networks is that if it works in nature, it must have the capacity to work in computers. The fate of neural systems, however, lies in the advancement of equipment. Much like the propelled chess-playing machines like Deep Blue, quick, effective neural Networks rely upon the equipment being indicated for its eventual use.

Research that focuses on creating Neural Networks is comparatively less. Because of the restrictions of processors, neural systems take a long time to learn. A few organizations are endeavoring to make what is known as a ”silicon compiler” to produce a particular sort of Integrated circuit that is upgraded for the use of Neural Networks. Digital, analog, and optical chips are the distinctive sorts of chips being created. One may promptly dismiss analog signals as a relic of times gone by. Nonetheless, in fact, neurons in the brain work more like analog signals than advanced digital signals. While advanced digital signals have two specific states (1 or 0, on or off), analog signals change amongst minimum and maximum values. It should not take long to see optical chips are utilized as a part of business applications.

2.2 ARCHITECTURE

Real and Artificial Neural Network: Before we go ahead, it’s also worth noticing that, To be fair, neural networks produced in this way are actually called artificial neural networks (or ANNs) to signify their difference from real neural networks (collections of interconnected brain cells)in our brains. The neural networks are also referred to by names like connectionist machines (the field is also called connectionism), parallel distributed processors (PDP), thinking machines, and so on.

Neural Network: The idea of neural networks came from the way the human brain works. In the brain there are nerve cells consist of dendrites , cell body, synapses and axons. The dendrites collect the information and pass it through the synapses to cell body

to process. The cell body processes the information and pass it to axons. The axons then connect to another nerve cell through dendrites. The artificial neural network essentially follows the same process.

The fundamental thought behind a neural network is to replicate lots of densely interconnected nerve cells in the brain inside a computer so that it learns to map new inputs or experiences, recognize intrinsic patterns, and make decisions on its own in a humanlike way. The most exceptional fact about the neural network is that it learns by itself, no need to program it explicitly to learn. It is the very way a human brain learns. Accomplishing this goal has been the most significant and far-reaching outcome of Neural Networks. Many scientists have been working relentlessly to achieve this automation. With the development of advanced computational methods, the goal of the artificial neural network has become more fathomable.

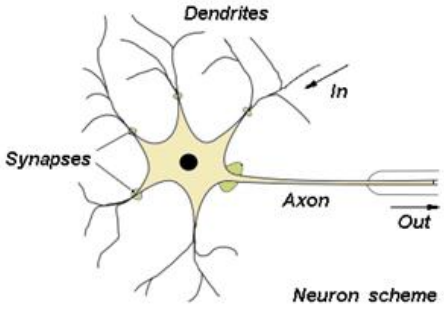


Figure 2.1: Human nerve cell

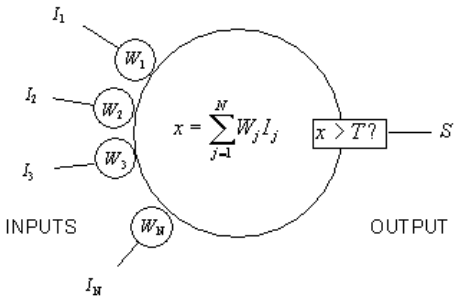


Figure 2.2: Neural Network

Here the dendrites can be compared to input units, cell body to nodes, synapses to activation function and axons to the output units.

But it is not a brain. It is imperative to understand that neural networks are actually software simulations of the brains nerve cells. They are created by programming ordinary computers, working in a very typical manner with their ordinary transistors and series of connected logic gates, to behave as though they are created from billions of interconnected

brain cells working in parallel setup. No-one has yet attempted to build a computer by wiring up transistors in a densely parallel structure exactly like the human brain. The neural networks differ from human brains in exactly the very same way that a computer model for the weather differs from the real clouds, sunshine or snowflakes. Computer simulations are just aggregations of algebraic variables and mathematical equations by connecting them in a way such as numbers stored in a machine whose values are constantly changing. They mean nothing significant to the computers they run in but are significant to the people who program them.

Elements of the Neural network: A typical neural network consists of a few dozen to even millions of artificial neurons, named units, arranged in a series of hidden layers. These hidden layers can be connected to other layers on both sides, which is why the names input layer and the output layer are specified. Whereas input layers are designed to receive information of various forms from the outside world, the output layer will predict from the learning previously achieved. The learning in the input layers consists of recognizing the patterns and other attributes of the data set fed to it. Between the input layer and the output layer there are one or more hidden layers. The hidden layers are the ones that form the most part of the artificial neural networks. Basically, all the layers in a neural network are interconnected to one another. Thus each input in the input layer has an impact on the output it produces. The connection between the layers, from one input to each node in the hidden layers are associated with a weight. These weights can be positive or negative depending on the nature of the weight. The more the weight the more impact it has on the output. Apart from input layer , hidden layers and output layer, there is a bias associated with each layer to adjust the error effect on each layer. At each node of an artificial neural network there is an activation function through which the information passes after being processed. The activation function in each node() makes decision whether to pass the information or not depending on the intensity of the processed information. It also puts bound to the outputs.

The reason why we use sigmoid function (more generally logistic function) as an activation function is that it exists between (0 to 1). Since the probability of anything exists only between the range of 0 and 1(the bound), sigmoid is the right choice. Learning: Information goes through a neural network in one of the two ways. The learning phase is the time of training and operating normally after training. Patterns of information are put into the network through the units in the input layer. The information is then conveyed to the units in the hidden layers, and after that the processed information arrives at the units in the output layer. This typical design is called a feedforward network. However, not all units ”fire” at the time. Each unit on the right receives input from the units on its left. Then the input is multiplied by the weights at each connection between the units they travel through. Every unit adds up all the inputs it receives in this way, and if the sum of the product of weight and the input plus the bias is more than a certain preset threshold value, the unit ”fires” and triggers the units it is connected to on its right. The learning of a neural network requires an element of feedback. The bigger the difference between the outcome we get and the outcome we supposed to get, the bigger the changes we make. The neural network learns in the same way; typically this feedback process is called the backpropagation algorithm. It involves comparing the output we get with the predefined output we are supposed to get. The correction is made by adjusting the weights between the nodes, working backward from the output nodes through the hidden nodes to the input nodes. With enough training, the network learns and corrects the outcome toward the intended outcome by reducing the differences. Thus the neural network figures out what the actual outcome should be. For example, Once the neural network has been trained with enough training inputs for the desired outcome, it reaches a level of understanding where we can feed it an entirely new set of input dataset that it never experienced before and see how it responds. Let us say a neural network has been trained with lots of pictures of chairs and tables, prepared in some appropriate manner that it can understand and telling the network whether each image is a

chair or a table. Now, let us feed it with 25 different chairs and 25 different tables of new designs that it never had before and see what happens. Based on how we have trained it, it will attempt to classify the new examples as either chairs or a table, generalizing based on its past experiences, just like a human brain.

That does not mean to say that a neural network can just take a piece of furniture and can instantly respond to it in a meaningful way; it's not a human brain. Referring to the example above: the network is not actually looking at the piece of furniture. The neural network inputs are essentially binary numbers: each input unit is either yes or no. So if we have five input units, we could feed in information about five different attributes of different chairs using binary answers. The questions about the attributes could be 1) Does it have a top? 2) Does it have a back? 3) Can we sit on it comfortably for long periods of time? 4) Does it have soft upholstery? 5) Can we put lots of things on top of it? Typically, a chair would then present answers as Yes, No, Yes, Yes, No or 10110 in binary outcomes, while a table might present answers as No, Yes, No, No, Yes or 01001 in binary numbers. So, throughout the learning phase, the network is basically looking at lots of numbers like 10110 and 01001 and so on and learning that some of them mean chairs which is likely to be an output of 1 while others mean tables with an output of 0.

Uses: In our experiments, we have used the neural networks to classify images of old English manuscripts. The letters have been cropped directly from the manuscript images. Then we took out the colors from those images to make them black and white. We used an algorithm to fit the images in a specific frame. Then created an image matrix based on the presence and absence of ink, i.e. presence of ink means 1 and absence of ink means 0. Then vectorized the matrix and fed it and train the neural network to recognize them. However, lots of different applications for neural networks involve recognizing patterns and making simple decisions about the inputs. In airplanes, we might use a neural network as a basic autopilot, with input nodes reading signals from the various cockpit instruments

and output nodes modifying the plane's controls accordingly to keep it safely on course. In a factory, we can use a neural network for quality control. Let us say we are producing clothes washing detergent in some big, convoluted chemical process. We could measure the final detergent in various ways, i.e., by its color, acidity, thickness etc. Then feed those measurements into our neural network as inputs, and have the network decide whether to accept or reject that particular batch.

Many of our everyday work involves recognizing patterns and making decisions based on them, So, neural networks can help us out in lots of different ways. They can help us forecasting the stock market or the weather, operating radar scanning systems that can automatically identify enemy aircraft or ships, and even help doctors to figure out complex diseases based on their symptoms. There might be neural networks ticking away inside our computer or cell phone right this instant. If we are using cell phone apps that recognize our handwriting on a touch screen, it might be applying a simple neural network to find out which characters we are writing by looking out for distinct features in the marks we make with our fingers and the order in which we make them. Some voice recognition software also use neural networks. And so do some of the email programs that automatically differentiate between real emails and spams. Neural networks have been proving highly effective in translating text from one language to another. For example, Google's automatic translation makes use of neural network a lot. In 2016, Google announced that they were using something called Neural Machine Translation (NMT) to convert entire sentence instantly, with a 5585 percent reduction in errors. All in all, neural networks have made computer systems vastly useful by making them more like a human brain.

2.3 SOFTWARE

Neuralnet:

Neuralnet version 1.33 is a very flexible package in R created to train multi-layer perceptron

using the concept of regression analysis. Thus, backpropagation and resilient backpropagation algorithms are used with the flexibility of shifting to one from the other. It provides the freedom of choosing activation function, error function as well as number of covariates, response variables and hidden layers.

Neuralnet relies on packages, Grid, MASS, grDevices, stats, utils and doesnt need compilation. It has numerous arguments such as formula, data, hidden, threshold, rep, algorithm, err.fct, likelihood, etc. Authored by Stefan Fritsch, Frauke Guenther, Marc Suling, Sebastian M. Mueller and Maintained by Frauke Guenther.

RStudio :

Rstudio is a collection of integrated tools designed to make R more productive. It consists of a console, an editor where codes can be highlighted executed directly, numerous tools for various plotting, smart indention, integrated R help, comprehensive workspace, interactive debuggers, authoring with Sweave and R Markdown, etc.

RStudio version 1.0.44, compatible for Bits x32/x86; x64. Its stakeholders are Students; Researchers. Published by RStudio, OS Supported by All Current Windows, Freely Available to All.

2.4 Multiclass classification with Neural Network

An Artificial Neural Network with multiple outputs is represented in Figure 2.3 and it is a natural extension of the single output network in Figure 1.4. In general, one can see an ANN as a succession of layers with multiple neurons, where the first layer is the input and the last layer is the output of the network. The data “propagates” through the network from left to right (forward propagation, from input to output) and at each layer a non-linear transformation is performed. We will give next careful definition of the ANN model with multiple outputs and explain how this model performs multi-class classification.

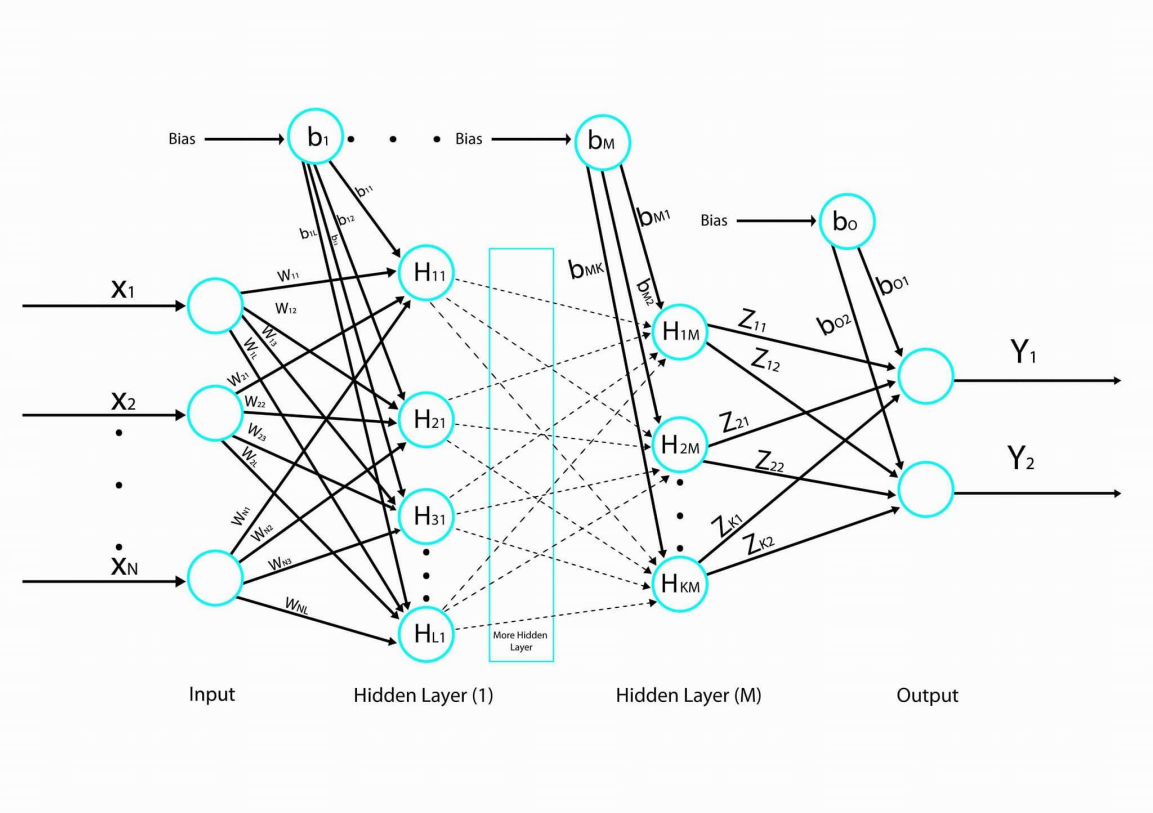


Figure 2.3: Artificial Neural Network with multiple outputs

Definition 4. [ANN forward step nonlinear transformation] An Artificial Neural Network forward step nonlinear transformation, from a hidden layer i with H_i neurons (can be the input layer) to a hidden layer j with H_j neurons (can be the output layer), is a mapping:

$$h_{ij} : \mathbb{R}^{H_i} \to \mathbb{R}^{H_j}$$

$$h_{ij}(\mathbf{x}) = \sigma_{H_i}\left(W_{ij}^T \cdot \overline{\mathbf{x}}\right)$$

where:
$\overline{\mathbf{x}} = \begin{bmatrix} 1 \\ \mathbf{x} \end{bmatrix}$, $\mathbf{x} \in \mathbb{R}^{H_i}$ is the augmented input variable,
$\sigma_{H_i} : \mathbb{R}^{H_i} \to \mathbb{R}^{H_i},\ \sigma_{H_i}(\mathbf{x}) = (\sigma(x_1), \ldots, \sigma(x_{H_i}))$,
$\sigma : \mathbb{R} \to \mathbb{R},\ \sigma(x) = \dfrac{1}{1+e^{-x}}$ is the sigmoid function, and
$W_{ij} \in \mathbb{R}_{(H_i+1)\times H_j},\ W_{ij} = \begin{bmatrix} b_j & b_j & \cdots & b_j \\ w_{11} & w_{12} & \cdots & w_{1H_j} \\ \cdots & \cdots & \cdots & \cdots \\ w_{H_i1} & w_{H_i2} & \cdots & w_{H_iH_j} \end{bmatrix}$ is the list of ANN parameters from layer i to layer j (including the bias into layer j).

The ANN model with N_h hidden layers and multiple outputs is a composition of successive non-linear transformations, as follows.

Definition 5. [ANN general model] The ANN general prediction model with N_h hidden layers and m outputs is a mapping $N : \mathbb{R}^n \to (0,1)^m$:

$$\mathbb{R}^n \xrightarrow{h_{0H_1}} \mathbb{R}^{H_1} \xrightarrow{h_{H_1H_2}} \cdots \xrightarrow{h_{H_{N_h}H_{N_h+1}}} (0,1)^m$$

where $N_h \geq 1$ is the number of hidden layers, H_l, $l = 1 \cdots N_h$, are the number of neurons per each hidden layer, and $h_{ft} : \mathbb{R}^{H_f} \to \mathbb{R}^{H_t}$ is the ANN forward step nonlinear transformation from the hidden layer f to the hidden layer t (where $f = 0$ means the input layer and $t = N_h + 1$ means the output layer). The h_{ft} transformations are created by solving an optimization problem as in (1.3).

Given a number of classes C and an ANN model $N = (Y_1, Y_2, \ldots, Y_C)$ as in Definition 5 (that is, with C outputs), we define the ANN multi-class prediction model as:

Definition 6. [Multi-class classification using ANN]

$$\begin{aligned} &\widehat{F_C} : \mathbb{R}^n \to \{1, 2, \ldots, C\} \\ &\widehat{F_C}(\mathbf{x}) = \operatorname{argmax}_i(Y_i), \;\; i = 1, \ldots, C \end{aligned} \tag{2.1}$$

In other words, the prediction of class $1, \ldots, C$ is given by the largest output Y_i of the ANN model.

With these formal definitions we can introduce now a practical way of measuring the accuracy of the model.

Definition 7. [Accuracy of an ANN model N] The accuracy of an ANN model N for a given test set $\mathcal{T} \subset \mathcal{I}$ is the number of times the ANN model N agrees with the corresponding ideal model M over the total number of samples in $\mathcal{T}$.

In Chapter 4 we present numerical values of the model accuracy in various contexts. We used a 10-fold cross validation to sample $\mathcal{I}$ and selected the training set for computing the parameters of each model, and the test set for testing computing the accuracy of the model.

In the next chapter we will explain how to apply the ANN multi-class classification model to the problem of character image recognition.

CHAPTER 3

CHARACTER RECOGNITION

3.1 INTRODUCTION

Early character recognition can be traced back to 1914 when Emanuel Goldberg developed a machine that read characters and converted them into standard telegraph code [14]. In the late 1920s into 1930s Goldberg developed what he called a "Statistical Machine" for searching microfilm archives using an optical code recognition system.

Optical Character Recognition (OCR) is nowadays widely used for various tasks: helping visually impaired persons; automatically recognizing number plates; automatically entering data; converting handwriting (on a computer touch screen) into typed text; making images of printed documents searchable, etc. There are many off-the-shelf tools (such as Adobe Acrobat) that can perform excellent OCR in images. However, while OCR on images of printed materials can be performed with high accuracy, recognizing handwritten characters poses a much bigger challenge. One of the most famous collections of handwritten digits is the MNIST database (Modified National Institute of Standards and Technology database). MNIST is a large database of handwritten digits that is commonly used for training various image processing systems. Each digit image in the database was normalized to fit into a 28x28 pixel bounding box and there are 60,000 training images and 10,000 testing images. Using complex machine learning techniques (multi-layer neural networks) one can produce high accuracy character recognition for these images (around 98% accuracy).

However, when dealing with little training information all these sophisticated models fail to produce satisfactory results. For instance, while Adobe Acrobat performs a very accurate OCR on printed materials as in Figure 3.1 (left), the results on handwritten materials (Figure 3.1, right) are less satisfactory. Our main goal is to find an easy and a practical way to perform character recognition on the images taken from the manuscripts using a small

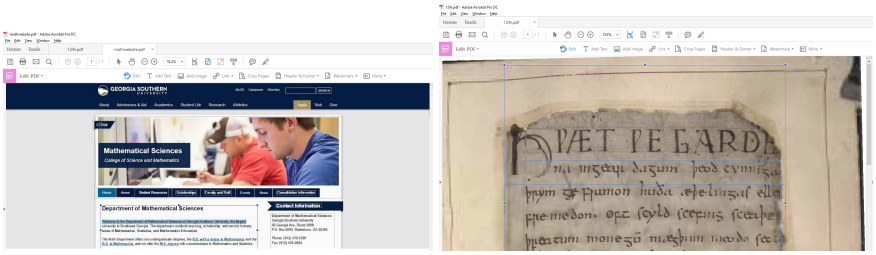


Figure 3.1: OCR with Adobe Acrobat on printed material (left) vs. handwritten characters (right)

size training set. Such enterprise would produce numerous practical benefits: easy information access, availability of rare and unique cultural materials to more than one billion people, help preserving rare materials, search support directly in manuscript images, etc.

3.2 Character Image Data and the Image Recognition Model

For our study we considered the images of manuscripts of Beowulf, the famous Old English epic poem, which is currently held by the British Library. Our data set consists of character images extracted from the Electronic Beowulf [5] manuscript images. Figure 3.2 top shows an original manuscript page. We converted these manuscript images in black and white (B&W) format (Figure 3.2 bottom) before extracting character images (only for lower-case characters), then individual character images where extracted as in Figure 3.3.

Let us first establish some notations. We denote by $\mathcal{I}$ the set of all lower-case character images in the manuscript (Table 3.1 is an excerpt of this set). Then the set of all 22 distinct letters we extracted is

$$\mathcal{L} = \{a, ae, b, c, d, e, eth, f, g, h, i, l, m, n, o, p, r, s, t, thorn, u, w\}$$

where ae, eth, and $thorn$ denote the Old English letters , , and , respectively.

Our goal is to create a model capable of taking a character image as an input and produce the corresponding letter as the output. More formally, the model is defined as

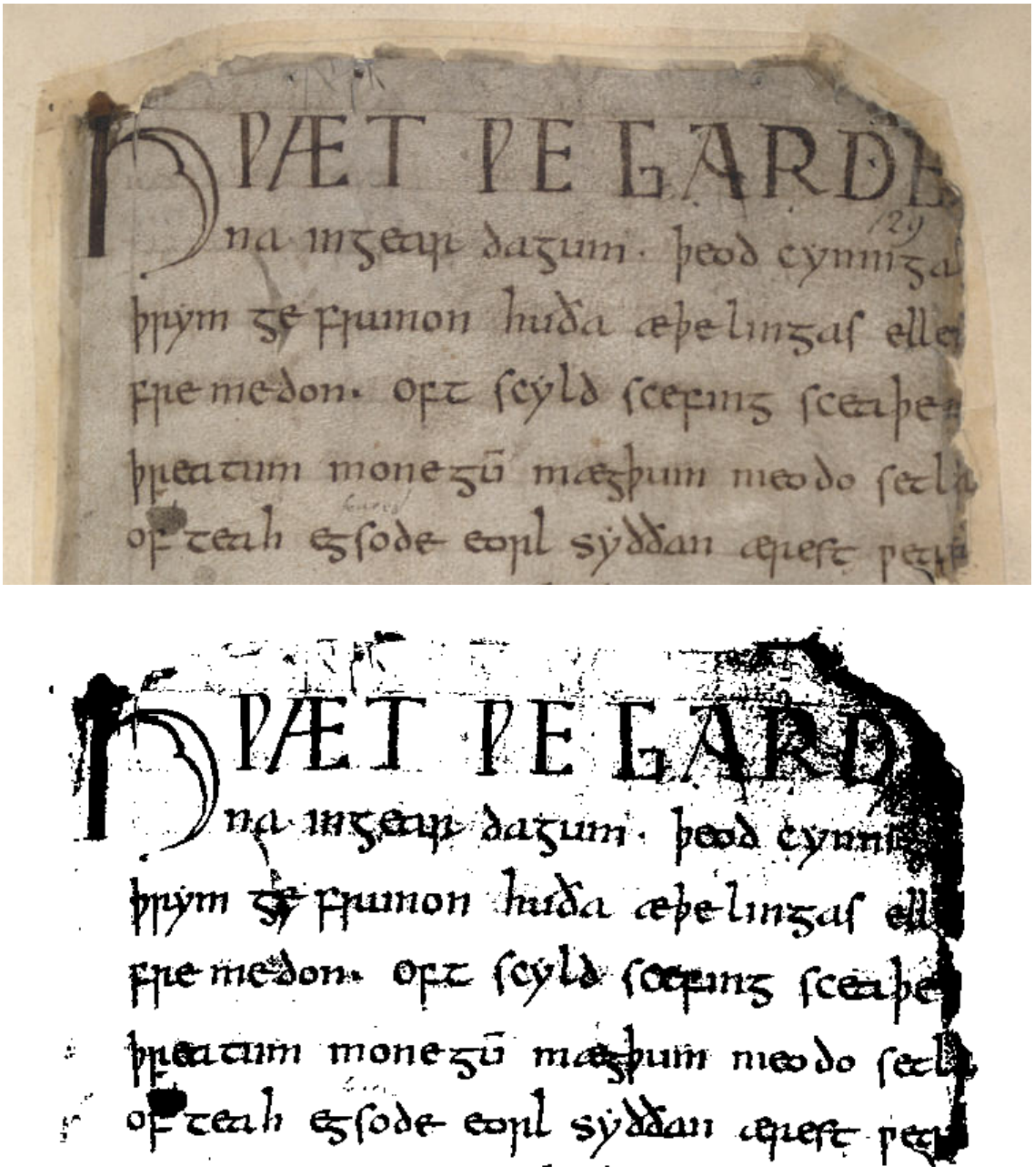


Figure 3.2: Manuscript image: original (top) and converted to B&W (bottom)

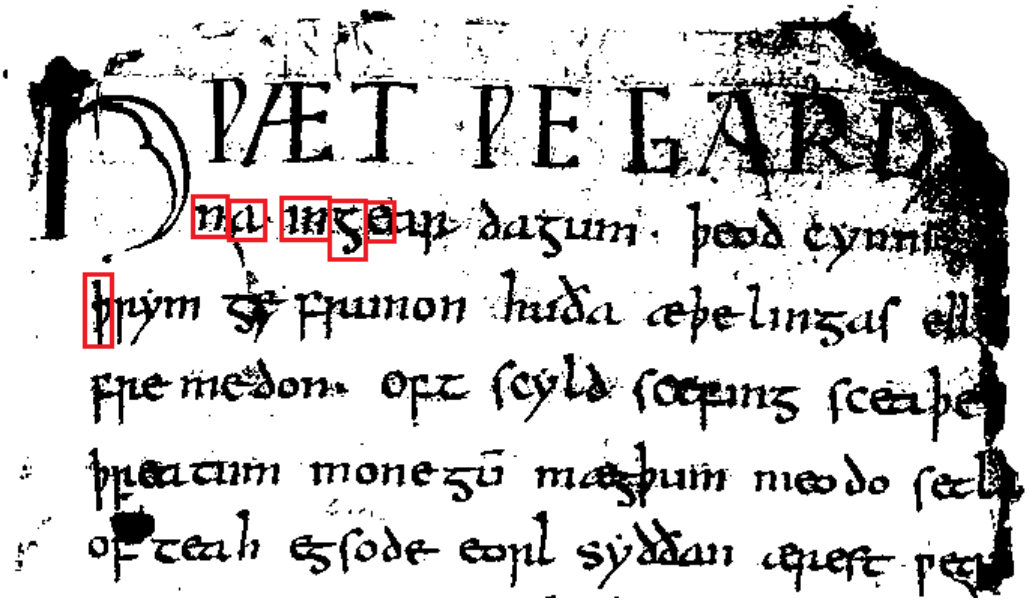


Figure 3.3: Extracting character images from a B&W manuscript image

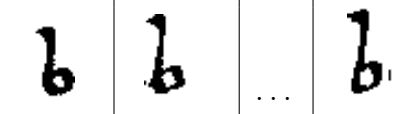
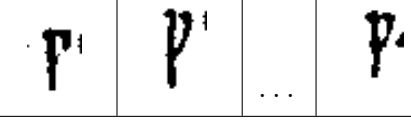
Char	Character Images			
a			...	
b			...	
...	...	...	...	...
w			...	

Table 3.1: The set $\mathcal{I}$ of all lower case character images in manuscript

follows.

Definition 8. [The character recognition model] The character image recognition model M is a mapping:

$$M : \mathcal{I} \rightarrow \mathcal{L} \tag{3.1}$$

that takes as input a character image in $\mathcal{I}$ and produces as a result the letter in $\mathcal{L}$ corresponding to the character image.

Example 3.1.

$M\left(\right) = a$

$M\left(\right) = ae$

$M\left(\right) = eth$

While the model defintion is very clear and straightforward, a practical model for performing the recognition may be daunting. The current state of the art image recognition is quite advanced and certainly capable of recognizing printed characters with excellent precision. However, our character images can be considred quite noisy (missing small parts, tiny extra ink spots, etc.) for setting accuracy hopes too high. As we already mentioned, off-the-shelf software did not perform satisfactory on our data.

In the next section we present an ANN based model for character image recognition that turns to be effective in practice.

3.3 Character Image Classification using Artificial Neural Networks

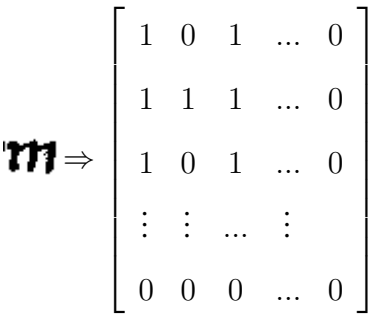
A first obvious obstacle in using an ANN (as modeled by Definition 5) to implement the model in Definition 8 is a total mismatch between the inputs of the two models: the former

takes as input a vector $\mathbf{x} \in \mathbb{R}^n$ whereas the latter takes as input an image. We overcome this obstacle by first constructing a mapping:

$$mat : \mathcal{I} \to \mathbb{R}_{p\times q} \tag{3.2}$$

where $\mathbb{R}_{p\times q}$ is the set of matrices of dimensions $p \times q$, and p, q are the maximum height and width, respectively, of images in $\mathcal{I}$. This converts every image in $\mathcal{I}$ to a matrix of same size for all images in $\mathcal{I}$.

Example 3.2. *Converting a black and white image into the corresponding matrix is typically straightforward. The matrix with entries 0 and 1 is created of the same size as the image, then each white pixel is converted to 0, and each black pixel to 1. For instance, the transformation of a character image for "m" is:*



$$\Rightarrow \begin{bmatrix} 1 & 0 & 1 & \dots & 0 \\ 1 & 1 & 1 & \dots & 0 \\ 1 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \dots & \vdots & \\ 0 & 0 & 0 & \dots & 0 \end{bmatrix}$$

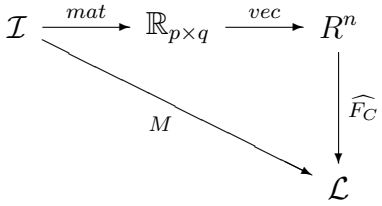
The transformation (3.2) is still not enough for our purpose of converting an image to a vector. We construct next a mapping:

$$vec : \mathbb{R}_{p\times q} \to \mathbb{R}^n \tag{3.3}$$

which vectorizes matrices in $\mathbb{R}_{p\times q}$. We performed matrix vectorization using different methods, described in detail in Chapter 4. Based on the experimental results we then selected the vectorization method that produced the best results, measured using the model accuracy.

Finally, our character image classification model using an ANN can be summarized by the following commutative diagram:

Definition 9. The character image classification model M in (3.1) is represented by the commutative diagram:



In the diagram, without loss of generality, we consider that the set of letters $\mathcal{L}$ is indexed and that the index $1, \ldots, C$ produced by $\widehat{F_C}$ from (2.1) is directly converted into the corresponding letter in $\mathcal{L}$.

We proceed next to explain the implementation of our model, describe the experiments, and the numerical results.

CHAPTER 4

IMPLEMENTATION AND NUMERICAL RESULTS

We describe next the experimental results of multi-class classification for Old English characters retrieved from manuscript images. Our experiments were performed in the following three directions.

1. Experiment 1: we performed multi-class classification of all letter images using direct vectorization by columns of the image matrix.

2. Experiment 2: we performed and compared multi-class classification of all letter images using various vectorization techniques.

3. Experiment 3: we performed multi-class classification by groups of letters, using the best classification method identified in Experiment 2.

The experiments and their results are discussed in the following sections. We start by giving the description of the character image pre-processing, which is essentially an image normalization and vectorization process.

4.1 PRE-PROCESSING CHARACTER IMAGES

4.1.1 IMAGE NORMALIZATION

Performing image recognition on character images extracted directly from manuscript images raised a natural challenge: how to create a model for input data of various dimensionalities? Clearly, some kind of normalization procedure must map inputs of various dimensions into same dimension output, then use these outputs to create a model and subsequently perform classification. The process of normalization we applied to all images in $\mathcal{I}$ (which essentially implements the mapping (3.2), $mat : \mathcal{I} \to \mathbb{R}_{p\times q}$) is described in Algorithm 1. The algorithm essentially shifts the character image to the top-left corner

Algorithm 1 Image normalization algorithm

procedure MAT($img \in \mathcal{I}$) ▷ computes image matrix
Input: $img \in \mathcal{I}$
Output: $M \in \mathbb{R}_{p \times q}$
    $M \leftarrow img$ pixel values
    while first column of M only zeros **do**
        remove first column of M
    end while
    while first row of M only zeros **do**
        remove first row of M
    end while
    append rows with zeros up to p rows
    append columns with zeros up to q columns
    return M ▷ Returns the normalized image matrix
end procedure

(lines 5–10), then pads with zeros in order to bring the image matrix size to the uniform dimension $p \times q$. This dimension is pre-computed as the largest dimension each character image fits into.

4.1.2 MATRIX VECTORIZATION

The image normalization process described in the previous section converts the image of any character into a matrix in the space $\mathbb{R}_{p\times q}$ of matrices of same dimensions $p \times q$. We will explain now different matrix vectorization methods we used in our experiments. Let us consider a matrix $A \in \mathbb{R}_{p\times q}$, $A = [a_{ij}]$. Each vectorization method aims to convert such a matrix A into a vector. The dimension of the resulting vector varies from method to method.

Vectorization by columns

This method converts matrix A into the vector obtained by stacking the columns of A.

Vectorization by rows

This method converts matrix A into the vector obtained by stacking the rows of A.

Using the correlation matrix vectorized by rows/columns

The correlation matrix of $A = [a_{ij}] = [A_1\ A_2\ \ldots\ A_q]$ is computed as follows:

- sample mean:
$$M = \frac{1}{q}(A_1 + A_2 + \cdots + A_q)$$
- mean deviation form:
$$B = [A_1 - M\ A_2 - M\ \ldots\ A_q - M]$$

- covariance matrix:

$$S = \frac{1}{q-1} BB^T$$

- correlation matrix:

$$C = corr(A) = (diag(S))^{-\frac{1}{2}} \, S \, (diag(S))^{-\frac{1}{2}}$$

Then the correlation matrix C was vectorized by columns, as described above.

Using singular value decomposition, then a number k of left-singular vectors

We perform singular value decomposition of matrix A:

$$A = U \cdot \Sigma \cdot V^T$$

where U, V are orthogonal matrices and Σ is a diagonal matrix of non-negative values (singular values). We then select k columns form matrix U (left-singular vectors) and create the vectorization of A.

4.2 Experiment 1: Classification results using image vectorization by columns

For this experiment we created a single ANN model for multi-class classification of all letters in $\mathcal{L}$. We used vectorization by columns for character images. The confusion matrix for the classification results is presented in Table 4.1 and the accuracy of classification was 61.36%. We found this classification accuracy rather disappointing and proceeded to improve it.

Table 4.1: The confusion table for results in Experiment 1 (Accuracy: 0.6136363636)

		Predicted																					
		a	ae	b	c	d	e	et	f	g	h	i	l	m	n	o	p	r	s	t	th	u	w
Original	a	0	0	0	0	1	0	0	0	0	0	0	0	0	1	1	0	0	1	0	0	0	0
	ae	0	2	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
	b	0	0	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	c	0	0	0	3	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
	d	0	0	0	0	2	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0
	e	0	0	0	0	0	3	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
	et	0	0	0	0	0	0	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	f	0	0	0	0	0	0	0	2	0	0	1	0	0	0	0	0	0	0	0	0	0	1
	g	0	0	0	1	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
	h	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	1	0	1	0	0	0	0
	i	0	0	0	0	0	0	0	0	0	0	4	0	0	0	0	0	0	0	0	0	0	0
	l	0	1	0	0	0	0	0	0	0	1	0	2	0	0	0	0	0	0	0	0	0	0
	m	0	0	0	0	0	0	0	0	0	0	0	0	4	0	0	0	0	0	0	0	0	0
	n	1	1	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0
	o	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0
	p	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	2
	r	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	1	0
	s	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	0	0	0	0
	t	1	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	1	0	0	0
	th	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	0	0
	u	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	0
	w	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3

4.3 Experiment 2: Comparison between classification using multiple image vectorization methods

Similar as in Experiment 1, we created a single ANN model for multi-class classification of all characters in $\mathcal{L}$. However, this time we experimented with different matrix vectorization methods. The best experimental result we obtained was for the singular value decomposition with one left-singular vector. The confusion matrix is reported in Table 4.2 and the accuracy of classification increased to 65.91%. While the accuracy was not impressive, the fact that the best accuracy was obtained using a "signature" of one left-singular vector for each image was impressive: the model input dimensionality was dramatically decreased from about $p \times q = 46 \times 46$ to just 46.

Yet, it was clear that a model that classifies all letters at once can hardly be improved, given the fact that our training data set was very small. In the next experiment we describe a clever way to perform multi-class classification by using a few smaller models for groups of letters rather than one big model for all letters.

4.4 Experiment 3: Hierarchical classification using columns correlation of the image matrix

The main idea of this experiment relies on the fact that smaller models may perform better due to the reduced size of the training data. That is, a classification model for a group of letters such as $\{b, c, e, u\}$ will likely fare better in terms of accuracy than a model for the whole set of letters $\mathcal{L}$. We proceeded to trials and identified 5 models corresponding to 5 groups of letters for which the accuracy was far better than the best accuracy we obtained in Experiment 2. However, none of these smaller models can classify the whole set of letters, that is, none solve our classification problem completely. We call this process incremental learning for creating the models (Algorithm 2) and hierarchical multi-class classification

Table 4.2: The confusion table for results in Experiment 2 (Accuracy: 0.6590909091)

		Predicted																					
		a	ae	b	c	d	e	et	f	g	h	i	l	m	n	o	p	r	s	t	th	u	w
Original	a	2	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	ae	0	1	0	0	0	1	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0
	b	0	0	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	c	0	0	0	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	d	2	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	e	0	0	0	0	0	3	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
	et	0	0	0	0	2	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
	f	0	0	0	0	0	0	0	3	0	0	0	0	0	1	0	0	0	0	0	0	0	0
	g	0	1	0	0	0	0	0	1	2	0	0	0	0	0	0	0	0	0	0	0	0	0
	h	0	0	2	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0
	i	0	0	0	0	0	0	0	0	0	0	4	0	0	0	0	0	0	0	0	0	0	0
	l	0	0	0	0	0	0	0	0	0	0	0	4	0	0	0	0	0	0	0	0	0	0
	m	0	0	0	0	0	0	0	0	0	0	0	0	4	0	0	0	0	0	0	0	0	0
	n	0	0	0	0	0	0	0	0	0	0	1	0	2	1	0	0	0	0	0	0	0	0
	o	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	0	0	0	0	0	0	0
	p	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	1	1	0	0	0	0	0
	r	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	2	1
	s	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	0	0	0	0
	t	0	0	0	0	0	0	0	0	0	0	1	0	1	0	1	0	0	0	1	0	0	0
	th	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	0	0
	u	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	0
	w	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	3

when using these models to perform character recognition (Algorithm 3).

Algorithm 2 Incremental learning algorithm

1: **procedure** INCREMENTLEARN($\mathcal{I}$) ▷ computes models for groups of letters in $\mathcal{I}$
2: **Input**: $\mathcal{I}$
3: **Output**: $(N_i, \mathcal{L}_i)$
4: $\mathcal{L} \leftarrow$ list of all letters
5: $i \leftarrow 1$ ▷ Group counter
6: **while** letters in $\mathcal{L}$ **do**
7: create best model for all letters in $\mathcal{L}$
8: $\mathcal{L}_i \leftarrow subset\ of\ \mathcal{L}$ ▷ Select a group of 4-6 best classified letters
9: remove $\mathcal{L}_i$ from $\mathcal{L}$
10: $N_i \leftarrow$ model for $\mathcal{L}_i$
11: $i \leftarrow i + 1$ ▷ Increment group counter
12: **end while**
13: **return** $(N_i, \mathcal{L}_i)$ ▷ Returns all groups of letters and their models
14: **end procedure**

The experimental results of Algorithm 2 and the groups of letters it produces are summarized in the following:

Group 1 : b, c, e, u, zz, where zz denotes "other". We used a model with one hidden layer with 32 neurons. The accuracy was over 92% and the complete confusion matrix is shown in Figure 4.1.

Group 2 : $a, d, s, thorn, w, zz$, where zz denotes "other". We used again a model with one hidden layer with 32 neurons. The accuracy was over 92% and the complete confusion matrix is shown in Figure 4.2.

Group 3 : ae, l, m, o, p, zz, where zz denotes "other". We used a model with one hidden layer

```
[1] "Letters: b, c, e, u, zz"
[1] "Layers: 32"
[1] "Confusion Table"
        Predicted
Original  b  c  e  u zz
      b   2  0  0  0  2
      c   0  4  0  0  0
      e   0  0  1  0  3
      u   0  0  0  4  0
      zz  0  1  0  1 70
[1] "Accuracy: "
[1] 0.9204545455
```

Figure 4.1: Model predictions results for Group 1: b, c, e, u, zz (with $zz = other$)

with 32 neurons. The accuracy was over 85.2% and the complete confusion matrix is shown in Figure 4.3.

Group 4 : eth, g, h, i, zz, where zz denotes "other". We used a model with two hidden layers, with 32 and 10 neurons, respectively. The accuracy was over 88.6% and the complete confusion matrix is shown in Figure 4.4.

Group 5 : f, n, r, t, zz, where zz denotes "other". We used a model with three hidden layers, with 64, 30, and 12 neurons, respectively. The accuracy was 87.5% and the complete confusion matrix is shown in Figure 4.5.

As these results clearly show, smaller models can produce results with significantly better accuracy (compared with the models for predicting all letters, with accuracy in the low 60's).

```
[1] "Letters: a, d, s, thorn, w, zz"
[1] "Layers: 32"
[1] "Confusion Table"
        Predicted
Original  a  d  s thorn  w zz
   a      2  0  0     0  0  2
   d      1  2  0     0  0  1
   s      0  0  4     0  0  0
   thorn  0  0  0     4  0  0
   w      0  0  0     0  3  1
   zz     0  0  1     0  1 66
[1] "Accuracy: "
[1] 0.9204545455
```

Figure 4.2: Model predictions results for Group 2: $a, d, s, thorn, w, zz$ (with $zz = other$)

```
[1] "Letters: ae, l, m, o, p, zz"
[1] "Layers: 32"
[1] "Confusion Table"
          Predicted
Original ae  l  m  o  p zz
      ae  1  0  0  1  0  2
      l   0  4  0  0  0  0
      m   0  0  4  0  0  0
      o   0  0  1  2  0  1
      p   0  0  0  0  1  3
      zz  3  0  2  0  0 63
[1] "Accuracy: "
[1] 0.8522727273
```

Figure 4.3: Model predictions results for Group 3: ae, l, m, o, p, zz (with $zz = other$)

```
[1] "Letters: eth, g, h, i, zz"
[1] "Layers: 32, 10"
[1] "Confusion Table"
          Predicted
Original eth  g  h  i zz
     eth   3  0  0  0  1
     g     0  2  0  0  2
     h     0  0  2  0  2
     i     1  0  0  2  1
     zz    0  1  0  2 69
[1] "Accuracy: "
[1] 0.8863636364
```

Figure 4.4: Model predictions results for Group 4: eth, g, h, i, zz (with $zz = other$)

```
[1] "Letters: f, n, r, t, zz"
[1] "Layers: 64, 30, 12"
[1] "Confusion Table"
        Predicted
Original  f  n  r  t zz
      f   1  1  1  0  1
      n   0  2  0  0  2
      r   0  0  1  0  3
      t   0  0  0  2  2
      zz  0  0  1  0 71
[1] "Accuracy: "
[1] 0.875
```

Figure 4.5: Model predictions results for Group 5: f, n, r, t, zz (with $zz = other$)

Finally, with the 5 models and groups of letters produced as above, we proceed on experimenting hierarchial prediction using Algorithm 3. The results of Experiment 3 (hier-

Algorithm 3 Hierarchical multi-class classification algorithm

procedure HIERARCHICALMULTICLASS($img \in \mathcal{I}$) ▷ multi-class prediction
Input: $img \in \mathcal{I}, (N_i, \mathcal{L}_i)$ ▷ an image, all models and letter groups
Output: $letter$ ▷ $letter$ classification of img
    $letter \leftarrow NONE$
    for each $(N, \mathcal{L})$ in list $(N_i, \mathcal{L}_i)$ **do**
        $letter \leftarrow N(img)$ ▷ Classify img with model N
        if $letter \in \mathcal{L}$ **then**
            return $letter$ ▷ classification of img found
        end if
    end for
    $letter \leftarrow$ best classification among all N_i's
    return $letter$ ▷ Returns the classification of img
end procedure

archical multi-class classification) are shown in Table 4.3. We clearly succeeded in significantly improving the accuracy (by over 10%) of results in Experiments 1 and 2 (Tables 4.1 and 4.2).

Table 4.3: The confusion table for results in Experiment 3 (Accuracy: 0.7613636364)

		Predicted																					
		a	ae	b	c	d	e	et	f	g	h	i	l	m	n	o	p	r	s	t	th	u	w
Original	a	2	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	ae	0	2	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0
	b	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
	c	0	0	0	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	d	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
	e	0	1	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	eth	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
	f	0	0	0	0	0	0	0	2	0	0	0	0	0	1	0	0	0	0	1	0	0	0
	g	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	1	0	0	0
	h	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	1	0	0	0
	i	0	0	0	0	0	0	1	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0
	l	0	0	0	0	0	0	0	0	0	0	0	4	0	0	0	0	0	0	0	0	0	0
	m	0	0	0	0	0	0	0	0	0	0	0	0	4	0	0	0	0	0	0	0	0	0
	n	0	0	0	0	0	0	0	0	0	0	1	0	1	2	0	0	0	0	0	0	0	0
	o	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	1	0	0	0
	p	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	2	0	0	1	0	0	0
	r	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	3	0	0	0	0	0
	s	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	0	0	0	0
	t	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	3	0	0	0
	thorn	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	0	0
	u	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	0
	w	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	3

CHAPTER 5

CONCLUSION AND FUTURE WORK

We are proposing a practical, accurate, and computationally efficient method for Old English character recognition from manuscript images. Our method relies on a modern machine learning model, Artificial Neural Networks, to perform character recognition on individual character images cropped directly from manuscript pages. We propose model dimensionality reduction methods that improve accuracy and computational effectiveness. Our experimental results show that the model we propose produces satisfactory results and better prediction accuracy than off-shelf automatic text recognition software.

Humanities scholars working with manuscripts typically perform an initial manual text extraction from manuscript images, followed by adding various metadata information from images or editorial work. This work can be extended two-fold: (i) automatic text extraction from manuscript images and (ii) combining edited manuscript textual information with character images recognition (described in this work) to produce more accurate character recognition for directly searching manuscript images.

REFERENCES

[1] Thanatip Chankong, Nipon Theera-Umpon, and Sansanee Auephanwiriyakul, *Automatic cervical cell segmentation and classification in pap smears*, Computer Methods and Programs in Biomedicine **113** (2014), no. 2, 539 – 556.

[2] M. Anousouya Devi, S. Ravi, J. Vaishnavi, and S. Punitha, *"classification of cervical cancer using artificial neural networks"*, Procedia Computer Science **89** (2016), 465 – 472, Twelfth International Conference on Communication Networks, ICCN 2016, August 19 21, 2016, Bangalore, India Twelfth International Conference on Data Mining and Warehousing, ICDMW 2016, August 19-21, 2016, Bangalore, India Twelfth International Conference on Image and Signal Processing, ICISP 2016, August 19-21, 2016, Bangalore, India.

[3] Kelwin Fernandes, Jaime S. Cardoso, and Jessica Fernandes, *"transfer learning with partial observability applied to cervical cancer screening"*, Proceedings of Iberian Conference on Pattern Recognition and Image Analysis (IbPRIA), 2017.

[4] Frauke Günther and Stefan Fritsch, *neuralnet: Training of Neural Networks*, The R Journal **2** (2010), no. 1, 30–38.

[5] Kevin Kiernan and Ionut E. Iacob, *Electronic Beowulf*, online electronic edition, 2018, http://ebeowulf.uky.edu/.

[6] Paulo J. Lisboa and Azzam F.G. Taktak, *The use of artificial neural networks in decision support in cancer: A systematic review*, Neural Networks **19** (2006), no. 4, 408 – 415.

[7] Olvi L. Mangasarian, W. Nick Street, and William H. Wolberg, *Breast Cancer Diagnosis and Prognosis Via Linear Programming*, Operations Research **43** (1995), no. 4, 570–577.

[8] Laurie J. Mango, *Computer-assisted cervical cancer screening using neural networks*, Cancer Letters **77** (1994), no. 2, 155 – 162, Computer applications for early detection and staging of cancer.

[9] Yu Qiao, *The MNIST Database of handwritten digits*, online, retrived February 2018, 2007, http://www.gavo.t.u-tokyo.ac.jp/q̃iao/database.html.

[10] Xiaoping Qiu, Ning Tao, Yun Tan, and Xinxing Wu, *Constructing of the risk classification model of cervical cancer by artificial neural network*, Expert Systems with Applications **32** (2007), no. 4, 1094 – 1099.

[11] Alejandro Lopez Rincon, Alberto Tonda, Mohamed Elati, Olivier Schwander, Benjamin Piwowarski, and Patrick Gallinari, *Evolutionary optimization of convolutional neural networks for cancer mirna biomarkers classification*, Applied Soft Computing (2018).

[12] Abid Sarwar, Vinod Sharma, and Rajeev Gupta, *Hybrid ensemble learning technique for screening of cervical cancer using papanicolaou smear image analysis*, Personalized Medicine Universe **4** (2015), 54 – 62.

[13] Siti Noraini Sulaiman, Nor Ashidi Mat-Isa, Nor Hayati Othman, and Fadzil Ahmad, *Improvement of features extraction process and classification of cervical cancer for the neuralpap system*, Procedia Computer Science **60** (2015), 750 – 759, Knowledge-Based and Intelligent Information & Engineering Systems 19th Annual Conference, KES-2015, Singapore, September 2015 Proceedings.

[14] Wikipedia, *Optical character recognition*, online, 2018, https://en.wikipedia.org/wiki/Optical_character_recognition.

Appendix A

R CODE

A.1 READING AND NORMALIZING IMAGES DATA

```
#####################################################################
#
# Reads image data from SOURCE folder and all letters in LETT list
# For each letter NLETT image samples are expected.
#
#####################################################################

library(pixmap)
#read raw image data and determine max dimensions for each image
W <- 0
H <- 0
N <- length(LETT)
alldata <- list()
for (i in 1:N) {
  dir <- paste(SOURCE, LETT[i], sep = "/")
  #j <- 1:NLETT
#limg <- lapply(j,function(j) read.pnm(paste(dir, "/",j,".pgm",sep = "")
  limg <- list()
  for (j in 1:NLETT) {
    #path to image file
    f <- paste(dir, "/", j, ".pgm", sep = "")
    #read image matrix
```

```
    im <- read.pnm(f)@grey
    #trim image (remove empty spage around margins)
    rn <- dim(im)[1]
    cn <- dim(im)[2]
    #1.top & bottomw empty rows
    k <- 1:rn
    rs <- unlist(lapply(k, function(k) sum(im[k,]) > 0))
    #2.left & right empty columns
    k <- 1:cn
    cs <- unlist(lapply(k, function(k) sum(im[,k]) > 0))
    #3. trim
    im <- im[as.vector(rs),as.vector(cs)]

    #compute max dimensions
    H <- max(H, dim(im)[1])
    W <- max(W, dim(im)[2])

    #store in list
    limg[[j]] <- im
  }


  alldata[[LETT[i]]] <- limg
}

########################################
```

```
#normalize data
########################################
#pad with zeros (to the right and to the bottom) up to max dimensions
for (i in 1:N) {
  #extract the list of letter images
  limg <- alldata[[i]]
  for (j in 1:NLETT) {
    #extract the letter image matrix
    im <- limg[[j]]
    #extract the image dimensions
    rn <- dim(im)[1]
    cn <- dim(im)[2]
    #pad if necessary
    if (rn < H || cn < W) {
      #matrix of zeros
      imn <- matrix(0, nrow = H, ncol = W)
      #put the data in the top-left corner
      imn[1:rn,1:cn] <- im
      #store the new matrix
      alldata[[i]][[j]] <- imn
    }
  }
}
```

A.2 IMAGE VECTORIZATION

```
library(functional)
```

```
#############################################################
#
# A collection of methods to convert an image matrix to
# a vector.
#
#############################################################

##############################
# Method: by colum
# Parameters:
# m = the image matrix
# Value: the image vector
##############################
vectorize1 <- function(m) {
  x <- t(c(m))

  return (x)
}

##############################
# Method: by row
# Parameters:
# m = the image matrix
# Value: the image vector
##############################
```

```
vectorize2 <- function(m) {
  x <- t(c(t(m)))

  return (x)
}

##############################
# Method: by rows and columns
# Parameters:
# m = the image matrix
# Value: the image vector
##############################
vectorize3 <- function(m) {
  x <- vectorize1(m)
  y <- vectorize2(m)

  return (cbind(x,y))
}

##############################
# Method: by im*t(im)
# Parameters:
# m = the image matrix
# Value: the image vector
##############################
vectorize4 <- function(m) {
```

```
  x <- t(c(m %*% t(m)))

  return (x)
}

##############################
# Method: by t(im)*im
# Parameters:
# m = the image matrix
# Value: the image vector
##############################
vectorize5 <- function(m) {
  x <- t(c(t(m) %*% m))

  return (x)
}

##############################
# Method: by column correlation
# Parameters:
# m = the image matrix
# Value: the image vector
##############################
vectorize6 <- function(m) {
  cm <- cor(m)
  cm[!is.finite(cm)] <- 0
```

```
  x <- t(c(cm))

  return (x)
}

##############################
# Method: by row correlation
# Parameters:
# m = the image matrix
# Value: the image vector
##############################
vectorize7 <- function(m) {
  cm <- cor(t(m))
  cm[!is.finite(cm)] <- 0
  x <- t(c(cm))

  return (x)
}

##############################
# Method: by SVD and approx matrix
# Parameters:
# m = the image matrix
# Value: the image vector
##############################
vectorize8 <- function(im, NSVs = 5) {
```

```
  svd <- svd(im)

  #vectorize
  x <- t(c(svd$u[,1:NSVs]))
  #y <- t(c(svd$v[1:NSVs,]))
  #x <- cbind(x,y)
  #im <- svd$u[,1:NSVs] %*% diag(svd$d[1:NSVs]) %*% t(svd$v[,1:NSVs])
  #     x <- t(c(im))
}
```

A.3 Experiment 1

```
#######################################################
#
# Experiment 1: All letters classification using
#                matrix vectorization by columns.
#
#######################################################
#set memory limit
memory.limit(6410241024*1024)

#all letters
LETT <- c("a", "ae", "b", "c", "d", "e", "eth", "f", "g", "h",
    "i", "l", "m", "n", "o", "p", "r", "s", "t", "thorn", "u", "w")
SLETT <- LETT
#data folder
SOURCE <- "../OEJan31"
```

```
#number of samples
NLETT <- 20
#the training fraction (typically 80% training, 20% testing)
PTRAIN <- 0.8

#read data; all data will be stored in "alldata" list
source("readData.R")

#load image2vector functions
source("imagetovector.R")

#format each image matrix as a vector;
#Put each image vector in a data frame
size <- dim(alldata[[1]][[1]])
df.training <- data.frame()
df.testing <- data.frame()
N <- length(SLETT)

set.seed(222)
tf <- PTRAIN
ind1 <- sample(NLETT, tf * NLETT)
ind2 <- (1:NLETT)[-ind1]

for (i in 1:length(LETT)) {
```

```
  #extract the list of letter images
  limg <- alldata[[i]]
  #let <- LETT[i]
  let <- rep(0, N)
  idx <- which(SLETT == LETT[i])[1]
  if (is.na(idx)) {
    idx = N
  }
  let[idx] <- 1
  for (j in ind1) {
    #extract the letter image matrix
    im <- limg[[j]]
    x <- vectorize1(im)  #by cols
    #add to frame
    df.training <- rbind(df.training,cbind(x, t(let)))
  }
  for (j in ind2) {
    #extract the letter image matrix
    im <- limg[[j]]
    x <- vectorize1(im)  #by cols
    #add to frame
    df.testing <- rbind(df.testing,cbind(x, t(let)))
  }
}

df <- rbind(df.training, df.testing)
```

```
#################################################################
# Perform classification using NN
#################################################################

cols <- ncol(df)

#Neural Networks with h hidden layers
library(neuralnet)
set.seed(333)

#ann parameters
#no. hidden layers
h <- c(32)
#the predictors
data.pred <- 1:(cols - N)
#the target variable(s)
data.tar <- (cols - N + 1) : cols
#the prediction formula left-hand-side terms
pflhst <- colnames(df)[data.tar]
#the prediction formula right-hand-side terms
pfrhst <- colnames(df)[data.pred]
pfl <- paste(pflhst, collapse = '+')
pfr <- paste(pfrhst, collapse = '+')
sigmoid = function(x) {
  1 / (1 + exp(-x/20))
```

```
}
#ann model
ann.formula <- as.formula(paste(pfl, '~', pfr))
ann.model <- neuralnet(ann.formula,
                       data = df.training,
                       hidden = h,
                       err.fct = "sse",
                       threshold = 0.01, #default 0.01
                       stepmax = 1e+05,  #default 1e+05
                       rep = 2,          #default 1
                       #act.fct = sigmoid,#default "logistic"
                       linear.output = F)
#plot(ann.model)

# Use the ANN model for Prediction
ann.output<- compute(ann.model, df.testing[,data.pred])

# Display the results

# Compute and Show the confusion matrix

ans.orig <- list()
ans.pred <- list()
for (r in 1:nrow(df.testing)) {
  i <- which.max(df.testing[r,data.tar])
  ans.orig[r] <- SLETT[i]
```

```
  i <- which.max(ann.output$net.result[r,])
  ans.pred[r] <- SLETT[i]
}

print(paste0('Confusion Table'))
print(table(unlist(ans.orig), unlist(ans.pred),
       dnn = c("Original", "Predicted")))
ctable <- as.data.frame.matrix(table(unlist(ans.orig), unlist(ans.pred),
          dnn = c("Original", "Predicted")))

print("Accuracy: ")
print(sum(unlist(ans.orig) == unlist(ans.pred))/length(ans.orig))
```

A.4 Experiment 2

```
#######################################################
#
# Experiment 2: All letters classification using
#               various matrix vectorization methods.
#
#######################################################
#set memory limit
memory.limit(6410241024*1024)

#all letters
LETT <- c("a", "ae", "b", "c", "d", "e", "eth", "f", "g", "h",
   "i", "l", "m", "n", "o", "p", "r", "s", "t", "thorn", "u", "w")
```

```
SLETT <- LETT

#data folder
SOURCE <- "../OEJan31"



#number of samples
NLETT <- 20
#the training fraction (typically 80% training, 20% testing)
PTRAIN <- 0.8

#read data; all data will be stored in "alldata" list
source("readData.R")

#load image2vector functions
source("imagetovector.R")

#format each image matrix as a vector;
#Put each image vector in a data frame.
size <- dim(alldata[[1]][[1]])
df.training <- data.frame()
df.testing <- data.frame()
N <- length(SLETT)

set.seed(222)
tf <- PTRAIN
```

```
ind1 <- sample(NLETT, tf * NLETT)
ind2 <- (1:NLETT)[-ind1]

for (i in 1:length(LETT)) {
  #extract the list of letter images
  limg <- alldata[[i]]
  #let <- LETT[i]
  let <- rep(0, N)
  idx <- which(SLETT == LETT[i])[1]
  if (is.na(idx)) {
    idx = N
  }
  let[idx] <- 1
  for (j in ind1) {
    #extract the letter image matrix
    im <- limg[[j]]
#     x <- vectorize1(im)  #by cols
#     x <- vectorize2(im)  #by rows
#     x <- vectorize3(im)  #both row and cols
#     x <- vectorize4(im) #by im*t(im) <- very poor
#     x <- vectorize5(im) #by t(im)*im <- very poor
#     x <- vectorize6(im)  #col correlation
#     x <- vectorize7(im)  #row correlation
    x <- vectorize8(im, 1) #svd
    #add to frame
    df.training <- rbind(df.training,cbind(x, t(let)))
```

```
  }
  for (j in ind2) {
    #extract the letter image matrix
    im <- limg[[j]]
    #     x <- vectorize1(im)  #by cols
    #     x <- vectorize2(im)  #by rows
    #     x <- vectorize3(im)  #both row and cols
    #     x <- vectorize4(im) #by im*t(im) <- very poor
    #     x <- vectorize5(im) #by t(im)*im <- very poor
    #x <- vectorize6(im)  #col correlation
    #     x <- vectorize7(im)  #row correlation
         x <- vectorize8(im, 1) #svd
    #add to frame
    df.testing <- rbind(df.testing,cbind(x, t(let)))
  }
}

df <- rbind(df.training, df.testing)

############################################################
# Perform classification using NN
############################################################

cols <- ncol(df)

#Neural Networks with h hidden layers
```

```
library(neuralnet)
set.seed(333)

#ann parameters
#hidden layers
h <- c(32)
#the predictors
data.pred <- 1:(cols - N)
#the target variable(s)
data.tar <- (cols - N + 1) : cols
#the prediction formula left-hand-side terms
pflhst <- colnames(df)[data.tar]
#the prediction formula right-hand-side terms
pfrhst <- colnames(df)[data.pred]
pfl <- paste(pflhst, collapse = '+')
pfr <- paste(pfrhst, collapse = '+')
sigmoid = function(x) {
  1 / (1 + exp(-x/20))
}
#ann model
ann.formula <- as.formula(paste(pfl, '~', pfr))
ann.model <- neuralnet(ann.formula,
                       data = df.training,
                       hidden = h,
                       err.fct = "sse",
                       threshold = 0.01, #default 0.01
```

```
                        stepmax = 1e+05,  #default 1e+05
                        rep = 2,          #default 1
                        #act.fct = sigmoid, #default "logistic"
                        linear.output = F)
#plot(ann.model)

# Use the ANN model for Prediction
ann.output<- compute(ann.model, df.testing[,data.pred])

# Display the results

# Compute and Show the confusion matrix

ans.orig <- list()
ans.pred <- list()
for (r in 1:nrow(df.testing)) {
  i <- which.max(df.testing[r,data.tar])
  ans.orig[r] <- SLETT[i]
  i <- which.max(ann.output$net.result[r,])
  ans.pred[r] <- SLETT[i]
}

print(paste0('Confusion Table'))
print(table(unlist(ans.orig), unlist(ans.pred),
      dnn = c("Original", "Predicted")))
ctable <- as.data.frame.matrix(table(unlist(ans.orig),
```

```
        unlist(ans.pred), dnn = c("Original", "Predicted")))

print("Accuracy: ")
print(sum(unlist(ans.orig) == unlist(ans.pred))/length(ans.orig))
```

A.5 EXPERIMENT 3

```
#######################################################
#
# Experiment 3: Groups of letters with "other" option
#               for letters outside the group.
#
#######################################################
#set memory limit
memory.limit(6410241024*1024)

#all letters
LETT <- c("a", "ae", "b", "c", "d", "e", "eth", "f", "g", "h",
"i", "l", "m", "n", "o", "p", "r", "s", "t", "thorn", "u", "w")

#data folder
SOURCE <- "../OEJan31"

SLETTbest0 <- c("c", "l", "m", "s", "thorn", "u") #h <- c(20)

SLETTbest01 <- c("e",
          "i", "o","w")
```

```
SLETTbest1 <- c('b', 'c', 'e', 'u')

SLETTbest2 <- c("a", "d", "s", "thorn", "w")

SLETTbest3 <- c("ae", "l", "m", "o", "p")

SLETTbest4 <- c("eth", "g", "h", "i")

SLETTbest5 <- c("f", "n", "r", "t")

#add "other" to the list
SLETT <- cbind(t(SLETTbest01), c("zz"))



#number of samples
NLETT <- 20
#the training fraction (typically 80% training, 20% testing)
PTRAIN <- 0.8

#read data; all data will be stored in "alldata" list
source("readData.R")

#load image2vector functions
source("imagetovector.R")
```

```
#format each image matrix as a vector; put image vector in a data frame
size <- dim(alldata[[1]][[1]])
df.training <- data.frame()
df.testing <- data.frame()
N <- length(SLETT)

set.seed(222)
tf <- PTRAIN
ind1 <- sample(NLETT, tf * NLETT)
ind2 <- (1:NLETT)[-ind1]

for (i in 1:length(LETT)) {
  #extract the list of letter images
  limg <- alldata[[i]]
  #let <- LETT[i]
  let <- rep(0, N)
  idx <- which(SLETT == LETT[i])[1]
  if (is.na(idx)) {
    idx = N
  }
  let[idx] <- 1
  for (j in ind1) {
    #extract the letter image matrix
    im <- limg[[j]]
```

```
#         x <- vectorize1(im)  #by cols
#     x <- vectorize2(im)  #by rows
#     x <- vectorize3(im)  #both row and cols
#     x <- vectorize4(im) #by im*t(im) <- very poor
#     x <- vectorize5(im) #by t(im)*im <- very poor
#     x <- vectorize6(im)  #col correlation
#     x <- vectorize7(im)  #row correlation
    x <- vectorize8(im, 1) #svd
    #add to frame
    df.training <- rbind(df.training,cbind(x, t(let)))
  }
  for (j in ind2) {
    #extract the letter image matrix
    im <- limg[[j]]
    #     x <- vectorize1(im)  #by cols
    #     x <- vectorize2(im)  #by rows
    #     x <- vectorize3(im)  #both row and cols
    #     x <- vectorize4(im) #by im*t(im) <- very poor
    #     x <- vectorize5(im) #by t(im)*im <- very poor
    #     x <- vectorize6(im)  #col correlation
    #     x <- vectorize7(im)  #row correlation
        x <- vectorize8(im, 1) #svd
    #add to frame
    df.testing <- rbind(df.testing,cbind(x, t(let)))
  }
}
```

```
df <- rbind(df.training, df.testing)

#################################################################
# Perform classification using NN
#################################################################

cols <- ncol(df)

#Neural Networks with h hidden layers
library(neuralnet)
set.seed(333)

#ann parameters
#hidden layers
h <- c(64)
#the predictors
data.pred <- 1:(cols - N)
#the target variable(s)
data.tar <- (cols - N + 1) : cols
#the prediction formula left-hand-side terms
pflhst <- colnames(df)[data.tar]
#the prediction formula right-hand-side terms
pfrhst <- colnames(df)[data.pred]
pfl <- paste(pflhst, collapse = '+')
pfr <- paste(pfrhst, collapse = '+')
```

```
sigmoid = function(x) {
  1 / (1 + exp(-x/20))
}
#ann model
ann.formula <- as.formula(paste(pfl, '~', pfr))
ann.model <- neuralnet(ann.formula,
                       data = df.training,
                       hidden = h,
                       err.fct = "sse",
                       threshold = 0.01, #default 0.01
                       stepmax = 1e+05, #default 1e+05
                       rep = 2,          #default 1
                       #act.fct = sigmoid,#default "logistic"
                       linear.output = F)
#plot(ann.model)

# Use the ANN model for Prediction
ann.output<- compute(ann.model, df.testing[,data.pred])

# Display the results

# Compute and Show the confusion matrix

ans.orig <- list()
ans.pred <- list()
for (r in 1:nrow(df.testing)) {
```

```
  i <- which.max(df.testing[r,data.tar])
  ans.orig[r] <- SLETT[i]
  i <- which.max(ann.output$net.result[r,])
  ans.pred[r] <- SLETT[i]
}

print(paste0('Confusion Table'))
print(table(unlist(ans.orig), unlist(ans.pred),
       dnn = c("Original", "Predicted")))

print("Accuracy: ")
print(sum(unlist(ans.orig) == unlist(ans.pred))/length(ans.orig))
```