DEEP LEARNING CONSIDERING PATTERN INVARIANCE CONSTRAINT

A THESIS SUBMITTED TO THE
GRADUATE SCHOOL OF APPLIED SCIENCES
OF
NEAR EAST UNIVERSITY

By

OYEBADE KAYODE OYEDOTUN

In Partial Fulfillment of the Requirements for the Degree of Master of Science

In

Electrical & Electronic Engineering

NICOSIA, 2015

Oyebade Kayode Oyedotun: Deep Learning Considering Pattern Invariance Constraint

Approval of Director of Graduate School of Applied Sciences

Prof. Dr. İlkay SALİHOĞLU

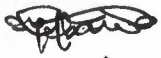
We certify this thesis is satisfactory for the award of the degree of Masters of Science in Electrical and Electronic Engineering

Examining Committee in Charge:

Prof. Dr. Rahib Abiyev	Computer Engineering Department, NEU
Assist. Prof. Dr. Boran Şekeroğlu	Information Systems Engineering Department, NEU
Assist. Prof. Dr. Kamil Dimililer	Electrical & Electronic Engineering Department, NEU
Assist. Prof. Dr. Ali Serener	Electrical & Electronic Engineering Department, NEU
Assist. Prof. Dr. Elbrus Imanov	Computer Engineering Department, NEU
Assist. Prof. Dr. Kamil Dimililer	Supervisor, Electrical & Electronic Engineering Department, NEU

I hereby declare that all information in this document has been obtained and presented in accordance with the academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all materials and results that are not original to this work.

Name, last name: Oyebade Kayode Oyedotun

Signature: 

Date: 14/05/2015

ACKNOWLEDGMENT

I would like to sincerely thank Assist. Prof.Dr. Kamil Dimililer for his guidance, understanding, patience, and most importantly, his supervision during my graduate studies at Near East University.

My unreserved gratitude also goes to Assist. Prof.Dr. Ali Serener for providing an academic platform where students can acquire well-rounded knowledge relating to advance expertise in Electrical & Electronic Engineering.

I cannot but also acknowledge Prof.Dr. Adnan Khashman, who introduced us to the concept of machine intelligence, the grace and method with which he presented artificial neural networks is unrivalled.

I would also like to NEU Grand library administration members, since it provided me with the appropriate environment for conducting my research and writing my thesis.

The support and love shown by my parents, Dr. & Mrs. Elijah Olusoji Oyedotun, and my siblings during my thesis are remarkable, and I really appreciate you.

Also, many thanks to my fellow colleague, Ebenezer Olaniyi, whose discussions and interest were quite important to the success of this work.

In addition, my heartfelt appreciation to Gideon Joseph, Kingsley Agu, Amina Abubakar, and Tabitha Joseph, for roles played individually.

Lastly, God must be acknowledged for the grace and sound mind, he bestowed on humans to achieve great things, and to master the universe; He has always reflected that His faith in my endowment is boundless.

ABSTRACT

The ability of human visual processing system to accommodate and retain clear understanding or identification of patterns irrespective of their orientations and presentations is quite plausible. Although, this area of computer vision has recently received a massive boost in interest by researchers, the situation is far from being resolved. The problem of pattern invariance in the computer vision world is not one that can be overemphasized; obviously one's definition of an intelligent system broadens considering the large variability with which same patterns can occur and have to be coped with. This research investigates the performance of feedforward networks against convolutional and deep neural networks when tasked with recognition problems, considering pattern invariances such as translation, rotation, scale, and moderate noise levels. The architecture of the considered networks in relation to the human visual perception processing has also been explored as a reference to the built-in invariances achievable from these networks due to structure and learning paradigms. Although, single hidden layer or shallow networks have the capability to approximate any function, the benefits of having several hidden layers of such networks have been considered and hypothesized by researchers for some time, but the difficulty in training these networks has led to little attention given to them. Recently, the breakthrough in training deep networks through various pre-training schemes have led to the resurgence and massive interest in them, significantly outperforming shallow networks in several pattern recognition contests; moreover the more elaborate distributed representation of knowledge present in the different hidden layers concords with findings on the biological visual cortex. This research work reviews some of the most successful pre-training approaches to initializing deep networks such as stacked denoising auto encoders, and deep belief networks based on achieved error rates, computational requirements, and training time. Also, as it has been debated among researchers, the results of this research suggest that the optimization effect obtained from network pre-training dominates the regularization effect, though both effects are achieved. While, various patterns can be used to validate this query, handwritten Yoruba vowel characters have been used in this research. Databases of the images containing pattern constraints of interest were collected, processed, and used to train the designed networks.

Keywords: Artificial neural networks, deep learning, pattern invariance, character recognition, Yoruba vowel characters

ÖZET

karşılamak ve net bir anlayış ya da desen kimlik korumak için insan görsel işleme sisteminin yeteneği ne olursa olsun yönelim ve sunumlar oldukça makuldür. Bilgisayar vizyonu bu alanda son zamanlarda araştırmacılar tarafından ilgi büyük destek aldı rağmen, durum çok çözülmüş olmaktan değildir. bilgisayar vizyonu dünyada desen değişmeyen sorunu gereğinden fazla vurgulanan edilebilir biri değildir; Açıkçası akıllı bir sistem kişinin tanımı aynı desenler ortaya ve başa gereken hangi büyük değişkenliği göz önünde genişletmektedir. Bu araştırmalar, çeviri, döndürme, ölçek ve orta gürültü seviyeleri gibi model invariances dikkate tanıma problemleri ile görevli katlamalı ve derin sinir ağları, karşı ileri beslemeli ağlar performansını inceler. İnsan görsel algı işleme ilişkin dikkate ağların mimarisi de yapı ve öğrenme paradigmaları nedeniyle bu ağlardan elde yerleşik invariances bir referans olarak incelenmiştir. Tek gizli katman veya sığ ağlar herhangi bir işlevi yaklaştığı yeteneğine sahip olsa da, bu tür ağların çeşitli gizli katmanları olan faydaları kabul edilir ve bir süre araştırmacılar tarafından hipotez, ama bu ağları eğitimi konusunda zorluk verilen biraz dikkat yol açtı edilmiştir Onlara. Son zamanlarda, çeşitli ön-eğitim programları aracılığıyla derin ağları eğitimi konusunda atılım anlamlı birkaç örüntü tanıma yarışmalarında sığ ağları geride bırakarak, onları yeniden dirilişi ve kitlesel ilgi yol açmıştır; Biyolojik görsel korteks üzerinde bulguları ile farklı gizli katmanlar concord un mevcut bilginin üstelik daha ayrıntılı dağıtılmış gösterimi. Bu araştırma çalışmaları gibi elde hata oranları, hesaplama gereksinimleri ve eğitim süresine göre yığılmış denoising oto enkoderler ve derin bir inanç ağları gibi derin ağlar başlatılıyor yaklaşımları en başarılı öncesi eğitimin bazı değerlendirmeleri. Bu araştırmacılar arasında tartışma konusu olmuştur olarak da, bu araştırmanın sonuçları, her iki etkileri elde olsa ağ öncesi eğitimden elde edilen optimizasyon etkisi, düzenlilestirme etkisi hakim olduğunu göstermektedir. Çeşitli desenler bu sorguyu doğrulamak için kullanılabilir iken, el yazısı Yoruba ünlü karakterleri bu araştırmada kullanılmıştır. Ilgi desen kısıtlamaları içeren görüntülerin Veritabanları toplandığı, işlendiği ve tasarlanmış ağları eğitmek için kullanıldı.

Anahtar Kelimeler: Yapay sinir ağları, derin öğrenme, desen değişmezliği, karakter tanıma, Yoruba ünlü karakterler

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER ONE
INTRODUCTION

Images, and therefore patterns are very important data to humans. They are invariably one of the easiest and fastest way human beings assimilate and appreciate information; the ease and speed with which we process the details of images are very amazing.

The human brain is quite able to capture and analyse images almost effortlessly, recognizing even intrinsic patterns that are sometimes embedded in the images. The best understanding of how human beings achieve these tasks are still somewhat subject to some scientific debate and research is still ongoing .e.g. bottom-to-top or top-to-bottom hierarchical process of perception or an integration of both. 'It is a difficult experimental issue to determine the relative importance of bottom-up and top-down processes' (Delorme, Rousselet, Mace´, Fabre-Thorpe, 2004).

Images are arguably one of the most used data formats in the intelligent systems and computing world; this is evident in the ease with which they can be captured and lend themselves to the representation of information that needs to be processed for further use. e.g. control process, database content based search etc.

However, it is very somewhat evident that such tasks are not that too easily achievable in the computing world; especially computer vision.

Furthermore, the volume of image data that is now available to us from different sources have grown exponentially recently, especially due to a surge in internet accessibility, large memory mobile phones and related devices that keep being churned out by electronics companies. Unfortunately, we more than often require some sense of content-based image filtering or sorting to make efficient use of these data.

A recognition system is a system that has the intrinsic capability to accept data patterns and output the corresponding classes to which the inputs belong; actually, the act of matching the inputs to output classes using perfect examples is known as template matching.

Generally, the pattern to be used in such systems as input are presented as images; occasionally after some image processing might have been performed on them.

Character recognition is the process of having a system that has the capability to identify sample of characters with which it has been trained. The identification process is seen in the test phase of the system, where new characters are supplied to the system for identification. The system is meant to accept the characters as input and output the classes of the characters; this phase is sometimes also known as the simulation phase.

It is worthy of note that such a recognition system would lose its concept of being considered "intelligent" if it cannot also recognize to an extent characters with which it has not been trained with but are quite similar to the ones it has been trained with i.e. such a system should possess good generalization power for characters recognition.

1.1 Contributions of Research

1. Designing intelligent recognition systems for Yoruba vowel characters

2. Investigating the built-in tolerance achieved by the recognition systems to variances in input patterns such as scale, translation, rotation, and noise.

3. Evaluating the performance of each designed model based on achieved error rates and computational requirements, and training time.

5. Associate the architecture of the considered networks to the built-in invariances achieved by the networks.

4. Establishing the optimization (under-fitting) and regularization (over-fitting) effects of pre-training in deep networks, and therefore also validating which effect is greater.

1.2 Scope of Research

The scope of this research work will be limited to the design of various intelligent recognition systems for Yoruba vowel characters. Handwritten images of characters will be collected from selected individuals; processed and used to train the designed systems. Also, validation of learning will be carried out concurrently during training; after which the systems are simulated with character images which were not used to train the systems. It is to be noted

that varying degree of translation, rotation, scale, and noise levels in patterns will be used for testing.

Hence, the performance of each model will then be evaluated on based on some performance parameters.

1.3 Approaches to Pattern Recognition and Applications

Intelligent pattern recognition involves using features and the structure of such features derived from objects in grouping them into their corresponding classes.

Description of objects or pattern is the stage where unique ways to describe objects or patterns are developed. It is from these features that rules for identifying objects are derived. It is usually the job of designers to craft such rules and how objects or patterns are to be represented.

In template matching, a test pattern is presented to a recognition system, which it compares with the stored templates, then outputs the class of the input pattern based on correlation of matching. i.e. outputted class of the test pattern belongs to the class of the template to which it has the highest correlation.

This approach aims to generate a standard perfect object, example or pattern to represent a group (class) with which other objects, examples or patterns are compared. It is worthy of note that this method may involve the use of the whole object (pattern) representation (e.g. whole image) which is considered as global template matching or some regions of the whole object (pattern) representation (e.g. some regions of an image) which is considered as local template matching.

Inasmuch as this suffices in lots of situations for recognition systems, the disadvantage lies in that only perfect example, therefore perfect data can be correctly classified. The system lacks flexibility to moderate variations in the presented data for recognition, and hence termed "non-intelligent"; more technically, one can say that such recognition systems lack tolerance to translational variance, rotational variance and scale mismatch.

Syntactic approach is also known as feature analysis, in which some important descriptors or features that allow objects or patterns to be represented as uniquely as possible are extracted.

The structural combination of such descriptors is what is leveraged on when such a recognition system is building the identification of patterns. The descriptors are parsed using some set algorithms so that the whole pattern can be realized.

Two of the methods used for feature analysis are:

- Stroke analysis: patterns are classified from analysing their vertical and horizontal line structures
- Geometric feature analysis: using the general form of a pattern and defining some geometric shapes within the pattern (Khashman, 2014).

More advance feature analysis methods exist in image processing, speech processing and other complex problems.

Another technique for pattern recognition uses statistical models of patterns that have been transformed into data. In contrast to using training data to determine suitable algorithms or heuristics, decision and probability theories are applied.

The features from training data largely determine the choice of statistical model that is adopted; hence other suitable choices for representing features of interest are therefore usually exploited. Common techniques to exploring other options of representations are clustering, principal component analysis, and discriminant analysis.

Intelligent classifiers are recognition systems that have the capability to learn from examples in a phase known as training. They are shown several examples of task they are required to learn, during which they gain experiential knowledge. After learning has been achieved, these systems are tested by supplying them sample images of the ones they have been trained with; to ascertain that proper training has been achieved and not that the system only memorized the examples, variants of image samples are then used during the test phase and error rates at recognition can be determined.

The most commonly used intelligent recognition systems in the field of machine learning are artificial neural networks. Their robustness and tolerance to moderate noise is what makes them very important in intelligent recognition.

Pattern recognition is studied in many fields, including psychology, ethnology, forensics, marketing, artificial intelligence, remote sensing, agriculture, computer science, data mining,

document classification, multimedia, biometrics, surveillance, medical imaging, bioinformatics and internet search (Kidiyo Kpalma and Joseph Ronsin, 2007).

Intelligent recognition for character recognition has also been used significantly in reading off bank checks, postal codes and zip codes; this automated process has significantly sped up the process of check clearing, postal delivery and applications dependent on machine vision.

Recently, intelligent recognition has led to a boom in the field of robotics, as the problem of robotic navigation of its environment has been greatly improved. More sophisticated robots which have a better vision grasp of their environment based on the capability to recognize objects and therefore manoeuvre their path safely are being developed.

1.4 Thesis Overview

The remaining chapters of thesis describe the approaches and methods that have used to achieve the aims of the work.

Chapter two presents the literature reviews of image processing, and neural networks related to this thesis briefly.

Chapter three explains the collection of image data and processing, machine learning capability of neural networks, methodology and design of considered neural network architectures.

Chapter four describes in details the topology and particular network parameters used in the training of the networks.

Chapter five presents the results, analysis and discussions of the simulated networks.

A brief review of the thesis aim, summarizing as conclusion the findings of work and the recommendations as applicable to the findings are supplied in chapter six.

CHAPTER TWO
LITERATURE REVIEW

2.1 Overview

This chapter presents the detailed discussion on computational intelligence, under which is machine learning as the focus, and briefly the basics of artificial neural networks. Also, image processing schemes as are essential to this thesis were discussed. More importantly, this chapter articulates the relationship and importance of the above mentioned sections to the overall realization of this thesis.

2.2 Computational Intelligence

Computational Intelligence is the study of adaptive mechanisms to enable or facilitate intelligent behaviour in complex and changing environments (Engelbrecht, 2007).

It embodies any natural or biological inspired computational paradigms which include not limited to artificial neural networks, evolutionary computing, fuzzy systems, swarm intelligence etc.

Characteristics of computational intelligence systems are listed below (Gary G. Yen, 2014)

-Biologically motivated behaviour such as learning,
-Reasoning, or evolution (in the sense of approximation)
-Parallel, distributed information processing
-Mysterious power under real-world complications
-Lack of qualitative analysis
-Non-repeatable outcomes
-Stochastic nature

2.2.1 Machine learning

While machine learning on the other hand is a branch of computational intelligence involved with systems that can act in certain environments by learning from supplied data. The uniqueness of these systems lie in that they do not have to be domain-specifically programmed. They have intrinsic self-programming nature that allow the same discovery or

learning paradigm to be applied to different problems and still be able to perform satisfactorily.

Learning involves searching through a space of possible hypotheses to find the hypothesis that best fits the available training examples and other prior constraints or knowledge (Mitchell, 1997).

These systems are able to discover patterns, sequences, and relationship in supplied data; hence are very applicable in data filtering or mining.

More significant is the need not to write unique domain specific programming codes each time a problem is to be simulated as this allows for designers to focus more on understanding important features of problems which the network is required to learn; in contrast to convectional digital computing in which enormous time goes into writing huge codes for complex problems.

Generally, machine learning can be broadly divided into supervised, unsupervised and reinforced learning. The classes are explicitly listed below.

1. Supervised learning:

   The network is given examples and concurrently supplied with the desired outputs; the network is generally meant to minimize a cost function in order to achieve this, usually an accumulated error between desired outputs and the actual outputs.

   Examples of systems that use this learning paradigm are support vector machines, neural networks, kernels etc.
2. Unsupervised learning:

   The network is given examples but not supplied with the corresponding outputs; the network is meant to determine patterns between the inputs (examples) accordingly to some criteria and therefore group the examples thus.

   Examples of learning paradigms that use unsupervised learning are clustering, dimensionality reduction, competitive learning, deep learning etc.
3. Reinforced learning:

   There is no desired output presented in the dataset but there is a positive or negative feedback depending on output (desired/undesired) (Hristev, 1998).

   Markov decision process is an example of learning paradigm that is reinforced.

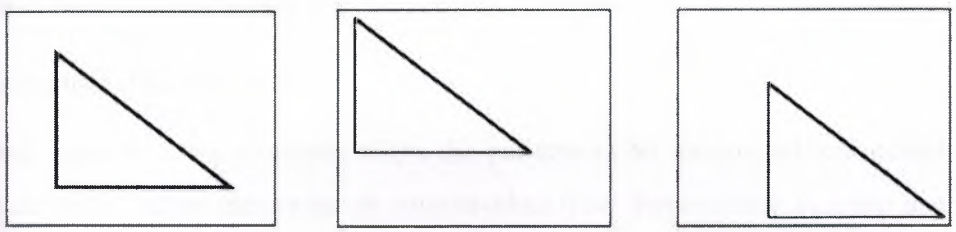
2.3 Pattern Constraints in Recognition Systems

This section describes in brief constraints that recognition systems usually encounter in real life. Even more, is the significance when we consider handwritten recognition, where writing styles and available domain of input space is quite large. Such constraints as considered in this thesis are translation, rotation, scale, and noise. Efficient recognition systems are required to cope fairly well with some of these constraints. i.e. the systems should maintain a relatively good identification of patterns in situations of moderate variances.

2.3.1 Translational invariance

The is the situation where the patterns to be recognized are not centred in the image; they have centres that lie in various part of the whole image. Usually, this involves only linear (horizontal or vertical) shift in the position of patterns in images. It is worthy to state that the distance relationship between pattern components and the scale are not altered in translational variance. This is a serious source of recognition error in non-intelligent systems.

Figure.2.1 shows an original image and some translated image versions of the original.



(a) Original pattern (b) Translated pattern (c) Translated pattern

Figure 2.1: Translational invariance

In figure 2.1, the original centred pattern is shown in (a), translated pattern northeast is shown in (b), and translated pattern south-west is shown in (c).

The depicted problem of translational invariance leads to wrong classification in recognition systems; the problem is usually resolved in either of the two ways given below.

1. Perform an operation on translated patterns to re-centre them in the image frames before feeding as inputs into recognition systems.

Registration algorithms attempt to align a pattern image over a reference image so that pixels present in both images are in the same location. This process is useful in the alignment of an acquired image over a template (McGuire, 1998).

2. The other alternative applicable in intelligent recognition systems is training such systems with translated copies of original images so that experiential knowledge is now broader, and hence such designed systems can handle translated patterns in images.

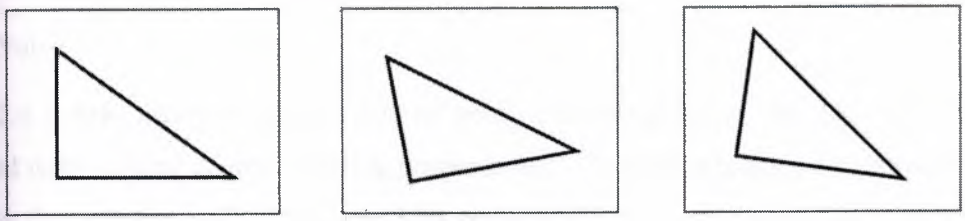
For any specific object, invariance can be trivially "learned" by memorizing a sufficient number of example images of the transformed object (Leibo1, Mutch, Rosasco, Ullman, and Poggio, 2010).

Conversely, a more sophisticated system that is translation invariant can be built. i.e. the centre of patterns in images does not affect recognition. Such systems have built-in structures that allow for the accommodation of moderate pattern variances.

Convolutional neural networks combine three architectural ideas to ensure some degree of shift, scale and distortion invariance (LeCunn, Bottou, Bengio, and Patrick Haffner, 1998).

2.3.2 Rotational invariance

Rotational variance is the situation where the patterns to be recognized are rotated spatially through an angle, either clockwise or counter-clockwise. Recognition systems usually have problems correctly classifying such patterns. This is shown in figure 2.2 below.



(a) Original pattern (b) Rotated pattern (c) Rotated pattern

Figure 2.2: Rotational invariance

It can be seen from figure 2.2 (b) and (c) which are rotated versions of (a), counter-clockwise and clockwise, respectively; and that in template matching, recognition error is probable to occur. Generally, intelligent recognition systems are more robust to rotational variance; this

can usually be built into the system during the learning phase or leveraging on convolutional neural networks based recognition systems.

2.3.3 Scale invariance

Scale mismatch, also known as scale variance, is the situation where input patterns have varying scales. Their locations in the images remain the same, and have not been rotated, but some now appear either blown up or scaled down (smaller). This is shown in Figure 2.3 below.

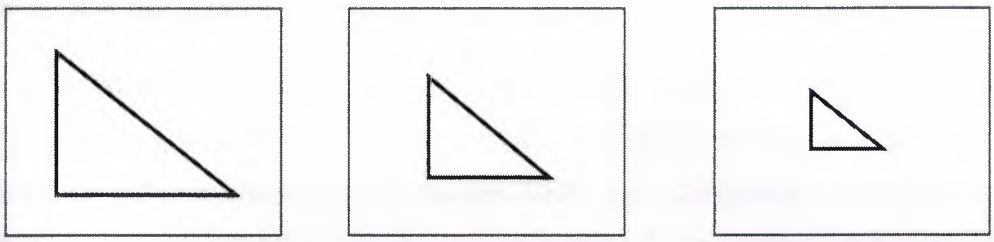


Figure 2.3: Scale invariance

Figure 2.3 (b) and (c) shows downsized copies of the original pattern (a). When such downsized or blown up images are fed into non-intelligent recognition systems, problem of mismatch occurs and patterns may be wrongly identified.

2.3.4 Noise

Figure 2.4 below shows various levels of noise affected patterns; the patterns have been simulated with varying degree of salt & pepper noise. The noise densities for patterns (a), (b), and (c) in figure 2.4 are 15%, 25%, and 35% respectively.

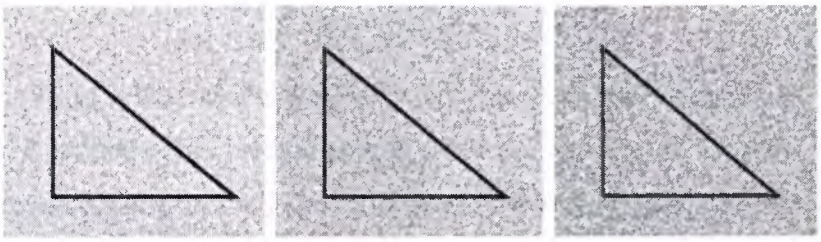


Figure 2.4: Noise

2.4 Image Processing

Digital image processing is the technology of applying a number of computer algorithms to process digital images (Zhou, Wu, Zhang, 2010). It is a very important aspect of computer vision field.

An image can be seen as a two-dimensional function f(x,y), where x and y are the spatial coordinates; the intensity of any part of an image is given by the amplitude of the function at that point. The intensity at a point in an image is sometimes called the gray level at that point (Gonzalez, Woods, 2002).

Image processing finds applications in photography, intelligent systems, bio-medical imaging, remote and forensics. During image processing, we obtain some parameters describing some characteristics of such images. These characteristics are usually employed in manipulating or conditioning images as deemed suitable. Some common operations achieved include translation of colour images to gray scale or binary images (black and white), filtering, segmentation, enhancements, restoration, compression etc.

For the purpose of this research, the image processing schemes that have been used are discussed briefly below.

2.4.1 Gray scale of an image

Generally, colour images have three channels, called the RGB channels, corresponding to the Red, Green, and Blue channels. The three methods for transforming an image from colour (RGB) to gray scale are listed below.

1. Lightness method: This involves taking the average of the maximum and minimum values of the RBG channels, the equation below describes the transformation.

$$f'(x,y) = \frac{(\max(RGB) + \min(RGB))}{2} \tag{2.1}$$

where, f'(x,y) is the transformed pixel for the original RGB pixels

2. Average method: This method simply takes the average of the pixel values for the particular RGB channel, and the formula to achieve this is shown below.

$$f'(x,y) = \frac{(R+G+B)}{3} \tag{2.2}$$

3. Luminosity method: This approach is quite more elaborate, taking into account the human visual perception, considering the fact that humans are most sensitive to green, followed by red, and least to blue colours; hence the weighting of the RGB transformation to a single intensity pixel is achieved as described by the equation below.

$$f'(x,y) = 0.21R + 0.72G + 0.07B \tag{2.3}$$

where, f'(x,y) is the transformed gray scale pixel.

2.4.2 Negative of an image

Generally, when gray images are of the range 0 to 1, the 0 pixels represent black and 1 pixels represent white, values between 0 and 1 represent lighter shades of black.

The negative transform is meant to turn black pixels to white and white pixels to black. This is usually achieved by the equation given below to transform all the individual pixel values.

$$f'(x,y) = 1 - f(x,y) \tag{2.4}$$

where, f'(x,y) is the transformed negative image of f(x,y).

2.4.3 Binarization of an image

This is the process of obtaining an image with only two possible gray levels. i.e. 0 or 1; there are no intermediate gray levels in between the level 0 and 1. There exists various ways of achieving this, but considered in this research is the global thresholding method. This method entails choosing a gray level value between 0 and 1, a value known as the threshold value, and deploying an algorithm such that all pixel values less than the threshold value will be transformed to 0 (black), and all pixels above or equal to the threshold value will be transformed to 1 (white).

$$f'(x,y) = 1, \quad for\ f(x,y) \geq T \tag{2.5}$$

$$f'(x,y) = 0, \quad for\ f(x,y) < T \tag{2.6}$$

where T is the global threshold value.

Equations 2.5 and 2.6 describe how binarization of an image can be achieved.

2.4.4 Image filtering

This is an usually an enhancement operation performed on images, often, noisy images. The operation algorithms are attempts to clean up the image (removing noise). Some of the techniques for achieving this include using mean filters, median filters, Gaussian filters, etc. The median filter has been used for this work and will be discussed briefly below.

The median filter, achieves filtering by taking the median of pixel values over a particular region of the image; usually what is done is that a fixed number of pixels in considered along both axes of the image to a particular pixel (usually with pixel of interest in the centre of the mask); the dimension of the pixels considered along both axes is considered the mask or window size used in the filtering process (e.g. m×n mask would mean m rows and n columns along the x and y axes respectively). The median value of the pixels is taken and used to replace the particular pixel of interest.

2.5 Artificial Neural Network (ANN)

Artificial neural networks as the name indicates are computational networks, which attempt to simulate, in a gross manner, the network of nerve cells (neurons) of the biological (human or animal) central nervous system (Graupe, 2007).
These networks simulate the human biological neural system in both structure and function.
The long course of evolution has given the human brain many desirable characteristics not present in von Neumann or modern parallel computers. These are listed below (Jain, Mao, and Mohiuddin, 1996).

- Massive parallelism,
- Distributed representation and computation,
- Learning ability,
- Generalization ability,
- Adaptability,
- Inherent contextual information processing,
- Fault tolerance, and low energy consumption.

By mimicking biological neuron features such as synapses, dendrites, cell body, and their working principles, corresponding artificial neural network features such as synaptic weights(memories), inputs, artificial neurons(computational units, especially perceptrons) and their firing based on thresholds, total potential and activation functions have can be achieved (Oyedotun and Khashman, 2014).

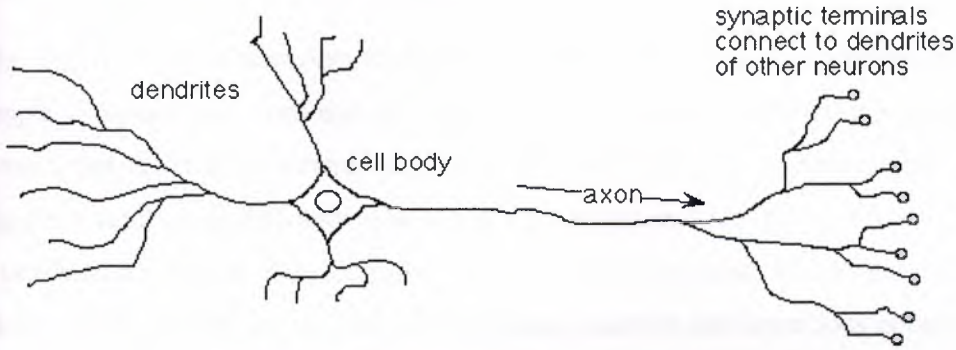


Figure 2.5: Biological neuron (Wilson, 2012)

Figure 2.5 shows a typical biological neuron, and as it can be seen from the comparison table of biological and artificial neurons that biological neurons have lower processing speed (several MHz) compared to todays convectional computers with processing speed ranging in GHz.

Table 2.1: Gross comparison of Biological and Artificial Neurons (Eluyode et al., 2013)

	Biological neuron	**Artificial neuron**
Attribute	Dendrites	Inputs
Attribute	Cell body	Processing element
Attribute	Synapses	Weights (memories)
Attribute	Axon	Output
Processing speed	Slow: Several milliseconds	Fast: Few nanoseconds
Processing system	Massively parallel: 1014 synapses	Massively parallel: 108 transistors

The processing power of artificial neural networks does not decisively rely on the processing speed of each neuron, but in the massively parallel interconnections that exist between such

individual units and hence incredibly ungraded overall processing power for the network is achieved. In essence of this, artificial neural networks behave similarly.

Furthermore, another attribute of artificial neural networks that make them of great interest is their adaptability to solving problems that are inherently 'troubles' for convectional (Von Neumann) computers; problems such as pattern recognition, noisy data etc.

Also, the ability of these networks to represent complex relationships between parameters with simple computational rules makes them quite easy to implement (self-programming). It could even be seen that different problems can be solved with the same learning algorithms (e.g. the learning rule) in neural networks while in convectional digital computers, for each unique problem, a different algorithm invariably must be developed. Moreover, while neural networks are fault tolerant, i.e. the failure of one component or hardware does not mostly lead to the complete collapse or failure of the whole system (inherent redundancy in neural networks). The convectional computers are sequential and the failure of one unit or hardware initially lead to the failure of the system. It takes the failure of a sufficient number of units in the neural networks for a collapse of the whole system to occur, a phenomenon regarded as graceful degradation (Eluyode and Akomolafe, 2013).

Research and advances in medicine and psychology over the years has led to a better understanding of the brain, how activities are achieved, the nature of signals involved and how these reflect in our expressions of such processes.

The growth of artificial neural networks therefore as from inception somehow depended on the results of significant researches in medicine and psychology. It is from these new findings that architectures of neural networks are being modified or new ones built.

Likewise, simulation of hypothesized neural network models and learning rules have been used extensively to validate findings in medicine and psychology.

The first artificial neuron, perceptron, was presented by F. Rosenblatt, 1958, which marked the beginning era of neural networks in computational intelligence.

The basic building unit for artificial neural networks are perceptrons, these perceptrons are massively interconnected in the networks. The connections or links between neurons are known as weights. These weights acts as long-term memory to the network, holding the information corresponding to the present knowledge of the network due to experience.

These neurons compute the weighted sum of the products of weights and inputs individually, and outputs a value known as Total Potential (T.P). The activation of each neuron is then determined by comparing the computed total potential to a reference value known as the threshold; the activation of neurons is also referred to as 'firing'.

Mathematically, it can be shown below that:

-Neuron fires if: T.P ≥ Threshold

-Neuron does not fire if: T.P < Threshold

$$net_j = \sum_{k=1}^{n} w_{kj} x_k \tag{2.7}$$

$$O_j = \varphi(net_j) \tag{2.8}$$

where O_j is the output of the neuron, net_j is the net input to the neuron, Θ_j is the threshold for the neuron, and ϕ is the activation function for the neuron. i.e. net_j is passed through the activation function ϕ to compute the final output of the neuron.

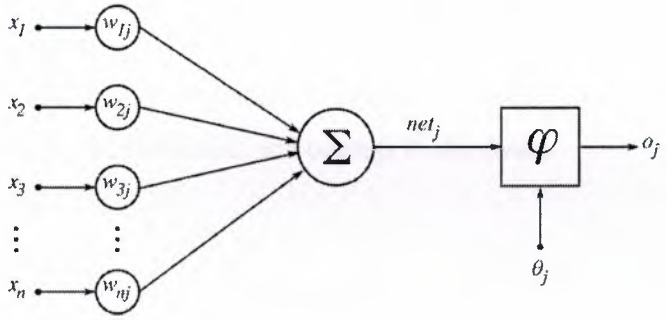


Figure 2.6: Artificial neuron

Sometimes times a bias term, b_j, is added to neurons so that equation 2.7 becomes

$$net_j = \sum_{k=1}^{n} w_{kj} x_k + b_j \tag{2.9}$$

2.5.1 Activation functions used in neural networks

There are several types of activation functions used in artificial neurons; usually, the application of the network determines which is suitable or more appropriate. Common activation functions used in neurons are shown below.

1. Linear activation function

   This outputs directly the weighted sum of the products of the inputs and weights. They are also known as identity functions. i.e.

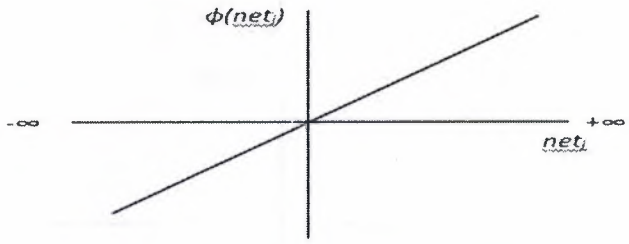


Figure 2.7: Linear activation function

Mathematically,

$$O_j = \varphi(net_j) = net_j \tag{2.91}$$

Sometimes a scalar multiplier is applied in the activation or a bias added so that the intercept of the graph no longer lies at the origin; the bias can be used to shift the graph around on the output axis, hence control the decision boundary.

2. Hard-Limit function

   This function has binary response, it outputs 1 when the Total Potential (T.P) is greater or equal to the threshold, and outputs 0 otherwise.

   Mathematically,

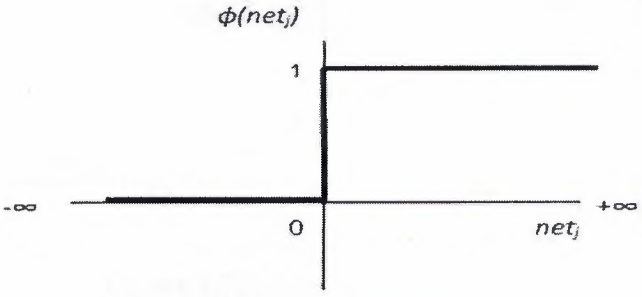


Figure 2.71: Hard-limit function

Mathematically,

$$\text{If, } net_j \geq T.P \text{, then } \quad O_j = \varphi(net_j) = 1 \tag{2.92}$$

$$\text{if, } net_j < T.P \text{, then } O_j = \varphi(net_j) = 0 \tag{2.93}$$

The hard-limit function is what is used in the McCulloch-Pitts model of artificial neurons, and it is sometimes referred to as the Heaviside function.

3. Signum function

   The signum function is similar to the hard-limit function save the fact that the output is either +1 or -1.

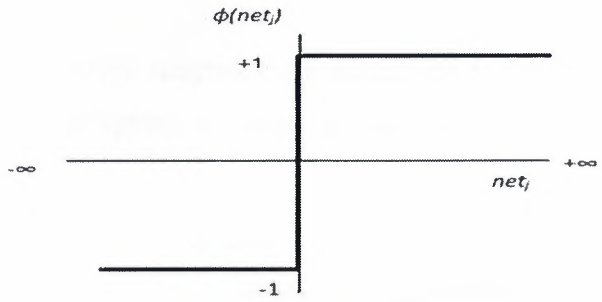


Figure 2.72: Signum function

Mathematically,

$$\text{If, } net_j \geq T.P \text{, then } \quad O_j = \varphi(net_j) = +1 \tag{2.94}$$

$$\text{If, } net_j < T.P \text{, then } O_j = \varphi(net_j) = -1 \tag{2.95}$$

4. Log-Sigmoid function

   The log-sigmoid function is a non-linear s-shape function, the output range is from +1 to 0. Mathematically, the output of sigmoid function is shown below.

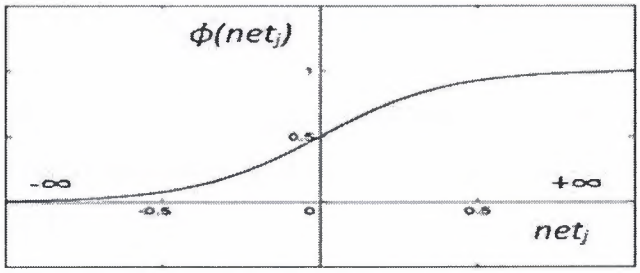


Figure 2.73: Log-Sigmoid function

Mathematically,

$$O_j = \varphi(net_j) = \frac{1}{1+e^{-a(net_j)}} \tag{2.96}$$

The log-sigmoid function is one of the most commonly used activation functions in neural networks. It is sometimes referred to a squashing function because it takes the

value of the net input and compress it to range from +1 to 0. The variable 'α' in equation 2.96 controls the steepness of the function.

For this reason, it is particularly important for networks applying back propagation algorithms. For larger values of 'a', the log-sigmoid function approximates a hard-limit function (Debes, Koenig, Gross, 2005).

5. Tan-Sigmoid function

This function, hyperbolic tangent, is quite similar to the log-sigmoid, it is also s-shaped, but its axis of symmetry passes through the origin and its output range from +1 to -1.

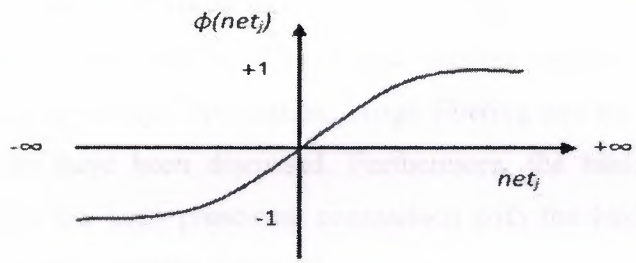


Figure 2.74: Tan-Sigmoid function

Mathematically,

$$O_j = \varphi(net_j) = \frac{e^{a(net_j)} - e^{-a(net_j)}}{e^{a(net_j)} + e^{-a(net_j)}} \tag{2.97}$$

Hyperbolic tangent functions can be used in hidden or output layers of neural networks.

6. Gaussian function

The Gaussian activation function can be used when finer control is needed over the activation range. The output range is 0 to 1; 0 when x=∞, and 1 when x=0 (Sibi, Jones, Siddarth, 2013).

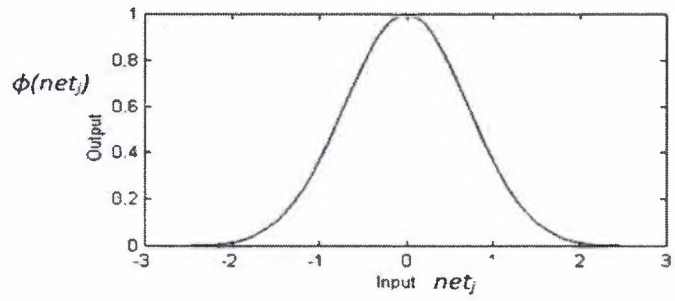


Figure 2.75: Gaussian function

Mathematically,

$$O_j = \varphi(net_j) = e^{-\frac{net_j^2}{2\sigma^2}} \tag{2.98}$$

where σ is used to control the steepness of the curve.

Gaussian activation functions are commonly used in Radial Basis Function Neural Networks.

2.6 Summary

The fundamental discussions of fields or areas to achieving the aim of this thesis have been introduced adequately in this chapter. The image process schemes such gray scale conversion, image negatives, image binarization, image filtering and the algorithms used to achieve these operations have been discussed. Furthermore, the basic understanding of artificial neural networks has been presented, comparison with the biological neuron, and their activations also have been briefly examined.

CHAPTER THREE
DATA COLLECTION, PROCESSING AND NEURAL NETWORKS

3.1 Overview

This chapter presents how the aim of this research was achieved sequentially, going through each design stage.

As it is the aim of this work to investigate different neural networks' responses and tolerance to some common variances in pattern recognition. The typical single hidden-layer back propagation neural network (BPNN) has been chosen for examination against convolutional neural network, and deep networks such as Stacked Auto Encoders (SAEs) and Deep Belief Networks (DBNs). Recent researches have shown that some emergent deep neural network architectures perform significantly better with moderate pattern variances such translation, rotation, scale, and noise, as against single hidden-layer feedforward networks.

Yoruba vowel characters have been used in this research to evaluate the extent to which performances may vary in the investigation domain.

3.2 Collection of data

The database for this research was gathered by asking different people to write the Yoruba vowel characters on a graphic drawing software. These images were then saved as jpeg files.

Figure 3.1: Unprocessed Yoruba vowel characters

The Figure 3.1 above shows a sample of the 7 unprocessed Yoruba vowel characters. The characters were handwritten employing several people; 100 samples were collected for each character.

3.3 Datasets and Stages of Design

The logic and sequence of research are shown below.

- Generate a **training database** of Yoruba vowel characters: A1
- Generate **validating database** for Yoruba vowel characters: A2
- Generate **translated database** for Yoruba vowel character patterns: A3
- Generate **rotated database** for Yoruba vowel character patterns: A4
- Generate **scale different database** for Yoruba vowel character patterns: A5
- Generate **noise affected database** for Yoruba vowel character patterns: A6
- Process image databases as necessary
- Train and validate all the different networks with created database A1 and A2 respectively.
- Simulate the different trained networks with A3, A4, A5, and A6.

3.4 Image Processing Phase

The inputs to the recognition system are images that have been processed as described below.

- **Conversion of images to gray**

The images were checked to ascertain whether conversion to gray is necessary so that image format, therefore pixel values now lie in the gray scale range (0-255) and the 3-dimensionality attributes of the images were reduced to 1-dimensionality.i.e. colour information eliminated. This process is also important in the advent where the designed system is simulated with colour images, as this part takes care of conversion to gray. Original unprocessed handwritten characters used in this research work were gray scale images of size 300 × 400 pixels.

- **Conversion of images to binary**

Recognition systems, which are neural network based, only accepts input in the range 0 to 1; hence the conversion of images to binary. However, in as much as it is possible to normalize the gray images to values from 0 to 1 and then feed as inputs to the recognition system, the use of binary images where suitable greatly reduces computational requirements.
Figure 3.2 below shows the binarized handwritten characters.

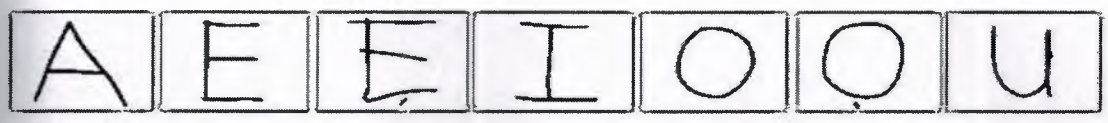


Figure 3.2: Binary Yoruba vowel characters

- **Conversion of images to negative**

The binary images were converted to negatives; this is achieved by subtracting pixel values of images from 1. The output of this process now makes the images' background black and foreground white. Figure 3.3 shows the negatives of the binarized images.

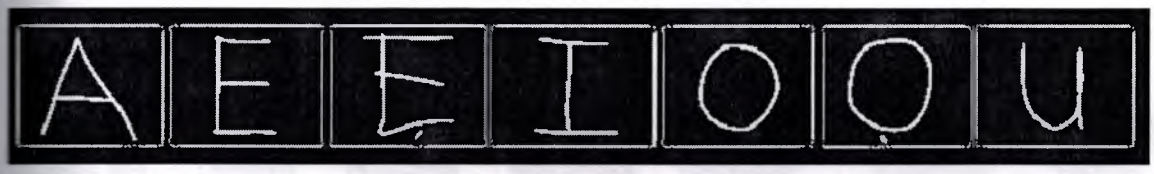


Figure 3.3: Negative Yoruba vowel characters

- **Filtering of images**

The images were filtered using a median filter of mask 10 × 10; this has the effect of smoothening out noises that may be present in the images at this stage and filling in some missing parts based on neighbourhood operations obtainable from the median filter.

Figure 3.4 shows the outcome of of the filtering the negative images.

Figure 3.4: Filtered Yoruba vowel characters

- **Rotation of images**

This done to build some moderate sense of rotational invariance into the network; rotated samples of the original images were included as input samples. Some samples are shown below in Figure 3.5.

Figure 3.5: Rotated Yoruba vowel characters

- **Cropping of pattern occupied part of images**

The character occupied part of the filtered images were then cropped automatically so that a large portion of the background that contains no relevant information (pattern) are cut away. This process results in varying sizes of images after cropping.

Figure 3.6 shows cropped handwritten characters.

Figure 3.6: Cropped Yoruba vowel characters

- **Resizing of images**

As it is evident from the previous operation that images will be of different sizes (pixels), hence it will be required that all the images have the same size. Hence, the images were resized to 32×32 pixels. The downscaling of image sizes also has the effect of reducing the number of input neurons to the network, which is somewhat related to the number of hidden neurons that will be suitable enough to serve as long-term memory to the neural network system. This consequently reduces computational requirements on the whole system.

3.5 Network Output Design and Coding

Yoruba language has 7 vowel characters, hence the designed recognition system should have 7 classes, and therefore 7 output neurons. i.e. each neuron responds maximally to a character.

3.6 Neural Network Models

Several neural network architectures exist today, and the choice of a particular network model to be implemented in solving a particular task depend on the application; some neural network models lend themselves to some range specific tasks than others.

Neural networks have become a very important area of computational intelligence and machine learning over the years; the applications in other diverse fields, even outside of engineering is a testimony of their significance.

The most common application of neural networks in computing today is to perform one of these "easy-for-a human, difficult-for-a-machine" tasks, often referred to as pattern recognition (Shiffman, 2012).

Generally, neural networks can be implemented as a single layer or multilayer, the suitable number of layers has to be determined by the designer depending on application; also, the number of suitable neurons in each layer must be determined.

In the following sections, the four models of neural network architectures considered in this thesis will be further discussed. These architectures include Back Propagation Neural Networks, Convolutional Neural Networks (CNNs), and Denoising Auto Encoders (DAEs), Deep Belief Networks (DBNs).

3.6.1 Back propagation neural network (BPNN)

Back Propagation Neural Networks are perhaps the most used neural network models in practice, they are also known as feedforward networks. The name back propagation is derived from the manner in which learning is achieved. These networks use a supervised learning algorithm; training examples are supplied to the network with corresponding desired outputs. The actual outputs are computed during the forward pass of the network, errors are computed at the output layer and propagated back into the network for correction.

This process is repeated until the set error goal is attained or maximum number of epochs have been executed. After this, how well the network has learnt is obtained by simulating the network with some examples of the same task on which the network has been trained; usually, examples that were not supplied as part of training data can be used to see whether the trained network can cope with such data. The ability of the network to cope with these examples that were not part of training data or set is called the generalization power of the

network. This is done so as to ascertain that the trained network has not only memorized the training data, while it performs poorly on data that were not part of training set; a phenomenon referred to over-fitting.

The layers in between are referred to as hidden layers, as they are not directly observable (Günther and Fritsch, 2010).

The weights determine the function computed. Given an arbitrary number of hidden units, any Boolean function can be computed with a single hidden layer (Mooney, 2008).

Theoretically, back propagation neural networks can have a number of hidden layers. In practice, rarely is more than 1 hidden layer used and some researchers have even proved that the problem of saturation of hidden units (neurons) makes convergence difficult, especially when training weights are initialized randomly as done classically. On the other hand, the hard saturation at 0 may completely block the gradients and make optimization harder (Glorot and Bengio, 2010).

Some important features in multilayer networks are listed below.

- The hidden layer does intermediate computation before directing the input to the output layer.
- The input layer neurons are linked to the hidden layer neurons; the weights on these links are referred to as input-hidden layer weights.
- The hidden layer neurons and the corresponding weights are referred to as output-hidden layer weights (Chakraborty, 2010).

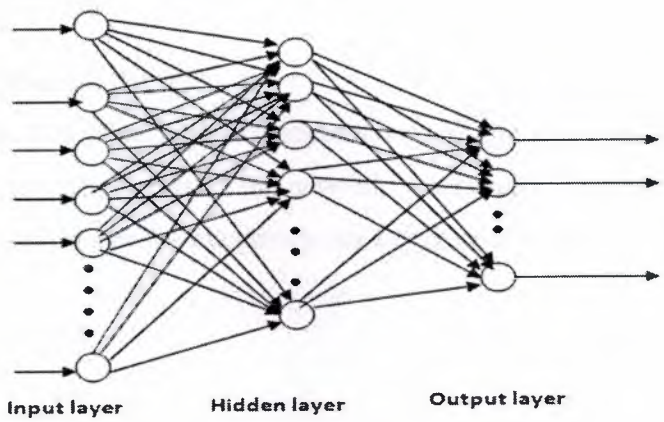


Figure 3.7: Back propagation neural network

The output layer requires linear separability. The purpose of the hidden layers is to make the problem linearly separable (Borga, 2011).

The training algorithm for back propagation networks as with other many networks is based on minimizing a cost function, in this case, the error between the desired output and actual output. The gradient descent approach, where the error surface is descended till the minimum point is reached is used; at this point, the weights of the network represent a mapping function of the input to the output.

Two common error computing functions used in neural networks are Least Mean Square (LMS) and Mean Square Error (MSE). The latter will be considered in this review of back propagation networks.

The gradient descent approach for minimizing the error cost function for back propagation networks come in two flavours as shown below.

$$e_j(n) = d_j(n) - y_j(n) \tag{3.1}$$

Where $d_j(n)$ and $y_j(n)$ are the desired and actual outputs of output neuron j at iteration n, $e_j(n)$ is the error of output neuron j at the nth iteration.

$$E_u(n) = \frac{1}{2}\sum_{j=1}^{k} e_j^2(n) \tag{3.2}$$

$E_u(n)$ is the sum of errors at the output layer when the u-th input pattern is given; or the accumulated errors of each individual output neuron, it is assumed in equation 13 that the output layer has k neurons.

3.6.1.2 Standard gradient descent

The standard gradient descent approach accumulates all the errors of the individual training patterns, then updates the weights of the network accordingly. This process is repeated until the desired error is reached or the maximum number of epochs has been executed.

$$E(\vec{w}) = \sum_{u=1}^{p} E_u(n) \tag{3.3}$$

Where, $E(\vec{w})$ is the accumulated error for all training samples p.

It is the aim of standard gradient method to then minimize $E(\vec{w})$ with respect to the weights of the network, mathematically

$$\nabla E(\vec{w}) = \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \frac{\partial E}{\partial w_2}, \ldots\ldots \frac{\partial E}{\partial w_i} \right] \tag{3.4}$$

This type of gradient descent is sometimes referred to as batch training and algorithm is supplied below.

1. Initial weights and thresholds to small random numbers.
2. Randomly choose an input pattern x(u)
3. Propagate the signal forward through the network
4. Compute δ_i^L in the output layer ($o_i = y_i^L$)

$$\delta_i^L = g'(h_i^L)[d_i^L - y_i^L] \tag{3.5}$$

Where h_i^L represents the net input to the ith unit in the lth layer, and g' is the derivative of the activation function g.

5. Compute the deltas for the preceding layers by propagating the error backwards.

$$\delta_i^l = g'(h_i^l)\sum_j w_{ij}^{l+1}\delta_j^{l+1} \tag{3.6}$$

For l = (L-1),.....,1.

6. Update weights using

$$\Delta w_{ij}^l = \eta \delta_i^l y_j^{l-1} \tag{3.7}$$

7. Go to step 2 and repeat for the next pattern until error in the output layer is below the a pre-specified threshold or maximum number of iterations is reached.

Where w_j are weights connected to neuron j, x_j are input patterns, d is the desired output, y is the actual output, t is iteration number, and η ($0.0 < \eta < 1.0$) is the learning rate or step size (Jain, Mao, Mohiuddin , 1996).

New weights update are carried out using the equation below

$$w_{new} \leftarrow w_{old} + \Delta w_{ij}^l \tag{3.8}$$

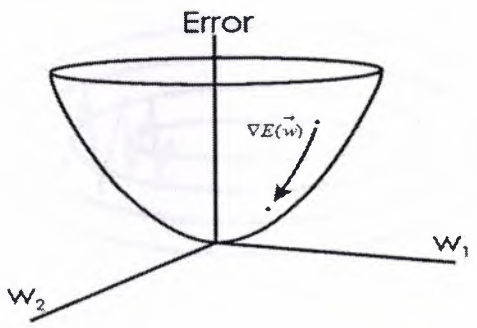


Figure 3.8: Standard gradient descent error surface (Leverington , 2009)

Figure.3.8 above shows the weight space of dimensionality two. i.e. w_1 and w_2. It is noteworthy that the weight space can be of far higher dimensionality.

The error surface is shown in the figure above, where each input batch forward pass outputs are compared against the desired outputs, errors gotten and then weights updated.

3.6.1.3 Stochastic gradient descent

In stochastic gradient method for error minimization, the actual output of the network is computed during the forward pass, when supplied with a particular pattern at the input, the computed output is compared against the desired output for that particular pattern, and then the error propagated back into the network so that weights can be updated. The main difference to the standard gradient descent method lie in that weights are updated after each pattern is supplied and error for that particular pattern is computed; as against standard gradient descent in which all input patterns outputs are processed as a batch, error computed for the batch, after which weights are updated. The idea behind stochastic gradient descent is to approximate this gradient descent search by updating weights incrementally, following the calculation of the error for each individual example.

Equations supporting the stochastic gradient descent are given below (Mitchell, 1997).

$$E_u(\vec{w}) = \frac{1}{2}\sum_{j=1}^{k} e_j^2(n) \tag{3.9}$$

$$\Delta w_{ij}^{l} = \eta e_j(n) y_j^{l-1} \tag{3.10}$$

Where, $E_u(\vec{w})$ is the sum of errors of j output neurons when the u-th input pattern is supplied.

Final weight updates are achieved using equation 3.8.

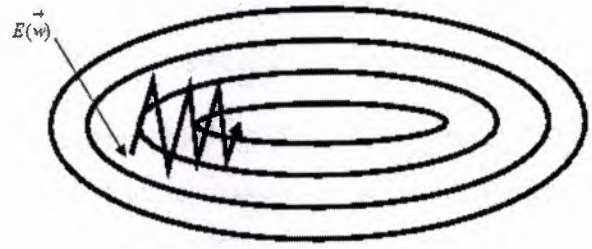


Figure 3.9: Stochastic gradient descent error surface (Yu, 2010)

Figure 3.9 shows a typical error minimization using the stochastic gradient descent approach. Since the stochastic algorithm does not need to remember which examples were visited during the previous iterations, it can process examples on the fly in a deployed system. In such a situation, the stochastic gradient descent directly optimizes the expected risk, since the examples are randomly drawn from the ground truth distribution (Bottou, 2012).

3.6.1.4 Local and global minima in gradient descent approach

One of the problems with the gradient descent approach for error minimization in back propagation networks is convergence. It has been seen in many literatures that the algorithm at times take so long to converge, and even when it does, may not converge to the point of least error on the error surface. This is the situation when the error surface has more than one minimum point; the lowest minimum on the error surface is referred to as the true or global minimum while the other minima points are known as local minima.

Figure 3.10 below exemplifies the problem, it can be seen that there exists various local minima, some better than the others. In any error surface, there is generally only one global minimum.

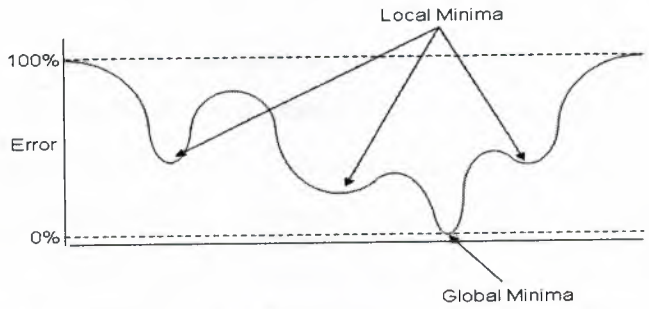


Figure 3.10: Local and global minima on error surface

From the figure above, it can be seen that gradient descent can be caught up in one of the local minima, which is not the true or global minimum.

Generally, to overcome the problem of the gradient descent approach being stuck in a local minimum, another term known as momentum, α, is added to the equations used to update the weights of the networks. Its value ranges between 1 and 0. i.e. $0 < \alpha < 1$.

The momentum term determines the effect of past weight changes on the current weight change (Bose, Liang, 1996).

The momentum term is meant to push the error past local minima, should the surface have local minima.

3.6.1.5 Issues with back propagation neural networks

1. Training data dimension: When back propagation networks are used in image classification, because of the large number of pixels involved, and therefore input neurons, the number of weights in the network becomes quite large (e.g. thousands).

Hence, such a large number of parameters increases the capacity of the system and therefore requires a larger training set (LeCun, Bottou, Bengio, and Haffner, 1998).

2. Translation and rotational Invariance: Back propagation networks trained with gradient descent algorithm suffer from translational variance, which is a recognition problem associated with moderate linear shifts of patterns in images. Generally, back propagation networks perform well on training data and classification tasks where images have been centred but perform relatively poor otherwise.

Alternatively, translational and rotational invariance can be built into these networks by including translated and rotated copies of the original patterns in the training set.

However, from the engineering perspective, invariance by training has two disadvantages. First, when a neural network has been trained to recognize an object in an invariant fashion with respect to known transformations, it is not obvious that this training will also enable the network to recognize other objects of different classes invariantly. Second, the computational demand imposed on the network may be too severe to cope with, especially if the dimensionality of the feature space is high (Haykins, 1999).

3. Scale invariance: Back propagation networks do have problems in recognition when trained networks are simulated with same patterns but of varying scales.

Moreover, gradient descent is not scale invariant in the parameters it seeks to optimize (Agrawal, 2012).

4. Illumination invariance: The problem of illumination variance arises when back propagation networks are trained, then simulated it same patterns but of varying illuminations. Since, back propagation networks are trained on pixel values as features for patterns, it is therefore evident that variation in pixel values due to illumination may lead to a false representation of patterns, and hence recognition may be affected.

5. Ignored input topology: This is a general problem with fully connected networks, and therefore back propagation networks in that local structure of inputs are rendered irrelevant.

The input variable can be presented in any (fixed) order without affecting the outcome of the training. On the contrary, images (or time-frequency representations of speech) have a strong 2D local structure: variables (or pixels) that are spatially or temporally nearby are highly correlated (LeCun, 1998).

Local correlations of pixels can be used to the advantage that local features are extracted and combined, hence objects recognition then achieved. i.e. a kind of bottom-top visual processing.

6. Vanishing gradient: Multilayer neural networks trained with back propagation do have the problem of vanishing gradient, where error gradients propagated back from higher to lower layers towards the input become exponentially decreasing and consequently learning becomes quite slow. This phenomenon is sometimes referred to saturation, because it is as a result of saturation of the activation functions of hidden units.

One common problem: saturation when the weighted sum is big, the output of the tanh (or sigmoid) saturates, and the gradient tends towards 0 (Bengio, 2003).

This problem had made the training of deep neural networks difficult in the past, but presently several approaches exist to resolve these problems when training deep networks.

3.6.2 Convolutional neural network (CNN)

Convolutional neural networks leverage on local feature extractions and combinations to overcome the above mentioned problems of back propagation networks.

Hierarchical models of the visual system are neural networks with a layered topology. In these networks, the receptive fields of units (i.e., the region of the visual space that units respond to) at one level of the hierarchy are constructed by combining inputs from units at a lower level. After a few processing stages, small receptive fields tuned to simple stimuli get combined to form larger receptive fields tuned to more complex stimuli (Serre, 2013).

The typical convolutional neural network consists of alternating convolution and sub-sampling layers, then the last layer is a fully connected network; typically a multilayer perceptron or classifier (e.g. back propagation network, radial basis function network, or support vector machine).

The first back convolutional neural network was presented by Yann LeCun et al, it was trained with back propagation and they applied it to the problem of handwritten digit recognition (LeCun, Boser, Denker, Henderson, Howard, Hubbard, and Jackel, 1990).

CNNs take translated versions of the same basis function, and "pool" over them to build translational invariant features. By sharing the same basis function across different image locations (weight-tying), CNNs have significantly fewer learnable parameters which make it possible to train them with fewer examples than if entirely different basis functions were learned at different locations (untied weights). Furthermore, CNNs naturally enjoy translational invariance, since this is hard-coded into the network architecture (Le et al., 2010).

Convolutional neural networks, fix some weights to be equal. In particular, they encode the 2d translational covariance, i.e filters applied in the top right patch will also be applied in the bottom left. Invariance and covariance are essential to the success of convolutional neural networks (Sauder, 2013).

Convolutional neural networks are apt for image applications, the extraction of local features and combinations to form higher feature objects makes them quite suitable.

Filters are used to extract some features from the data, in image processing, this could be edges, lines, corners, points etc. Some common filters used in image processing, hence can be

used in convolutional neural networks are Mexican hat filters, Gabor filters, Sobel, Canny etc. Note that filters are interchangeably used as kernels.

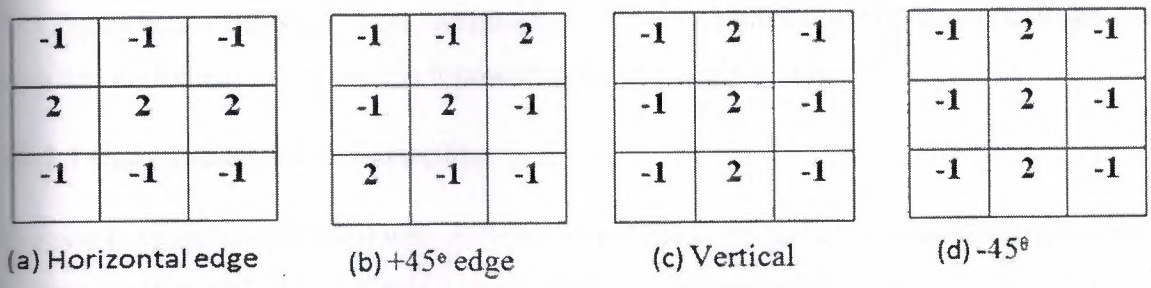


Figure 3.11: Line detection kernels (Gonzalez, Woods, and Eddins, 2004)

Figure 3.11 shows some kernels that can be used in extracting local features such as lines; (a), (b), (c), and (d) are horizontal , +45° , vertical, and -45° line detectors respectively.

These kernels are used to convolved the whole image so that localized features can be extracted. The size of the kernels used fixed and known as the receptive field.

Neuron units are arranged in subsequent layers in planes (2-dimensional), and all units in a particular plane share same set of weights (i.e. kernel weights). Units in a particular plane perform a specific operation on all regions of the input; a kernel or filter is used to convolve the whole image. i.e. each neuron is connected to the weights of the kernel.

Convolution operation is mathematically expressed below.

When two functions f(x) and v(x) are convolved, mathematically, the output

$$g(x) = \int_{-\infty}^{\infty} v(x')f(x-x')dx = v(x) * f(x) \tag{3.11}$$

When the v(x) is non-zero only across a finite interval $-n \leq v(x) \leq n$, and f(x) and v(x) are discrete, we can then re-write the equation above as shown below.

$$g(i) = \sum_{j=-n}^{n} v(j)f(i+j) \tag{3.12}$$

For two variable functions, equation can be re-expressed as,

$$g(i,j) = f(i,j) * v = \sum_{k=-n}^{n} \sum_{l=-n}^{n} v(k,l) f(i+k, j+l) \tag{3.13}$$

The figure below shows the basic architecture of a convolutional neural network, it will be seen that each convolution layer is followed by a sub-sampling layer.

3.6.2.1 Convolution and sub-sampling

The whole image is convolved with different kernel masks of size a×a (receptive field); when a particular kernel is used so that a particular local feature can be extracted, the output is contained in a respective feature plane. i.e. each unit or neuron in a particular feature plane has its weights as a×a. The number of different kernels of size a×a used will determine the number of different feature planes obtained. i.e. each feature plane contains a particular extracted local response to the filter. It is worth bearing in mind that during convolution, the kernel is not meant to fall outside the image at any rate, hence the sizes of feature planes are generally smaller than that of the image (in the no padding convolution). i.e. (i×i) < (K×L). During convolution, kernel masks generally overlap by a constant margin as it is shifted through the image.

It is remarkable that due to the fact units in each feature plane in C1 share same set of weights, there is a drastic reduction in the number of trainable weights, connections, and therefore overall training time for these networks as when compared to back propagation networks where all units' trainable weights are distinct.
The convolution, combination and implementation of feature maps can be achieved by using the relation given below.

$$x_j^l = f\left(\sum_{i \in M_j} x_i^{l-1} * k_{ij}^l + b_j^l\right) \tag{3.14}$$

Where j is the particular convolution feature map, M_j is a selection of input maps, k_{ij} is the convolution kernel, b_j is the bias of each feature map, l is the layer in the network, and f is the activation function.

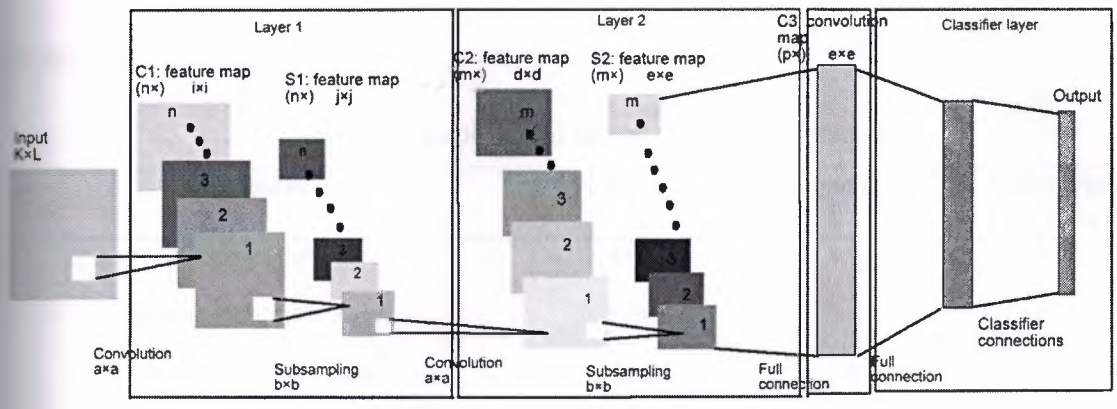


Figure 3.12: Convolutional neural network architecture

Figure 3.12 shows the convolutional neural network, with two 3 convolution layers and 2 sub-sampling layers. The last layer is a regular multi-layer network classifier. Each layer has grouped as composing a convolution layer and corresponding sub-sampling layer.

After feature map C1 has been obtained by convolving the input with the kernels, sub-sampling of the feature maps in C1 is then achieved by sweeping a mask of size j×j all over each feature plane in C1, hence, S2 has the same number of feature maps as C1. i.e. each feature map in C1 has a corresponding sub-sampling map in S1.

It is noteworthy that two approaches can be employed in the pooling operation for the sub-sampling layer S1; the average or max-pooling approach. The max-pooling used is described below.

Since each neuron in each S2 plane has j×j weights connected to the feature planes in C1, sub-sampling is achieved for neurons in each S2 plane by taking the maximum value of the inputs j×j, multiplied by a trainable coefficient, bias added, and passed through an activation function. It is to be noted that during sub-sampling, adjoining neurons' receptive field do not overlap as in the convolution from the input image to C1. The usefulness of sub-sampling includes reducing the resolution of the extracted local features in C1, damping of the response of the outputs to moderate equidistant translation and deformation. Since there is no overlap in during sub-sampling, it then follows that the size of feature maps in S1 is a factor 1/b of the feature maps in C1.

The table below shows the parameters of a typical convolutional neural network in Figure 3.12, and the relationship between each parameter and layers are further discussed below.

-Input image size: K×L

Table 3.1: Convolutional neural network parameters

Attributes	First layer	Second layer	Convolution map layer
Kernel size	a×a	a×a	N/A
Sub-sampling mask size	b×b	b×b	N/A
Number of feature maps	n	M	N/A
Each convolution feature map size	C1: i×i	C2: d×d	N/A
Each sub-sampling plane size	S1: j×j	S2: e×e	N/A
Number of convolution maps			p
Each convolution map feature plane size			e×e

It is also important to state that usually all the feature planes in C2 are not convolved with the feature planes in S1. Generally, a convolution table is developed to select which planes in S1 are convolved with which feature planes in C2. This technique greatly reduces again the number of connections in the network, and therefore trainable weights.

The application of the above approach forces different feature maps to extract different (hopefully complementary) features because they get different inputs; it also ensures that symmetry is broken in the network (LeCun, Bottou, Bengio, and Haffner, 1998).

The same process is repeated for layer 2 in figure 3.12; albeit that the last convolution layer, C3, has full connections to the sub-sampling layer, S2, and each plane is of the same size in S2, so that after convolution, each output would be a 1×1.

This hierarchical image processing has been adapted from the visual computation analogy found in the biological visual cortex.

Generally, the last layer in convolutional neural networks is the classifier layer; any suitable classifier can be adopted at this stage, but common options include back propagation networks, radial basis functions, support vector machine, etc. The number of neurons in the output layer of the classifier will be the number of patterns or objects for recognition.

Some mathematical equations relevant to these networks are provided below.

$$M_x^n = \frac{M_x^{n-1} - K_x^n}{S_x^n + 1} + 1 \tag{3.15}$$

$$M_y^n = \frac{M_y^{n-1} - K_y^n}{S_y^n + 1} + 1 \tag{3.16}$$

Where, (M_x^n , M_y^n) is the feature map size of each plane, (K_x^n, K_y^n) is the kernel size shifted over the valid input image region, (S_x^n, S_y^n) is the skipping factor of kernels in x and y-directions between subsequent convolutions, n indicates the layer. Each map in L^n is connected to at most M^{n-1} maps in layer L^{n-1} (Cires¸an, Meier, Masci, Gambardella, and Schmidhuber, 2011).

It will be noticed that in figure 3.12, square feature maps have basically been assumed, but the above equations can be used in any general case; also, the number of layers in the network may vary according to application and design.

At times, a contrast normalization scheme is implemented in the convolution map layer which significantly improves the network performance.

3.6.3 Deep learning

Deep learning is a response to some of the issues encountered in back propagation networks as mentioned in sections 3.6.1.5. These networks are 'basically' multilayer networks, but differ from the classical back propagation in the analogy in which training is achieved.
Deep learning is a term that has been in use recently, meaning training feedforward networks of more than one hidden layer; features are learnt in a hierarchical way from the input image to the classifier, each layer extracts more meaningful features from the previous layer. Also, this architecture allows sharing low level representation.

The problem of feedforward networks getting stuck in local minima is another situation generally as a result random initialization of weights in these networks. Some sense of features correlation or prior knowledge can be built into the hidden layers of these networks by supervised or unsupervised pre-training schemes.

3.6.3.1 Auto encoder (AE)

An auto encoder is basically a feedforward network that accepts the inputs as corresponding target outputs. They are used to learn compressed encoding of the training data. i.e. as

implemented when feedfoward networks are used in image compression (See figure 3.13 & 3.14 below).

Generally, many auto encoders can be layered on top of each other, in which case it referred to as stacked auto encoders. The approach to training this type of architecture build up is known as greedy layer-wise training because each hidden layer is "hand picked" for pre-training of weights; each layer is trained as in single hidden layer feedforward networks.

Since, an auto-encoder is an unsupervised learning scheme, we can leverage on the availability of large unlabelled data for the pre-training, then use the available labelled data for fine tuning the whole network.

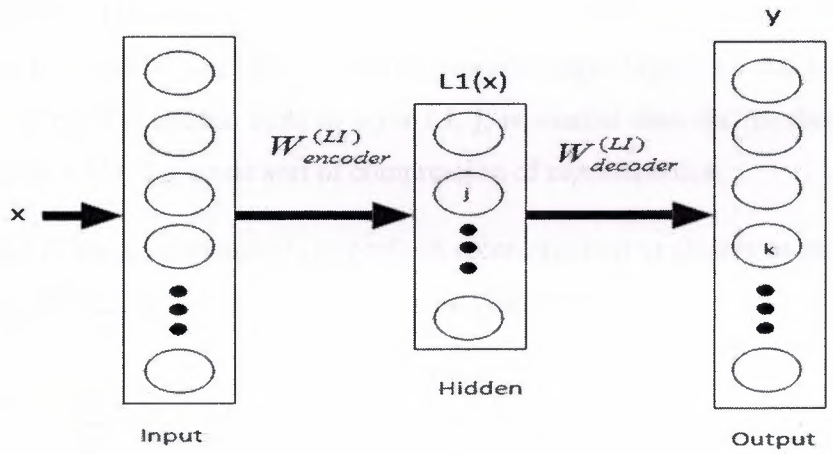


Figure 3.13: Auto encoder

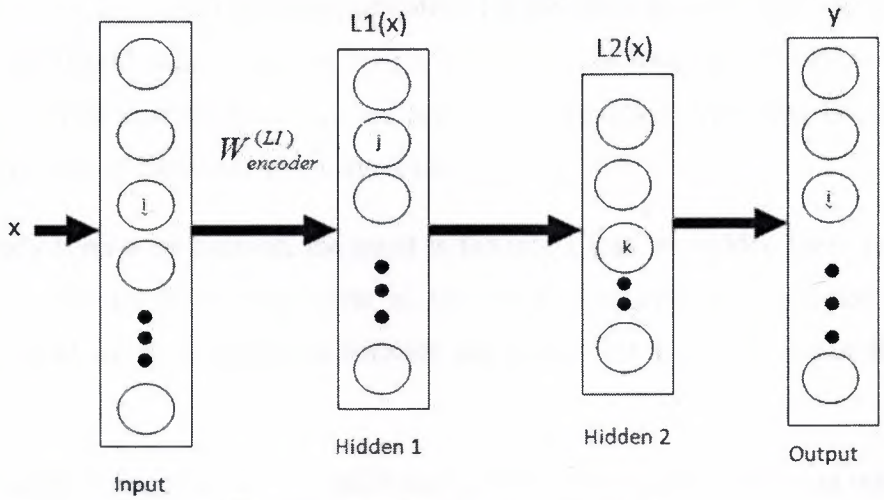


Figure 3.14: Stacked auto encoder

The auto encoder can be seen as an encoder-decoder system, where the encoder (input-hidden layer pair) receives the input, extracting essential features for reconstruction; while the decoder (hidden-output layer pair) part receives the features extracted from the hidden layer, performing reconstruction at its best (figure 3.13).

Since the auto encoder is basically a feedforward network, it can be shown that,

$$L1(x) = g(m(x)) = sigm(b^{(L1)} + W^{(L1)}_{encoder}x) \tag{3.17}$$

$$y = z(n(x)) = sigm(b^{(y)} + W^{(L1)}_{decoder}L1(x)) \tag{3.18}$$

Where, m(x) and n(x) are the pre-activations of the hidden and output layers L1 and y respectively; $b^{(L1)}$ and $b^{(y)}$ are biases of the hidden and output layers, L1 and y respectively. Generally, the number hidden units in layer L1, j, is smaller than the number of units at the input, k (figure 3.13) . i.e. some sort of compression of representation.

The objective of the auto encoder is to perform reconstruction as cleanly as possible, which is achieved by minimizing a cost functions such as given below

$$C(x, y) = \sum_{1}^{k} (y_k - x_k)^2 \tag{3.19}$$

$$C(x, y) = -\sum_{1}^{k} (x_k \log(y_k) + (1 - x_k)\log(1 - y_k)) \tag{3.20}$$

Equation 3.19 is used when the range of values for the input are real, and a linear activation applied at the output; while equation 3.20 is used when the inputs are binary or fall into the range 0 to 1, and sigmoid functions are applied as activation functions. Equation 3.19 is known as the sum of Bernoulli cross-entropies.

In the greedy layer-wise training, the input is fed into L1 as the hidden layer and L2 as the output; note that L2 have target data as the input. The network is trained as in back propagation and weights connection between the input layer and L1 saved or fixed (Figure 3.14).

The input layer is removed and L1 made the input, L2 the hidden layer, and output follows last. The activation values of L1 acts as now input to the hidden layer L2, and the output layer made the same as the training data, weights between L1 and L2 are trained and saved.

Finally, the pre-trained weights obtained from the greedy layer-wise training are coupled back to the corresponding units in the network so that final weights fine-tuning for the whole network can now be carried out using back propagation algorithm. i.e. the original training data is supplied at the input layer and the corresponding target outputs or class labels are supplied at the output layer. Note that the weights between the last hidden layer and the output network is randomly initialized as usual before final network fine-tuning or maybe discriminately pre-trained.

Another variant of auto encoders, known as denoising auto encoders which are very similar to the typical auto encoder, save that the input data is intentionally corrupted by some moderate degree (setting some random input data attributes to 0) and the targets are correct, unaltered data. Here, the denoising auto encoder is required to learn the reconstruction of corrupt input data; this greatly improves the performance of initialized weights for deep networks. Note that stacked auto encoders belong to the unsupervised learning pre-training approach in deep learning. i.e. it is a generative model.

3.6.3.2 Deep belief network (DBN)

This learning scheme allows these networks to have weights that are initialized with some level of correlation to the task the overall designed network is to learn.

These networks are built on unsupervised learning weights pre-training approach, and the basic units found in such networks are Restricted Boltzmann Machines (RBMs).

Restricted Boltzmann machines have visible and hidden layers, with undirected connections. The input layer is referred to as the visible layer and computation can proceed in either visible to hidden layer or hidden to visible layer; note that there is full connection between all units in both layers.

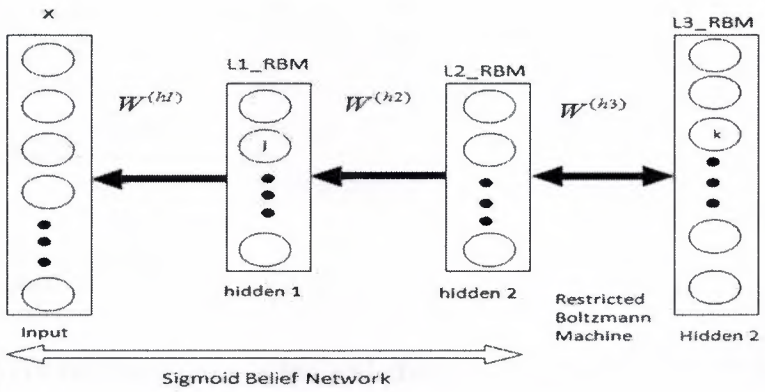


Figure 3.15: Deep Belief Network

A restricted Boltzmann machine has only two layers (figure 3.16); the input (visible) and the hidden layer. The connections between the two layers are undirected, and there are no interconnections between units of the same layer as in the general Boltzmann machine. We can therefore say that from the restriction in the interconnections of units between layers, units are conditionally independent.

The RBM can seen as a Markov network, where the visible layer consists of either Bernoulli (binary) or Gaussian (real values usually between from 0 to 1) stochastic units, and the hidden layer of stochastic Bernoulli units (Deng and Yu, 2013).
Figure 3.16 below shows an RBM, the backbone of DBNs.

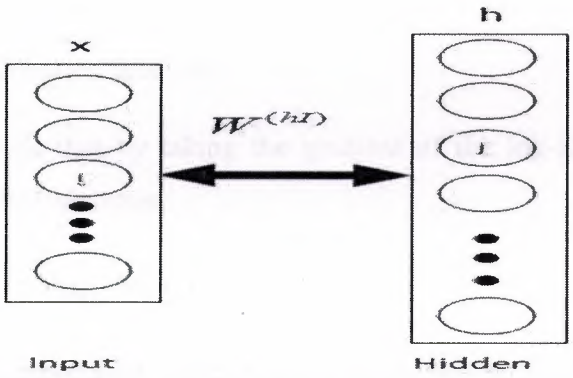


Figure 3.16: Restricted Boltzmann Machine

The main aim of a RBM is to compute the joint distribution of v and h, p(v,h), given some model specific parameter, ϕ.
This joint distribution can be described using an energy based probabilistic function as shown below.

$$E(x,h;\phi) = -\sum_i \sum_j W_{ij} x_i h_j - \sum_i b_i x_i - \sum_j b_j h_j \tag{3.21}$$

$$p(x,h;\phi) = \frac{e^{(-E(x,h;\phi))}}{Z} \tag{3.22}$$

$$Z = \sum_i \sum_h e^{(-E(x,h;\phi))} \tag{3.23}$$

Where, $E(x,h;\phi)$ is the energy associated with the distribution of x given h; x and h are input and hidden units activations respectively, i is the number of units at the input layer, j is the

number of units at the hidden layer, b_i is the corresponding bias to the input layer units, b_j is the corresponding bias to the hidden layer units, W_{ij} is the weight connection between unit x_i and h_j, $P(x,h;\phi)$ is the joint distribution of variable x and h, while Z is a partition constant or a normalization factor (Li Deng and Dong Yu) [45] (Yoshua Bengio et al., 2007).

For a RBM with binary stochastic variables at both visible and hidden layers, the conditional probabilities of a unit, given the vector of units variables of the other layer can be written as,

$$p(h_j = 1 \mid v;\phi) = \sigma(\sum_i W_{ij} x_i + b_j) \tag{3.24}$$

$$p(x_i = 1 \mid h;\phi) = \sigma(\sum_j W_{ij} h_j + b_i) \tag{3.25}$$

Where σ is the sigmoid activation function.

Deng observed in his work that by taking the gradient of the log-likelihood $p(x; \phi)$, the weight update rule for RBM becomes,

$$\Delta W_{ij} = E_{data}(x_i h_j) - E_{model}(x_i h_j) \tag{3.26}$$

Where, E_{data} is the actual expectation when h_j is sampled from x, given the training set; and E_{model} is the expectation of h_j from x, considering the distribution defined by the model.

It has also been shown that the computation of such likelihood maximization, E_{model}, is intractable in the training of RBMs, hence the use of an approximation scheme known as "contrastive divergence", an algorithm proposed to solve the problem of intractability of E_{model} by Hinton (Hinton, Osindero, and Teh, 2006).

Because of the way it is learned, the graphical model has the convenient property that the top-down generative weights can be used in the opposite direction for performing inference in a single bottom-up pass (Mohamed, Hinton, and Penn, 2012).

Hence, such an attribute as mentioned above makes feasible the use of an algorithm like back propagation in the fine-tuning or optimization of the pre-trained network for discriminative purpose.

Recently, the contrastive divergence algorithm was developed to train these networks, which is described in details below.

-Positive phase:

- An input sample v is clamped to the input layer.
- v is propagated to the hidden layer in a similar manner to the feedforward networks. The result of the hidden layer activations is h.

- Negative phase:

- Propagate h back to the visible layer with result v' (the connections between the visible and hidden layers are undirected and thus allow movement in both directions).
- Propagate the new v' back to the hidden layer with activations result h'.

- Weight update:

$$w(t+1) = w(t) + a(vh^T - v'h'^T) \tag{3.27}$$

Where α is the learning rate and v, v', h, h', and w are vectors (Vasilev).

The process of sweeping v and h between visible and hidden layers is repeated until there is a satisfactory reconstruction of the input that sample v is significantly close to h. i.e. Gibbs sampling. It will be seen from the above figure that the deep belief network is basically a stacked RBMs. Each layer is trained greedily; the input acts as the visible layer, input data are propagated between L1_RBM and the input layer (positive and negative phases) as described above under contrastive divergence algorithm. After the weights have been updated satisfactorily, input layer is decoupled and weights fixed. Then L1_RBM becomes the visible layer (input layer) to L2_RBM; the weights between L1_RBM and L2_RBM are updated using the contrastive divergence algorithm again. This process is repeated for all the hidden layers of the network, as this initializes the network's weights to values that give a significant overall network performance after fine-tuning using a supervised learning algorithm. Generally, the whole network's weights are fine-tuned using the typical back propagation algorithm. It is to be noted that there are no connections between units in the visible-visible layer or hidden-hidden layer.

3.7 Summary

This chapter gives good and strong technical insight into the processing of images that were used in this research; also, the neural network architectures that have been considered were briefly and sufficiently discussed.

The convectional back propagation neural network was discussed in details, with the learning algorithm. Furthermore, the problems associated with BPNN networks were presented and analysed. Convolutional networks, and its structure which simulates the biological visual perception were also discussed briefly. Deep networks were also discussed in view of the different architectures and pre-training schemes obtainable.

In all, this chapter presents a thorough technical background and insight into the designs implemented in the following chapter; lastly, it articulates the whole essence of the thesis purpose involving some image processing work and neural networks as classifiers.

CHAPTER FOUR
TRAINING OF NEURAL NETWORKS

4.1 Overview

As it is the aim of this research to investigate pattern invariance and the response or tolerance of some neural network models presented in chapter 3; the different designed networks will be trained on a training set, and validated using another data set.

Furthermore, in order to investigate the level of pattern invariance learned by each model, the designed systems were simulated with rotated, translated, scale varied, and noisy images.

The remaining sections of this chapter present the different neural network architectures, training parameters and considerations to choosing such parameters.

4.2 Neural Networks

The different neural network architectures considered in this thesis are presented in this section, with their corresponding training schemes and parameters which have been obtained heuristically during training. All the networks were trained, validated, and tested on the same set of data, hence there is clarity of comparison of training requirements in this chapter, and test results in the next chapter.

A. DATABASES A1

These database A1 contain the training samples for the different network architectures considered for this research. The characters in this database, A1, have been sufficiently processed with the key interest being that images are now centred in the images i.e. most redundant background pixels removed.

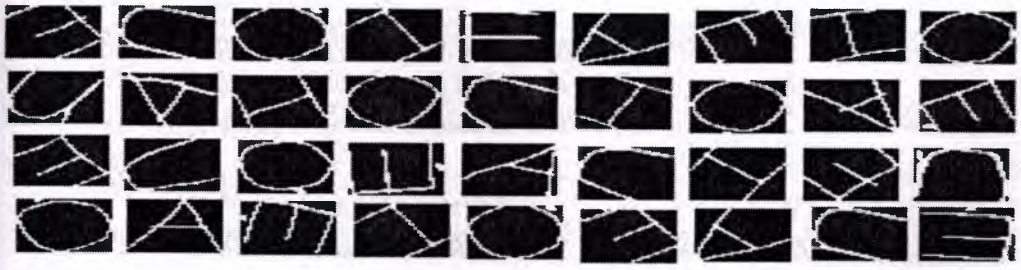


Figure 4.1: Training database characters

4.2.1 Back propagation neural network model

The final processed input images are all of 32 ×32 pixels (1024 pixels), it therefore follows that the number of input neurons is 1024. The suitable number of hidden neurons cannot be determined at this stage of the network design, but will be obtained heuristically during training. The number of different characters to be recognized, 7, therefore necessitates the number of output neurons to be 7.

The table below shows the training parameters used in training the back propagation network.

Table 4.1: Heuristic training of BPNN-1

Number of training samples	**14, 000**
Number of hidden neurons	65
Activation function at hidden and output layers	Sigmoid
Learning rate (η)	0.045
Momentum rate (α)	0.72
Epochs	1600
Training time (seconds)	502
Mean Square Error (MSE)	0.1120

A validation set of 2,500 samples was used to control the trained back propagation network from over-fitting data. i.e. memorizing data and hence losing generalization power.

Hence, the gradient algorithm with learning and momentum rate has been used in the training of the feedforward networks.

To further observed the performance of BPNN, a network of 2 hidden layers was also designed and the training parameters are shown below.

Table 4.2: Heuristic training of BPNN- 2

Number of training samples	**14, 000**
Number of neurons in hidden layer 1	95
Number of neurons in hidden layer 2	65
Activation function at hidden and output layers	Sigmoid
Learning rate (η)	0.082
Momentum rate (α)	0.65
Epochs	2000
Training time (seconds)	669

K-fold cross validation has been used during training, such that the error on the validation set is also monitored and in the advent that the error on the validation set increases for a specified number of epochs, training is stopped to avoid over-fitting.

4.2.2 Convolutional network neural model

The input image sizes are 32×32, and a kernel or receptive field of 5×5 pixels was used in the first convolution layer to extract local features; and 6 feature maps were extracted. The number of pixels (units) for each feature map is calculated below using Equations 4.1 & 4.2.

$$M_x^n = \frac{M_x^{n-1} - K_x^n}{S_x^n + 1} + 1 \quad , \quad C_x^1 = \frac{32-5}{0+1} + 1 = 28 \tag{4.1}$$

$$M_y^n = \frac{M_y^{n-1} - K_y^n}{S_y^n + 1} + 1 \quad , \quad C_y^1 = \frac{32-5}{0+1} + 1 = 28 \tag{4.2}$$

Therefore the size of feature maps in C1 is 28×28. The table below shows the full parameters for the network training.

Table 4.3: CNN units and feature maps

Attributes	First layer	Second layer	Convolution map layer
Kernel size	5×5	5×5	N/A
Sub-sampling mask size	2×2	2×2	N/A
Number of feature maps	6	12	N/A
Each convolution feature map size	C1: 28×28	C2: 10×10	N/A
Each sub-sampling plane size	S1: 14×14	S2: 5×5	N/A
Number of convolution maps			12
Each convolution map feature plane size			5×5

Since the C1 is sub-sampled using a mask of 2×2 and there is no overlap of regions during sub-sampled, it then follows that the size of each feature map in S2 is calculated below as:

$$S_x^1 = \frac{C_x^1}{b} = \frac{28}{2} = 14, \qquad S_y^1 = \frac{C_y^1}{b} = \frac{28}{2} = 14 \tag{4.3}$$

Therefore the size of each feature map in the sub-sampling layer 1, S1, is 14×14.

Size of convolution feature maps for layer 2 can be determined using the above formulas again as shown below.

$$C_x^2 = \frac{14-5}{0+1}+1=10 \qquad C_y^2 = \frac{14-5}{0+1}+1=10 \tag{4.4}$$

,

Hence, the size of each feature map in the convolution layer 2, C2, is 10×10.

The convolution layer 2 is then again down-sampled using the same mask size as in layer 1. i.e. 2×2.

$$S_x^2 = \frac{C_x^2}{b} = \frac{10}{2} = 5, \qquad S_y^2 = \frac{C_y^2}{b} = \frac{10}{2} = 5 \tag{4.5}$$

The number of feature maps at the convolution map layer (the layer just before the classifier layer) is 12, and of the same size as the last before it (i.e. 5×5), so that each convolution results in a single value.

Table 4.4: Training parameters for CNN

Number of training samples	**14, 000**
Activation function at hidden and output layers	Tanh
Learning rate (η)	0.8
Epochs	4500
Training time (seconds)	798
Mean Square Error (MSE)	0.1333

4.2.3 Denoising auto encoder model

This model is more or less the multilayer neural network, with the basic difference in the way weights are initialized for the network. A single hidden layer network was designed, therefore, the weights available for pre-training are:

- weights between the input and hidden layer.
- weights between the hidden and output layer.

The number of input neurons remains 1024, the number of input pixels as in the case of back propagation network, the number of sufficient hidden neurons to encode the inputs was chosen as 100, and number of output neurons remain 7, the number of desired output classes. The hidden layer was pre-trained as discussed in section 3.632 until the level of reconstruction of the input was found to be significantly suitable.

Table 4.5: Pre-training parameters for DAE

Number of training samples	**14, 000**
Number of hidden neurons	100
Input Zero Masked Fraction	0.5
Activation function at hidden and output layers	Sigmoid
Learning rate (η)	0.8
Epochs	10

The auto encoder was first trained by using inputs as corresponding targets, and the error of reconstruction noted until it became significantly low. i.e. hidden layer now has the capability to reconstruct inputs with significant performance. The pre-trained network now has weights initialized to values that favourable for better learning when it is coupled all up and trained as a back propagation network. i.e. the target outputs are now corresponding input labels. The table below, Table 4.6, shows the parameters used to fine-tune the pre-trained network.

Table 4.6: BPNN Parameters to fine-tuning DAE

Number of training samples	**14, 000**
Activation function at hidden and output layers	Sigmoid
Learning rate (η)	0.7
Epochs	500
Training time (seconds)	250
Mean Square Error (MSE)	0.1006

4.2.4 Stacked denoising auto encoder model

Table 4.7: Pre-training parameters for SDAE

Number of training samples	**14, 000**
Number of neurons in hidden layer 1	95
Number of neurons in hidden layer 2	65
Input Zero Masked Fraction	0.5
Activation function at hidden and output layers	Sigmoid
Learning rate (η)	0.8
Epochs	10
Training time (seconds)	118

In order to observe how distributed knowledge representation developed in hidden layers may affect performance of networks, a 2 hidden layer denoising auto encoder was also designed.

It should be noted that the 2 hidden layer auto encoder will be referred to a Stacked Denoising Auto Encoder (SDAE) as is standard and common practice; the training parameters are shown in the table below.

Table 4.8: BPNN Parameters to fine-tuning SDAE

Number of training samples	**14, 000**
Activation function at hidden and output layers	Sigmoid
Learning rate (η)	0.7
Epochs	400
Training time (seconds)	377
Mean Square Error (MSE)	0.1046

4.2.5 Deep belief network model

The deep model was designed with two hidden layers which were pre-trained as discussed in section 3.633. The hidden layers can be seen as stacked restricted Boltzmann machines, and the pre-training of weights was achieved using the contrastive divergence algorithm. The number of input neurons in the visible layer (first layer) is 1024, the input pixel size; number of neurons in the first hidden layer is 200, number in the second hidden layer is 150, while the number of output neurons remains 7.

Table 4.9: Pre-training parameters for DBN

Number of training samples	**14, 000**
Number of neurons in first hidden layer	200
Number of neurons in second hidden layer	150
Activation function at hidden and output layers	Sigmoid
Learning rate (η)	0.2
Epochs	5
Training time for hidden layer, L1 (seconds)	95
Training time for hidden layer, L2 (seconds)	103

The pre-trained weights were used to initialize the designed feedforward neural network, which is now favourable to converge faster to a better local minimum.

Table 4.10: BPNN Parameters to fine-tuning DBN

Number of training samples	**14, 000**
Activation function at hidden and output layers	Sigmoid
Learning rate (η)	0.6
Epochs	1000
Training time (seconds)	170
Mean Square Error (MSE)	0.1024

4.3 Summary

In this chapter, the trainings of the neural networks considered in this research have been presented, focusing on training parameters such as the number of required epochs, training time, and the achieved mean square error (MSE). It will be seen that the back propagation networks have the highest average training time compared to all other networks, after which comes convolutional networks. The large time required to train the BPNN models can be associated with some of the problems discussed in chapter three; problems such as saturating units, vanishing gradients in BPNN2, and the weights generally starting far away from the weights space that is favourable for fast convergence to a good local minimum. The convolutional network computational requirement and time is obvious in view of the convolutional and pooling operations that had to be achieved in the forward and backward pass of training data and error gradients respectively; with more convolution and pooling layers added to the network, training time may grow exponentially.

It will also been seen that the deep belief network and denoising auto encoders have the lowest MSE after training; this can be considered as a result of the pre-training of the networks, favouring the networks' weights starting out at a weight spaces that aids fast convergence, and to a better local minimum.

CHAPTER FIVE
RESULTS AND DISCUSSION

5.1 Overview

This section shows the validation and testing results of the models discussed in chapter 4; in this research work, processed databases of size 32×32 were used to observe the performance of the models. Each database for testing contains images with a particular variance of interest on which the tolerance of network models is to be observed.

5.2 Learning Curve and Validation of Network Models

The learning curves of the networks described in chapter four are presented here; the validation databases with which the different constraints have been now applied on were used to simulate the trained networks, the error rates achieved were then analysed and discussed accordingly.

5.2.1 BPNN model

The back propagation models were trained as discussed in chapter 4, and the MSE (Mean Square Error) plot is shown below (figure 5.1 (a) & (b)) . It will be seen that even though the training the train error (blue curve) kept decreasing, validation error (green curve) had stop decreasing, likewise the test error (red curve), hence training was stopped to prevent over-fitting of training data. Six validation checks were used in the training. i.e. the number of iterations such that the training error kept decreasing, but the validation error was increasing.

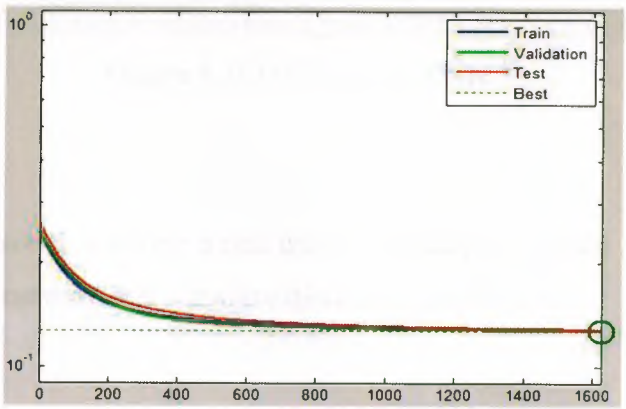


Figure 5.1(a): MSE plot for BPNN1

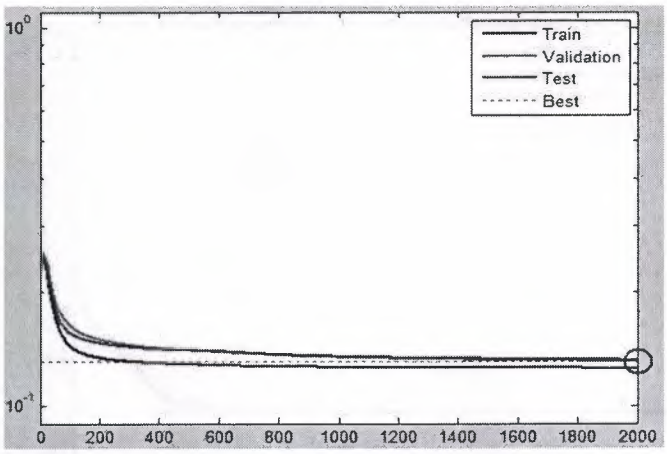


Figure 5.1(b): MSE plot for BPNN2

The learning curve for the BPNN2 (BPNN network with 2 hidden layers) is shown in Figure 5.1(b)

5.2.2 CNN model

The Convolution network (CNN) whose parameters were described in chapter 4 was trained and the learning curve is shown below in Figure 5.2.

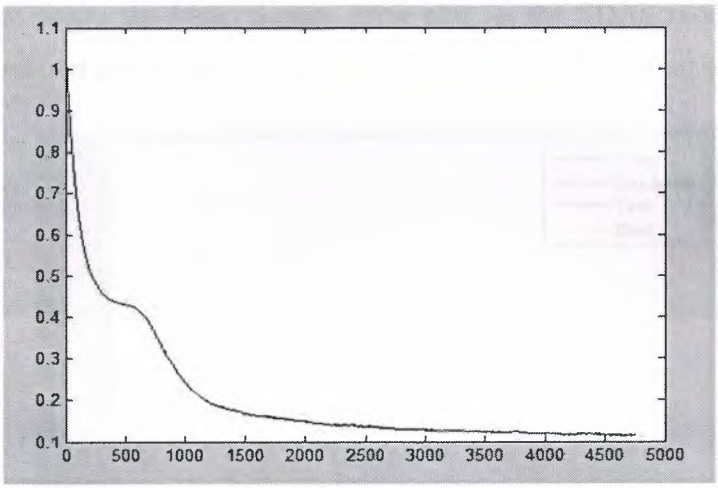


Figure 5.2: MSE plot for CNN

5.2.3.1 DAE model

The denoising auto encoder was fine tuned using the backpropagation algorithm to further reduce the error on the network and introduce discriminative feature.

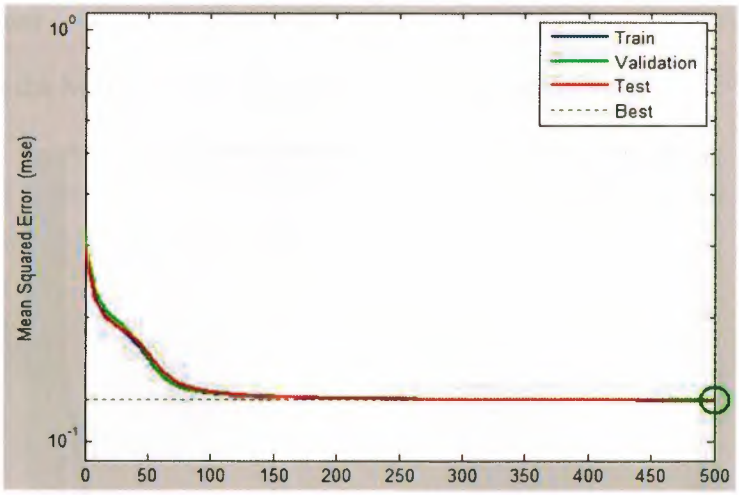


Figure 5.3: MSE plot for DAE fine-tuning

It will be seen in the above figure (Figure 5.3) that the train, validation, and test curves lie on one another indicating that validation did not stop before earlier than optimum learning was achieved.

5.2.3.1 SDAE model

Figure 5.4 below shows the Mean Square Error plot for the SDAE model, 2 hidden layers were implemented in the network, and the same denoising mask fraction of 0.5 was also used.

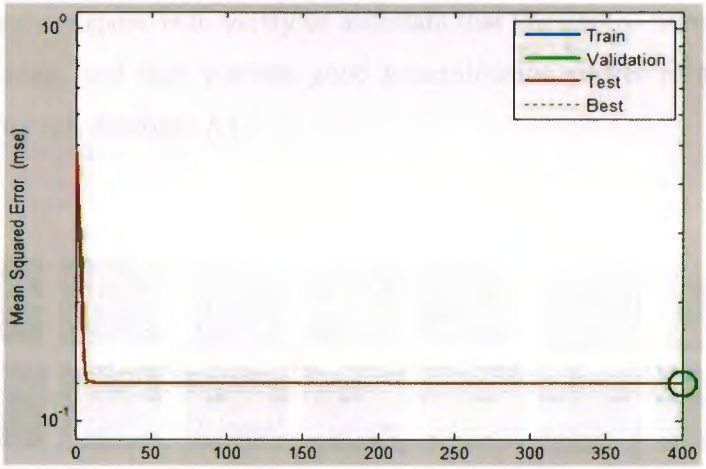


Figure 5.4: SDAE MSE plot for fine-tuning

The deep network was fine-tuned in using the backpropagation algorithm in order that the network can be used for discriminate tasks.

5.2.4 DBN model

Figure 5.5 shows the MSE plot for fine-tuning the DBN respectively.

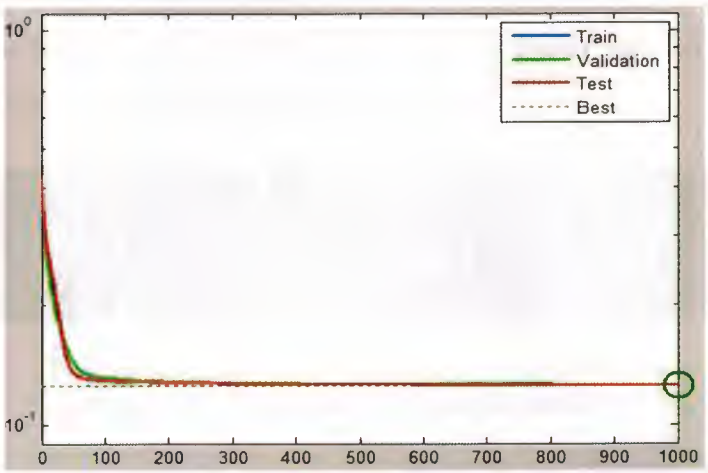


Figure 5.5: DBN MSE plot for fine-tuning

5.3 Databases for Testing/Simulating Trained Networks

A. DATABASE A2

This database contains sample images of the seven vowel characters as in the training database, and its sole purpose is to verify or ascertain that the trained networks did not over-fitting during training, and thus possess good generalization power to samples of images found in the training set, database A1.

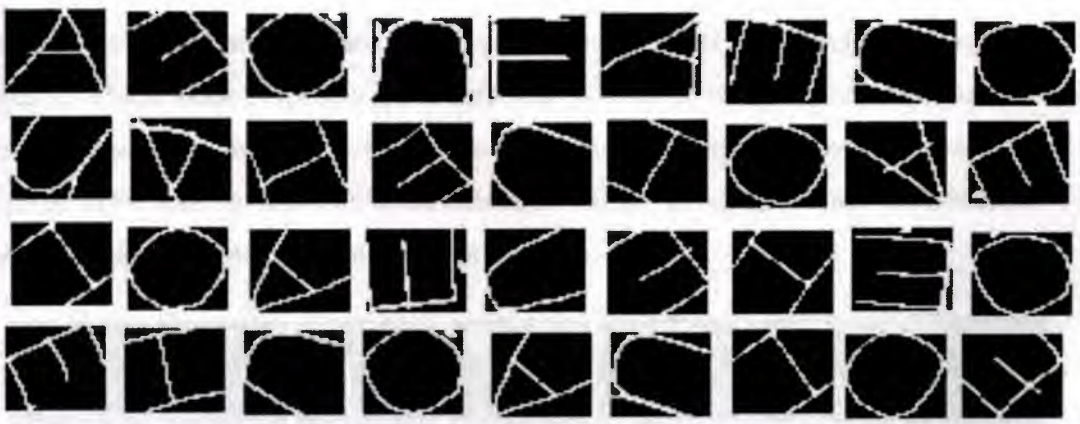


Figure 5.6: Validation database characters

B. DATABASE A3

For the purpose of evaluating the tolerance of the trained networks to translation a separate database was collected with the same characters and other feature characteristics in databases A1 and A2 save that the characters in the images have now been translated horizontally and vertically. The figure below describes these translations.

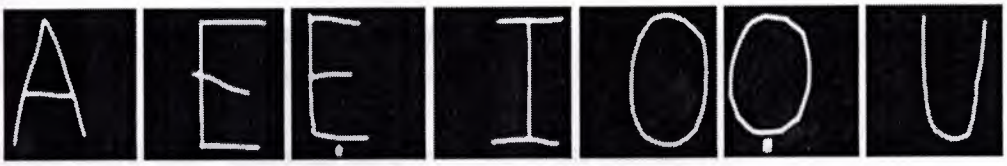


Figure 5.7: Translated database characters

C. DATABASE A4

This database contains the rotated characters contained in database A1 and A2. Its sole purpose is to further evaluate the performance of trained networks on pattern rotation. See Figure 5.8 for samples.

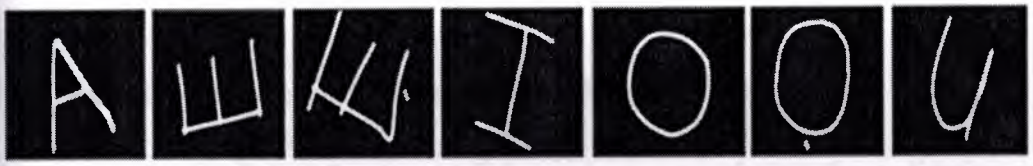


Figure 5.8: Rotated database characters

D. *DATABASE A5*

This database is essentially databases A1 and A2 except that the scales of characters in the images have now been purposely made different in order to evaluate the performance of the networks on scale mismatch. It will be seen that some characters are now bigger or smaller as compared to the training and validation characters earlier shown in chapter 4, Figure 4.1. Figure 5.9 shows samples contained in this database.

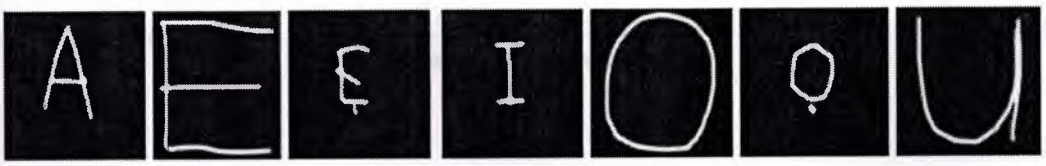


Figure 5.9: Scale varied database characters

E. DATABASE A6

In order to assess the performance of the networks on noisy data, sub-databases with added salt & pepper noise of different densities were collected as described below.

Database A6_1: 2.5% noise density
Database A6_2: 5% noise density
Database A6_3: 10% noise density
Database A6_4: 20% noise density
Database A7_5: 30% noise density

The figure below show character samples of database A6_4.

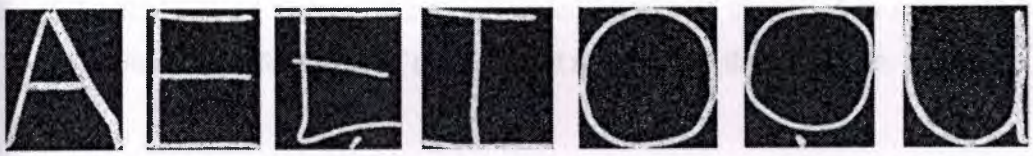


Figure 5.10: Database A6_4: characters with 20% salt & pepper noise density

5.4 Test Performances of Trained Network Models on Databases

This section details the performance of the trained networks on the validation and test databases as described in section 5.3. The trained networks were supplied with the databases, so that the corresponding classes of each image can be simulated.

14,000 samples were used as training set, 2500 samples as validation set, and 700 samples as test set on the variances.

The definition of recognition rates for the simulation or testing of database is given below.

Recognition rate can be obtained from error rate using the equations provided below.

$$\text{Recognition rate} = \frac{\textit{Number of correctly classified samples}}{\textit{Total number of test samples}} \tag{5.1}$$

Or

$$\textit{Error rate} = 1 - \text{Recognition rate} \tag{5.2}$$

Table 5.1: Recognition rates for training and validation data

Network models	Training data (14,000)	Validation data (2,500)
BPNN – 1 layer	95.67%	92.66%
BPNN – 2 layers	97.23%	93.61%
ConvNet	98.34%	97.98%
DAE – 1 layer	99.53%	93.21%
SDAE – 2 layers	99.72%	94.33%
DBN - 2 layers	99.77%	96.23%

The trained networks were firstly simulated on the training data and the recognition rates achieved are can be seen in table 5.1; also, the networks simulated on databases which contain translated, rotated images, and Scale varied images as described in section 5.3. The table below shows the performance of the different networks to the variances.

Table 5.2: Recognition rates for network architectures on variances

Network models	Translation (700)	Rotation (700)	Scale (700)
BPNN – 1 layer	14.29%	68.6%	64.29%
BPNN – 2 layers	17.14%	72.71%	63.42%
ConvNet	32.86%	86.29%	72.00%
DAE – 1 layer	20.00%	75.14%	69.43%
SDAE – 2 layers	25.71%	77.86%	72.57%
DBN - 2 layers	18.57%	80.14%	76.71%

The networks were also simulated on different levels of noise added to the images. i.e. database A6. The table below shows the recognition rates obtained.

Table 5.3: Recognition rates for networks on various noise densities

Network models	2.5%	5%	10%	20%	30%
BPNN – 1 layer	67.86%	65.14%	65.29%	58.71%	51.71%
BPNN – 2 layers	70.57%	68.42%	65.86%	55.14%	44.14%
ConvNet	83.57%	81.43%	79.86%	71.43%	68.29%
DAE – 1 layer	71.57%	67.57%	56.43%	40.14%	31.57%
SDAE – 2 layers	74.14%	69.57%	61.43%	46.14%	36.29%
DBN - 2 layers	76.29%	72.29%	61.57%	35.86%	22.71%

The figure below shows the performance of the different network architectures against 0%, 2.5%, 5%, 10%, 20%, and 30% noise densities. i.e. Figure 5.11.

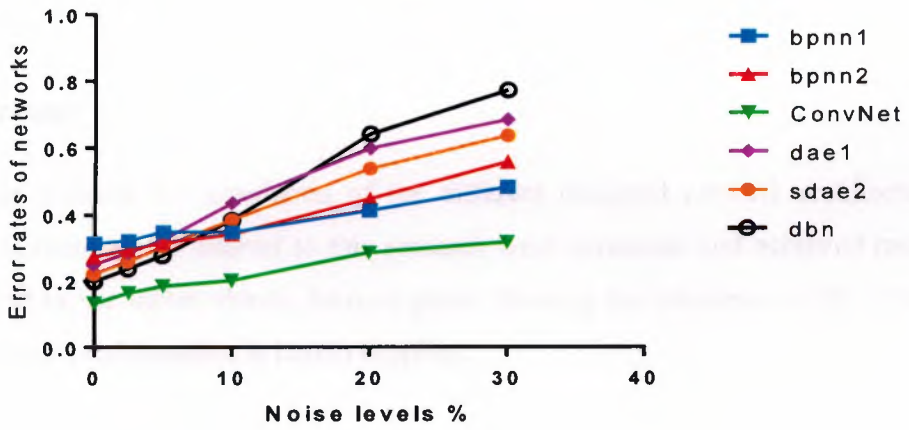


Figure 5.11: Performance of networks on various noise levels

5.5 Discussion of results

This research is meant to explore and investigate some common problems that occur in recognition systems that are neural network based. It will be seen that the convolutional neural network performed best compared to the other networks on variances like translation, rotation, scale mismatch, and noise; followed by the deep belief network on the average,

while its tolerance to noise decreased noticeably as the level of noise was increased as shown in Table 5.3, and Figure 5.11.

A noteworthy attribute of the patterns (Yoruba vowel characters) used in validating this research is that they contain diacritical marks which increases the achievable variations of each pattern, and as such, recognition systems designed and described in this work have been tasked with a harder classification problem.

The performance of the denoising auto encoder (DAE-1 hidden layer) and stacked denoising auto encoder (SDAE-2 hidden layer), on the average with respect to the variances in character images seems to second the deep belief network.

The performance of the denoising auto encoder is lower than that of the stacked denoising auto encoder, it can be conjured that the stacked denoising auto encoder is less sensitive to the randomness of the input; of course the training and validation errors for the SDAE are lower to the DAE, and the tolerance to variances introduced into the input significantly higher. i.e. a kind of higher hierarchical knowledge of the training data achieved.

5.6 Summary

This chapter presents the simulation of the different designed network architectures; the considered invariances of interest to this research were simulated and achieved recognition rates reported in the tables above. Also, a graph showing the tolerance of the networks to various level noise of densities is herein supplied.

CHAPTER SIX
CONCLUSION AND RECOMMENDATIONS

6.1 Conclusion

It is the hope that machine learning algorithms and neural network architectures which, when trained once, perform better on variances that can occur in the patterns that they have been trained with can be explored for more robust applications. This also obviously saves time and expenses in contrast to training many different networks for such situations.

Furthermore, building invariances by the inclusion of all possible pattern variances in the training phase, which can occur when deployed in applications is one solution; unfortunately this is not always feasible as the capacity of the network is concerned. i.e. considering number of training samples enough to guarantee that proper learning has been achieved.

It can be seen that the major problem in deep learning is not in obtaining low error rates on the training and validation sets (i.e. optimization) but on the other databases which contain variant constraints of interest (i.e. regularization). These variances are common constraints that occur in real life recognition systems for handwritten characters, and some of the solutions have been constraining the users (writers) to some particular possible domains of writing spaces or earmarked pattern of writing in order for low error rates to be achieved.

It is noteworthy that from the recognition rates obtained in Table 5.1, 5.2, and 5.3, it can be inferred that pre-training while has both optimization and regularization effect as has been observed by researchers, this research reinforces that the optimization effect is larger; this is seen in that lower error rates were obtained from the deep networks that were pre-trained (DAE, SDAE, DBN) compared to the networks without pre-training (BPNN 1-layer and BPNN 2-layers). In addition, it will be seen that as the level of added noise was increased, the errors on the deep networks began to rise; at 30% noise level, the shallow network (BPNN 1-layer) has the second best highest recognition rate, which can be explained by the fact that it has the lowest number of network units (neurons) and therefore a lower possibility of over-fitting data (see Table 5.3). It will be noticed that even though the stacked auto encoder has more units than the denoising auto encoder, hence should have had higher error rates as noise was increased (i.e. due to overfitting) as observed in the deep belief network, the SDAE was pre-trained using the drop-out technique, and which success in fighting over-fitting can be seen in Table 5.3, and Figure 5.11.

It has been shown that a flavour of neural networks, "convolutional networks" and its deep variant give very motivating performance on some of these constraints, however the complexity and computational requirements of these networks are somewhat obvious.

This work reviews the place of deep learning, a simpler architecture, in a more demanding sense, that is, a "train once-simulate all" approach; and how well these networks accommodate the discussed variances. It is the hope that with the emergence of better deep learning architectures and learning algorithms that can extract features that are less sensitive to these constraints, a new era in deep learning, neural networks and machine learning field could emerge in the near future.

6.2 Recommendations

This thesis has reviewed the performance of some neural network models on some common problems that occur in pattern recognition systems, pattern invariance, based on the built-in structure of such networks in accommodating or tolerating the pattern constraints considered. It is the view that the number of hidden layers of the BPNNs without pre-training can be increased to ascertain if there will be any appreciable improvement in performance considering the discussed invariances; also, the convolutional and stacked denoising auto encoder networks hidden layers can be increased in view of improvements in performance but bearing in mind the rapid increase in computational power that will be required in achieving training in reasonable time. e.g. processors of many cores may be required.

Furthermore, other types of generative networks that can be used in the pre-training of the deep networks, restricted Boltzmann machine, may be considered.

Lastly, in recent times, the pre-training of convolutional neural networks have been strongly considered, with the hope of initializing the weights to values that are favourable in converging to better local minimum; the auto encoder pre-training scheme can be exploited more for this purpose, but network initialization from deep Boltzmann machines could be quite a promising option as well.

REFERENCES

Agrawal, P. (2012). Collocation Based Approach For Training Recurrent Neural Networks, University of Carlifonia, Berkeley, Retrieved January 15, 2015, from http://www.eecs.berkeley.edu/~pulkitag/collocation-report.pdf

Back propagation neural network: Image Recognition with Neural Networks. (2007). Retrieved January 14, 2015, from http://www.codeproject.com/Articles/19323/Image-Recognition-with-Neural-Networks

Bengio, S. (2003). An Introduction to Statistical Machine Learning: Neural Networks, Dalle Molle Institute for Perceptual Artificial Intelligence (IDIAP). Retrieved January 16, 2015, from http://bengio.abracadoudou.com/lectures/old/tex_ann.pdf

Bengio, Y., et al. (2007). Greedy Layer-Wise Training of Deep Networks. *In Proceedings of Advances in Neural Information Processing Systems 19 (NIPS'06)* (pp. 153-160). Qu´ebec: Universit´e de Montr´eal Montr´eal.

Borga, M. (2011). Neural networks and learning systems. Retrieved February 12, 2015, from: https://www.cs.cmu.edu/~tom/10701_sp11/slides/CCA_tutorial.pdf

Bose, N., & Liang, P. (1996). Neural Network Fundamentals With Graphs, Algorithms and Applications: McGraw-Hill, Inc.

Bottou, L. (2012). Stochastic Gradient Tricks, Neural Networks, Tricks of the Trade, Reloaded, In G. Montavon, G.B. Orr and K. Müller. *Lecture Notes in Computer Science, Springer*, (LNCS 7700), 430–445.

Chakraborty, R. C. (2010). Fundamentals of Neural Networks : AI Course lecture 37 – 38, notes, slides. Retrieved January 27, 2015, from: www.myreaders.info/html/artificial_intelligence.html

Ciresan, D. C., Meier, U., Masci, J., Gambardella, L. M, & Schmidhuber, J. (2011). Flexible, High Performance Convolutional Neural Networks for Image Classification. *In Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence* (pp. 1238-1242). University of NSW.

Debes, K., Koenig, A., & Gross, H. (2005). Transfer Functions in Artificial Neural Networks - A Simulation-Based Tutorial. *Brains, Minds and Media*, *151*(7), 172-182.

Delorme1, A., Rousselet, G.A, Mace´, M., & Fabre-Thorpe, M. (2004). Research report: Interaction of top-down and bottom-up processing in the fast visual analysis of natural scenes, *Journal of Cognitive Brain Research*, 19 , 103-113.

Deng, L., & Yu, D. (2014). Deep Learning: Methods and Applications. *Foundations and Trends in Signal Processing*, *7*(3), 197-387, doi: 10.1561/2000000039.

Eluyode O. S., & Akomolafe, D. T. (2013). Comparative study of biological and artificial neural networks, *European Journal of Applied Engineering and Scientific Research*, *2*(1), 36-46.

Engelbrecht, A. P. (2007). Computational Intelligence: An Introduction. John Wiley & Sons Ltd.

Glorot, X., & Bengio, Y. (2011). Sparse Rectifier Neural Networks. *In Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics* (pp. 315-323).

Gonzalez, R. C., Woods, R. E., & Eddins, S. L. (2004). Digital Image Processing using MATLAB: Pearson Prentice Hall.

Graupe, D. (2007). Principles of Artificial Neural Networks, World Scientific Publishing Co. Pte. Ltd.

Günther, F., & Fritsch, S. (2010). Neuralnet: Training of Neural Networks. *The R Journal*, *2*(1), 30-38.

Haykins, S. (1999). Neural Networks A Comprehensive Foundation: Prentice-Hall International Inc.

Hinton, G. E, Osindero, S., & Teh, Y. (2006). A Fast Learning Algorithm for Deep Belief Nets. *Neural Computation*, 18, 1527–1554.

Hristev, R. M. (1998). The ANN book, *Reinforced learning* (pp.112-127): Niranjan Desai Press.

Jain, A. K, Mao, J., & Mohiuddin, K. M. (1996). Artificial Neural Networks: A Tutorial, *Computer - Special issue: neural computing: companion issue to Spring IEEE Computational Science & Engineering archive*, *29*(3), 31-44.

Khashman, A. (2004). Class notes for MSc. Course EE532, Near East University, Faculty of Engineering. Retrieved April 5, 2014, from: neu.edu.tr/en/node/6056

Kpalma, K., & Ronsin, J. (2007). An overview of advances of pattern recognition systems in computer vision. In Obinata and Dutta (Eds.), *Vision systems: Segmentation And Pattern Recognition* (pp.546-572). Vienna: I-Tech Education and Publishing.

Le, Q. V., Ngiam, J., Chen, Z. et al. (2010). Tiled convolutional neural networks. *In Proceedings of the Twenty-fourth Annual Conference on Neural Information Processing Systems* (pp.134-141). Canada: Vacouver.

LeCun, Y., Boser, B., & Denker, J. S. (1990). Handwritten digit recognition with a back-propagation network. *Advances in Neural Information Processing Systems*, 2, 396–404.

LeCun, Y., Bottou, L., Bengio, Y., et al. (1998). Gradient-Based Learning Applied to Document Recognition. *In Proceedings of the IEEE, 86*(11), 2278-2324.

Leibo, J. Z., Mutch, J., Rosasco, L. et al. (2012). Learning Generic Invariances in Object Recognition: Translation and Scale. Front Comput Neurosci, doi: 10.3389/fncom.2012.00037

Leverington, D. (2009). A Basic Introduction to Feedforward Backpropagation Neural Networks. Retrieved January 20, 2015, from http://www.webpages.ttu.edu/dleverin/neural_network/neural_networks.html

Li Deng. (2014). An Overview of Deep-Structured Learning for Information Processing. *In Proceedings of Asia-Pacific Signal and Information Processing Association: Annual Summit and Conference* (pp.2-4). Cambodia: Siem Reap.

McGuire, M. (1998). An image registration technique for recovering rotation, scale and translation parameters. Massachusetts Institute of Technology, Cambridge MA, Multilayer Neural Network Design. Retrieved December 23, 2014, from: http://mnemstudio.org/neural-networks-multilayer-perceptron-design.htm

Mitchell, T. M. (1997). Machine Learning: McGraw-Hill Science/Engineering/Math, ISBN: 0070428077

Mohamed, A., Hinton, G., & Penn, G. (2012). Understanding How Deep Belief Networks Perform Acoustic Modelling. *In Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (pp. 4273 – 4276). Canada: University of Toronto.

Mooney, R. J. (2008). CS 343: Artificial Intelligence: Neural Networks. Retrieved October 28, 2014, from: http://www.cs.utexas.edu/~mooney/cs343/slide-handouts/neural.pdf

Oyedotun, O. K., & Khashman, A. (2014). Intelligent Road Traffic Control using Artificial Neural Network. *In Proceedings of 3rd International Conference on Information and Intelligent Computing* (pp.145-152). Hong Kong

Sauder, N. (2013). Encoded Invariance in Convolutional Neural Networks. *In Proceedings of IEEE Conference on Computer Vision and Pattern Recognition* (pp.133-139). Chicago: University of Chicago.

Serre, T. (2014). Hierarchical models of the visual system. *Encyclopedia of Computational Neuroscience*, 1, 345-352.

Shiffman, D. (2012). The Nature of Code: Paperback Press.

Sibi, P., Jones, S. A, Siddarth, P. (2013). Analysis Of Different Activation Functions Using Back Propagation Neural Networks. *Journal of Theoretical and Applied Information Technology*, *47*(3), 1264-1268.

Vasilev, I. (2014). An Introduction to Deep Learning: From Perceptrons to Deep Networks. Retrieved January 10, 2015, from http://www.toptal.com/machine-learning/an-introduction-to-deep-learning-from-perceptrons-to-deep-networks

Wilson, B. (2012). The Machine Learning Dictionary: Biological neuron. Retrieved January 16, 2015, from http://www.cse.unsw.edu.au/~billw/mldict.html

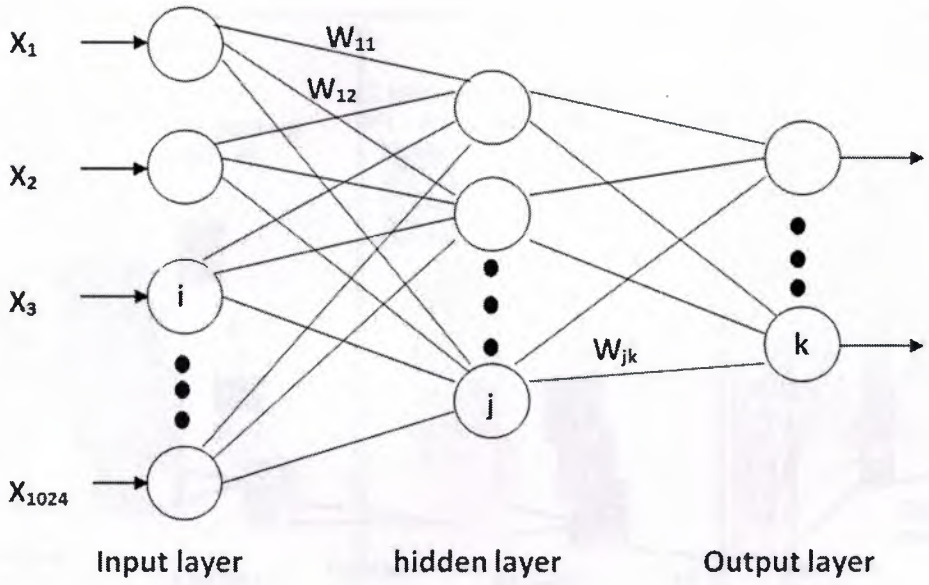
Yen, G. G. (2014). Evolutionary Computation, National Taipei University of Technology. Retrieved January 27, 2015, from http://www.cc.ntut.edu.tw/~ljkau/Course/981/ec/handout/1.1-%20Computational%20Intelligence.pdf

Yu, Z. (2010). Feed-Forward Neural Networks. Retrieved January 28, 2015, from: http://www.yukool.com/nn/layered.htm

Zhou, H., Wu, J., & Zhang, J. (2010). Digital Image Processing: Bookboon Publishing Co. Ltd.

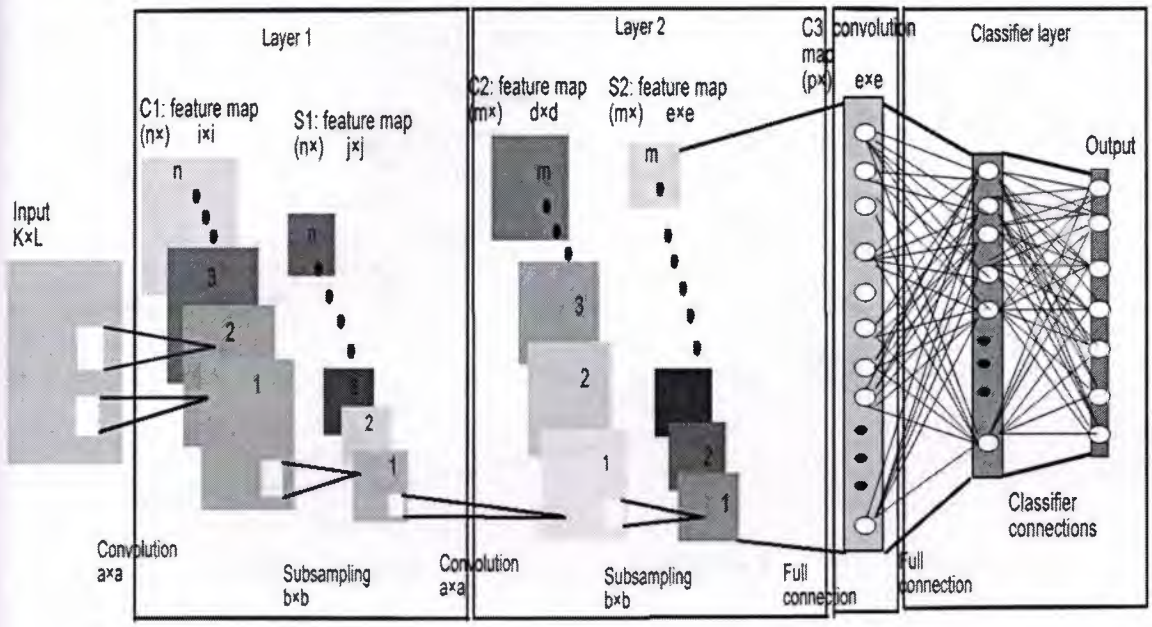
APPENDIX A

BPNN ARCHITECTURE



APPENDIX B

CNN ARCHITECTURE



K×L = 32×32

a×a = 5×5

n = 6

C1: i×i = 28×28

b×b = 2×2

S1: j×j = 14×14

m = 12

C2: d×d = 10×10

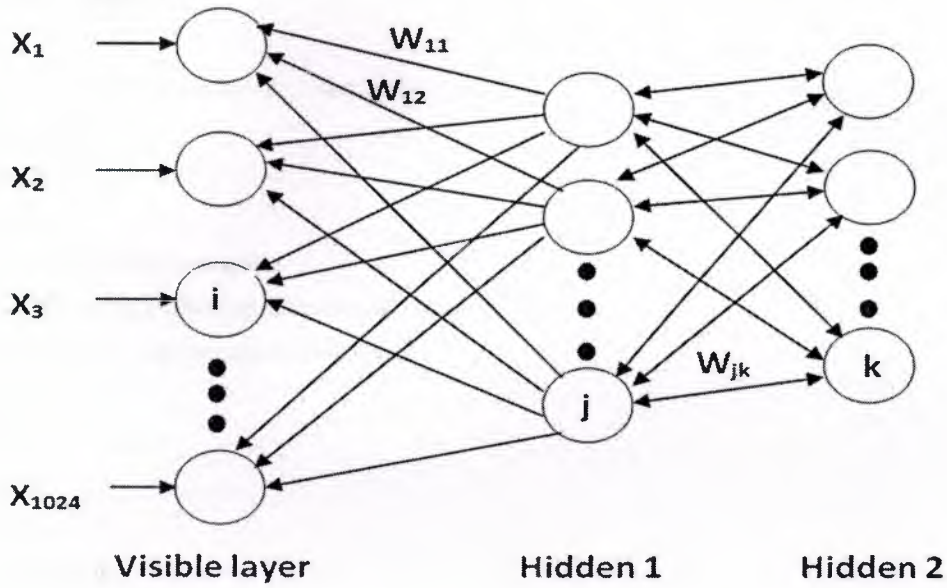
S2: e×e = 5×5

p = 12

C3: e×e = 5×5

APPENDIX C

DBN ARCHITECTURE



APPENDIX D

IMAGE PROCESSING CODES

```
%Training data ....character
for k=1
eyed = strcat('Train_data_vowels\A\A_', num2str(k), '.jpg');
a1 = imread(eyed);
b1=rgb2gray(a1);
b1=im2bw(b1);
c1=(1-b1);% covert image to negative
m1=medfilt2(c1,[10 10]);% median filter of mask 10 by 10
[v,w]=find(m1~=0);% crop out pattern from image
xmin=min(w);
xmax=max(w);
ymin=min(v);
ymax=max(v);
width = xmax - xmin;
height = ymax - ymin;
m3 = imcrop(m1,[xmin,ymin,width,height]);
m4=imresize(m3,[32 32]);% pixel no = 1024
d1=round(m4);
e1=reshape(d1,[ ],1);
for i=15:15:345
f=imrotate(m1,i,'nearest','crop');% rotate images at 15degrees counterclockwise,23 images
[v,w]=find(f~=0);% crop out pattern from rotated images
xmin=min(w);
xmax=max(w);
ymin=min(v);
ymax=max(v);
width = xmax - xmin;
height = ymax - ymin;
m3 = imcrop(f,[xmin,ymin,width,height]);
m4=imresize(m3,[32 32]);% pixel no = 1024
m5=round(m4);
g=reshape(m5,[ ],1);% reshape 32 by 32 1 to 1024 by 1 matrix
h=[e1 g];% concatenate reshaped images horizontally
e1=h;% safe images for reuse
s1=e1;% safe e1 into s1,incase
end
```

```
end
for k=2:85
eyed = strcat('Train_data_vowels\A\A_', num2str(k), '.jpg');
a1 = imread(eyed);
b1=rgb2gray(a1);
b1=im2bw(b1);
c1=(1-b1);% covert image to negative
m1=medfilt2(c1,[10 10]);% median filter of mask 10 by 10
[v,w]=find(m1~=0);% crop out pattern from image
xmin=min(w);
xmax=max(w);
ymin=min(v);
ymax=max(v);
width = xmax - xmin;
height = ymax - ymin;
m3 = imcrop(m1,[xmin,ymin,width,height]);
m4=imresize(m3,[32 32]);% pixel no = 1024
d1=round(m4);
e2=reshape(d1,[ ],1);
for i=15:15:345
f=imrotate(m1,i,'nearest','crop');% rotate images at 15degrees counterclockwise,23 images
[v,w]=find(f~=0);% crop out pattern from rotated images
xmin=min(w);
xmax=max(w);
ymin=min(v);
ymax=max(v);
width = xmax - xmin;
height = ymax - ymin;
m3 = imcrop(f,[xmin,ymin,width,height]);
m4=imresize(m3,[32 32]);% pixel no = 1024
m5=round(m4);
g=reshape(m5,[ ],1);% reshape 32 by 32 1 to 1024 by 1 matrix
h=[e2 g];% concatenate reshaped images horizontally
e2=h;% safe images for reuse
r=e2;
end
q=zeros(1024,1);% zeros matrix for concantenation
u=[e1 r];% concantenation with k=1
e1=u;
end
```

```
A1=u;
t1=ones(1,2040);
t2=zeros(6,2040);
A_target=[t1;t2];
Ai=[A1;A_target];
S1=Ai;% safe Ai
```

APPENDIX E

BPNN-1 CODES

```
% Solve a Pattern Recognition Problem with a Neural Network
% This script assumes these variables are defined:
inputs = A1_train;
targets = A1_target;
% Create a Pattern Recognition Network
numHiddenNeurons = 65;
net.trainParam.lr=0.045
net.trainParam.mc=0.72
net = newpr(inputs,targets,numHiddenNeurons);
% Set up Division of Data for Training, Validation, Testing
net.divideParam.trainRatio = 70/100;
net.divideParam.valRatio = 15/100;
net.divideParam.testRatio = 5/100;
% Train the Network
[net,tr] = train(net,inputs,targets);
% Test the Network
outputs = net(inputs);
errors = gsubtract(targets,outputs);
performance = perform(net,targets,outputs)
% View the Network
view(net)
% Plots
% Uncomment these lines to enable various plots.
% figure, plotperform(tr)
% figure, plottrainstate(tr)
% figure, plotconfusion(targets,outputs)
% figure, ploterrhist(errors)
```

APPENDIX F

BPNN-2 CODES

```
% Solve a Pattern Recognition Problem with a Neural Network
inputs = A1_train;
targets = A1_target;
% Create a Pattern Recognition Network
numHiddenNeurons = [95 65];
net.trainParam.lr=0.082
net.trainParam.mc=0.65
net = newpr(inputs,targets,numHiddenNeurons);
% Set up Division of Data for Training, Validation, Testing
net.divideParam.trainRatio = 70/100;
net.divideParam.valRatio = 15/100;
net.divideParam.testRatio = 5/100;
% Train the Network
[net,tr] = train(net,inputs,targets);
% Test the Network
outputs = net(inputs);
errors = gsubtract(targets,outputs);
performance = perform(net,targets,outputs)
% View the Network
view(net)
% Plots
% Uncomment these lines to enable various plots.
% figure, plotperform(tr)
% figure, plottrainstate(tr)
% figure, plotconfusion(targets,outputs)
% figure, ploterrhist(errors)
net = newpr(inputs,targets,numHiddenNeurons);
% Set up Division of Data for Training, Validation, Testing
net.divideParam.trainRatio = 70/100;
net.divideParam.valRatio = 15/100;
net.divideParam.testRatio = 5/100;
```

APPENDIX G

CNN CODES

```
%Train CNN
train_x=A1_train';
train_y=A1_target';
test_x=A1_test';
test_y=A1_test_desired';
train_x = double(reshape(train_x',32,32,14280));
test_x = double(reshape(test_x',32,32,2520));
train_y = double(train_y');
test_y = double(test_y');
%% ex1 Train a 6c-2s-12c-2s Convolutional neural network
%will run 1 epoch in about 200 second and get around 11% error.
%With 100 epochs you'll get around 1.2% error
rand('state',0)
cnn.layers = {
    struct('type', 'i') %input layer
    struct('type', 'c', 'outputmaps', 6, 'kernelsize', 5) %convolution layer
    struct('type', 's', 'scale', 2) %sub sampling layer
    struct('type', 'c', 'outputmaps', 12, 'kernelsize', 5) %convolution layer
    struct('type', 's', 'scale', 2) %subsampling layer
};
cnn = cnnsetup(cnn, train_x, train_y);
opts.alpha = 0.8;
opts.batchsize = 60;
opts.numepochs = 20;
cnn = cnntrain(cnn, train_x, train_y, opts);
[er, bad] = cnntest(cnn, test_x, test_y);
%plot mean squared error
figure; plot(cnn.rL);
```

APPENDIX H

DAE CODES

```
% Train SDAE
train_x=A1_train';
train_y=A1_target';
test_x=A1_test';
test_y=A1_test_desired';
train_x = double(train_x);
test_x  = double(test_x);
train_y = double(train_y);
test_y  = double(test_y);
%% ex1 train a 100 hidden unit DAE and use it to initialize a FFNN
%  Setup and train a stacked denoising autoencoder (DAE)
rand('state',0)
sae = saesetup([1024 100]);
sae.ae{1}.activation_function       = 'sigm';
sae.ae{1}.learningRate              = 0.8;
sae.ae{1}.inputZeroMaskedFraction   = 0.5;
opts.numepochs =   10;
opts.batchsize = 60;
sae = saetrain(sae, train_x, opts);
visualize(sae.ae{1}.W{1}(:,2:end)')
% Use the SDAE to initialize a FFNN
nn = nnsetup([1024 100 7]);
nn.activation_function              = 'sigm';
nn.learningRate                     = 0.7;
nn.W{1} = sae.ae{1}.W{1};
% Train the FFNN
opts.numepochs =   80;
opts.batchsize = 60;
nn = nntrain(nn, train_x, train_y, opts);
[er, bad] = nntest(nn, test_x, test_y);
assert(er < 0.16, 'Too big error');
```

APPENDIX I

SDAE CODES

```
% Train DAE
train_x=A1_train';
train_y=A1_target';
test_x=A1_test';
test_y=A1_test_desired';
train_x = double(train_x);
test_x  = double(test_x);
train_y = double(train_y);
test_y  = double(test_y);
%% train a 95 and 65  hidden units for 1st & 2nd hidden layers respectively and use it to initialize a FFNN
%  Setup and train a stacked denoising autoencoder (SDAE)
rand('state',0)
sae = saesetup([1024 95 65]);
sae.ae{1}.activation_function       = 'sigm';
sae.ae{1}.learningRate              = 0.8;
sae.ae{1}.inputZeroMaskedFraction   = 0.5;
opts.numepochs =   10;
opts.batchsize = 60;
sae = saetrain(sae, train_x, opts);
visualize(sae.ae{1}.W{1}(:,2:end)')
% Use the SDAE to initialize a FFNN
nn = nnsetup([1024 95 65 7]);
nn.activation_function              = 'sigm';
nn.learningRate                     = 0.7;
nn.W{1} = sae.ae{1}.W{1};
% Train the FFNN
opts.numepochs =   80;
opts.batchsize = 60;
nn = nntrain(nn, train_x, train_y, opts);
[er, bad] = nntest(nn, test_x, test_y);
assert(er < 0.16, 'Too big error');
```

APPENDIX J

DBN CODES

```
% Train DBN
train_x=A1_train;
train_y=A1_target;
test_x=A1_test;
test_y=A1_test_desired;
train_x = double(train_x');
test_x  = double(test_x');
train_y = double(train_y');
test_y  = double(test_y');
%% ex2 train a 200-150 hidden unit DBN and use its weights to initialize a NN
rand('state',0)
%train dbn
dbn.sizes = [200 150];
opts.numepochs =   5;
opts.batchsize = 60;
opts.momentum  =   0;
opts.alpha     =   0.2;
dbn = dbnsetup(dbn, train_x, opts);
dbn = dbntrain(dbn, train_x, opts);
%unfold dbn to nn
nn = dbnunfoldtonn(dbn, 7);
nn.activation_function = 'sigm';
%train nn
opts.numepochs =  20;
opts.batchsize = 60;
nn = nntrain(nn, train_x, train_y, opts);
[er, bad] = nntest(nn, test_x, test_y);
assert(er < 0.10, 'Too big error');
```