A COMPONENT BASED KNOWLEDGE TRANSFER MODEL FOR DEEP NEURAL NETWORKS

by

Sreenivas Sremath Tirumala

A thesis submitted in conformity with the requirements for the degree of Doctor of Philosophy,

School of Engineering, Computer + Mathematical Sciences
Auckland University of Technology,
Auckland, New Zealand

To my parents,

Sarala Devi

&

Venugopala Charyulu

who always believed in me

Acknowledgements

I would like to offer my soulful gratitude to Prof. Ajit Narayanan, for his guidance, counsel, encouragement, patience and knowledge sharing. Working under his supervision is the best experience I could ever have as I was blessed to experience the presence of such a ‘knowledgeable’ person and a true ‘Guru’. His intellect and professionalism made me who I am today as a researcher.

Assoc. Prof. Jacqueline Whalley provided critical recommendations as and when required and posed right questions to me at the right time. Her direction helped me to come up with alternate ideas and guided me to complete this thesis.

A special thanks to Mark, my ‘kiwi brother’ who provided the ‘necessary facilities’ including a beautiful and peaceful place to stay and helped me to keep focussed at the moments of distress.

I would like to thank my ‘divine’ sister Ramani who believed in me and sponsored my holiday trips. Gratitude to my other sisters Neeha, Yuki, Gayatri, Ahila and Noreen for listening to me with patience.

I would like to thank Vijay Naidu, Ahmad Wedyan, Shahid Ali and Maheswar Rao Valluri for their assistance through fruitful discussions. I would like to acknowledge the help of Dr. Seyed Reza Shahamiri for his suggestions in research activities particularly with experimental evaluation. A heartful thanks to David Nandigam, Murali, John Anna, Sathish and Anand Petrus for their encouragement.

A special thanks to the CMS PhD team particularly Karishma & Saide along with other support staff at AUT.

My brother Vamsi and friends Saravana, Chandu, Kiran & Phani needs a special mention for their support and help back in India.

Gratitude to my Guru Sri Chinmoy who showed me ‘The True Light’

Attestation of Authorship

I hereby declare that this submission is my own work and that, to the best of my knowledge and belief, it contains no material previously published or written by another person nor material which to a substantial extent has been accepted for the qualification of any other degree or diploma of a university or other institution of higher learning.

Signature of the Student

13-Aug-2019.

Student ID: 1260514

Abstract

This thesis explores the idea that features extracted from deep neural networks (DNNs) through layered weight analysis are knowledge components and are transferable. Among the components extracted from the various layers, middle layer components are shown to constitute knowledge that is mainly responsible for the accuracy of deep architectures including deep autoencoders (DAEs), deep belief networks (DBNs) and DNNs. The proposed component-based transfer of knowledge is shown to be efficient when applied to a variety of benchmark datasets including handwritten character recognition, image recognition, speech analysis, gene expression, as well as hierarchical feature datasets.

The importance of hidden layer and its position in the topology of Artificial Neural Networks (ANNs) is under-researched in comparison to the deployment of new architectures, components and learning algorithms. This thesis addresses this imbalance by providing an insight into what actually is learned by a neural network. This is because recent advances in layer-wise training enable us to explore systematically and rigorously the features that expose hidden layer by hidden layer in deep architectures.

The key contribution of this research is providing a transferable component model by extracting knowledge components from hidden layers. This thesis also provides an approach to determine the contribution of individual layers, thus providing an insight into the topological constraints that require addressing while designing a transfer learning model. Such transfer learning can mitigate the problem of needing to train each neural network 'from scratch.' This is important since deep learning currently can be slow and require large amounts of processing power. "Warm started" deep learning may open new avenues of research, especially in areas where 'portable' deep architectures can be deployed for decision making.

Publications

Citation	Page No.	Ref. No.
Tirumala, S. S., A. Narayanan. Classification and Diagnostic Prediction of Prostate Cancer Gene expression dataset. *Neural Computing and Applications*, (**2018)**	7, 142	56
Sreenivas Sremath Tirumala Deep Learning Using Unconventional Paradigms. *International Journal of Computer Research, 23*(3), 295. **(2016)**	6, 49, 50	36
Tirumala, S. S. (2018). A Deep Autoencoder-Based Knowledge Transfer Approach. In *Proceedings of International Conference on Computational Intelligence and Data Engineering* (pp. 277-284). Springer, Singapore. **(2018)**	57,160	203
Sreenivas Sremath Tirumala, A. Narayanan: *Hierarchical Data Classification Using Deep Neural Networks*. Neural Information Processing, 11/2015: pages 492-500;	6, 9, 43-46, 69, 80, 84-85	22

Contents

List of Tables

List of Figures

Chapter 1 Introduction

1.1. The Blossom Effect

Nature possesses true intelligence which has been adopted by humans to create intelligent systems. The opaque nature of Artificial Neural Networks (ANNs) has not been explored enough to provide reason for the success of ANNs. Questions like why ANNs are successful and what is the knowledge that ANNs possess still remain as open questions. Knowledge is the key to the success of ANNs particularly with respect to deep learning and is the basis of the learning mechanism of ANNs with large number of layers called deep neural networks (DNNs) [1].

The Blossom Effect is the process that occurs internally in DNNs in which features are folded into multi-level components in the middle layer and then are unfolded after passing through the middle layer towards classifiers similar to the diurnal/nocturnal cycle of the Sacred Lotus flower.

The Sacred Lotus flowers bloom by day and close by night and again bloom in the next morning. When a Sacred Lotus flower is closed, it folds the petals so that they can be unfolded again and bloom the next day perfectly. This is similar to what happens in the process of dimensionality reduction - a similar principle of ANNs - where the features are folded into the middle layer(s) and then reproduced for classification [2]. However, conventional neural

network theories provide no clear explanation on how the features are reconstructed after condensing them as weights that are just numerical values.

Dimensionality reduction could possibly be explained through further research on autoencoders, a special type of ANNs [3]. An autoencoder reproduces the output from input. The input is encoded into a condensed layer and then decoded from the values of that layer. The weights in the layers are mere numbers and possess some patterns when they are extracted through several different statistical techniques.

Consider a set of features condensed into a super feature. It is not feasible to bring back the original set of features from the super feature without knowing the exact process adopted in the condensation; is it some sort of folding or placing one upon the other so that they can be differentiated? For instance, an input of two numbers is averaged, and the result is two and it is impossible to find out what exact numbers were used as input.

1.1.1. Knowledge Components and the Blossom Effect

The Blossom Effect emphasises that the weights present in the middle layer(s) of a trained DNN possess underlying representations of input features or knowledge that responsible for the learning within the DNN. If this knowledge could be extracted from the weights of a trained DNN as knowledge components, they can provide same learning capability (may not be to the same extent) to an untrained DNN. The knowledge components that are transferred could, in theory, provide an efficient and systematic mechanism for transfer learning in DNNs. Such a solution diverges from existing transfer learning approaches.

1.1.2. Extraction of Knowledge Components

To extract knowledge components, a new transferable model is proposed that complements existing statistical models and the neural network transfer learning principles. In this research, a new concept called Weights of Weights (WofW) is used to extract a smaller number of weights from a large number of weights of a DNN. It is evident that not every weight in a DNN layer is necessarily important and some weights can even hinder the training process [4]. The WofW is a process to retain the knowledge in the middle layer(s) by condensing two layers (weights) which is similar to dimensionality reduction. Hence, the weights in the middle layer

that consists of underlying knowledge are WofW. The WofW concept along with statistical methods is adopted to extract transferable knowledge components, thus providing confidence in success through a well-established procedure. Chapter 5 provides mathematical, technical and experimental details of a proposed transferable knowledge component model.

This thesis investigates the existence of the Blossom Effect in DNN and provides a knowledge component model for deep transfer learning.

1.2. Significance of Investigating Deep Learning

The recent success of AI is attributed to 'deep learning' - a branch of ML. Deep learning is used for image and speech-based systems irrespective of domains and implementations. Deep learning is popular and is widely used in various AI based systems as adopted by Google, Microsoft, Apple, Amazon, Facebook, Twitter to name a few. The deep learning algorithm is used in applications such as image recognition, natural language processing, automated systems, bioinformatics, healthcare, and recently in neural network based cryptography called neurocryptography [5]. Deep learning has attained the state-of-the-art results for various benchmark problems and is widely used among all AI algorithms. ANN and its variants are the principal building blocks for deep learning architectures. In spite of the popularity and wide range of implementations, there is a lack of a clear consensus on the design of deep architectures and the working principles behind deep learning. For instance, the decision on initial topology, selection of parameters and the number of hidden nodes is still considerably difficult due to the lack of a standardised formal theory.

1.3. Artificial Neural Networks and Deep Learning

ANNs are connectionist systems inspired by the functionality of animal/human brain [6]. After intense research on the functionality of logical calculations in human brain, McCulloh and Pitt were inspired to think about creating a computing system based on brain activity. In 1943, McCulloh and Pitt published their work titled *"A logical calculus of the ideas immanent in nervous activity"* which is the first known work on ANNs [7]. However, the application of ANNs as practical systems became possible only after 30 more years when Paul Werbos, for the first time, described a mechanism for training ANNs through Back Propagation of errors [8, 9].

The initial research on modelling the cognitive and convolutional capability of human memory can be attributed to David Everett Rumelhart [10]. His early works on distributed processing [11] helped build and test intelligent models based on data. Rumelhart along with Hinton is attributed with the proposition of Back Propagation (BP) algorithm which is the fundamental learning principle of any ANN [12].

The hidden layer(s) where the weights are optimised provides no visible information on the functionality of ANN. The major criticism on ANNs is due to this 'hidden' functionality which describes ANNs as 'blackboxes' [13]. This is reflected in treating ANN applications as 'unreliable' in decision centric critical applications like medical diagnosis, financial forecasting, space and aeronautics, military and scientific research [14].

The research on ANNs, for the most part, concentrates on parameter optimization. The importance of architecture, especially in terms of its relationship with the problem solving, has not been investigated to the same degree. However, some prominent ANN researchers have hinted at the importance of input being systematically structured for the success of ANNs [15-17].

The training process of ANNs for supervised learning involves adjusting the weights and bias(es) so that the input produces a predefined output [16]. The trained ANN with known (training) data is expected to produce similar results for unknown (test) data. The most successful ANN training approach, the BP algorithm, uses the delta rule to determine the value for updating the weights [3]. The delta rule consists of a step parameter, called learning rate, with which the weights are updated. To escape the trap of local minima, a parameter called momentum is introduced which adds a fraction of previous weight updates to the learning rate. For the BP algorithm, learning rate and momentum are two vital parameters for optimising ANNs [18].

However, there is no standard procedure for setting the learning rate and momentum [19]. Generally, the ANN training starts with the learning rate at a higher value and a lower value for the momentum, both being set at random initially based on the problem and input data [20]. Sometimes an adaptive learning rate is used where the learning rate is updated in varying step sizes (either increasing or decreasing step sizes) depending on error gradient [18].

Apart from learning rate and momentum, it is vital to determine the number of hidden nodes for optimal ANN performance and accuracy. However, selection of the number of hidden nodes can be arbitrary. The problem of overfitting/underfitting also needs to be addressed. If training is too long, it results in overfitting whereas if the ANN is not trained enough it will cause underfitting. The presence of unnecessary nodes can also lead to overfitting. This problem has been addressed recently with a principle called dropout [4]. Dropout is the process of dropping (ignoring) randomly selected hidden nodes along with their connections during training.

Fukushima [21] initially proposed the Neocognitron approach by increasing number of hidden layers based on Hubul and Wiesel's theory on visual cortex phenomena of 1957. The practical implementation of Neocognitron was carried out by Lecun [22] for zip code recognition which uses BP as a training algorithm. Lecun was able to attain a fair amount of classification accuracy in spite of slow training caused by both the size of the data and the number of layers.

The process of identifying patterns in the weights is significant in understanding the underlying representations in the weights. The underlying representations might provide an insight into the knowledge that is attained through training since there is a direct influence of weights (optimisation of weight values) on the efficiency of an ANN.

One of the approaches that could be used to understand the patterns in high dimensional numerical values is by projecting the values into a different space or dimension and analysing the visualisation obtained through this projection [23]. In the case of ANN with one hidden layer, all the representations are condensed in the weights of one hidden layer. It is challenging to discover patterns or other models from the condensed representations [17]. So, it is necessary to consider ANN with multiple hidden layers where the representations are distributed across multiple layers. Early attempts to construct and train ANNs with multiple hidden layers attained little success due to hardware limitations and lack of efficient training mechanisms [24]. At that time, training was very time consuming and often resulted in overfitting [4].

Since the primary objective of this thesis is to explore and extract the knowledge representations of ANNs, the following queries are important:

- ➢ What exactly are ANNs acquiring in the form of knowledge?
- ➢ What is ANN knowledge and how is this knowledge represented?
- ➢ Where is this knowledge present within the ANN architecture? In which form is the knowledge represented? Is it a component or group of components or a definitive model?

1.4. Deep Neural Networks and Deep Learning

Deep learning is a hierarchical learning mechanism based on ANN-centric ML approach, principally implemented on architectures with sufficient 'depth.' Deep learning enables us to extract discrete features from input through a progressive learning mechanism from one layer to another in a multi-layer ANN or simple Deep Neural Network (DNN) [3, 16]. Deep learning can be implemented only on deep architectures i.e., architectures with sufficient depth [15]. The term 'deep' is associated with the 'depth' and in the case of DNNs, its number of layers [17].

A DNN is a multilayer ANN trained to calculate the probability of the output being a certain type based on training. This process of this categorisation of output is done by passing through each layer to identify different characteristics of input. The main success of DNNs is their ability to model non-linear relationships in complex data with multiple and overlapping features. Initially, Fukushima, Lecun and Schmidhuber [22, 25, 26] attempted to train DNNs with simple BP which resulted in slow training and overfitting. To train these type of 'deep' architectures efficiently, a new greedy layer-wise training mechanism was introduced by multiple people in and around 2006 [3, 27, 28]. In the seminal deep learning approach in [3, 27, 28], the output of each layer is fed as the input to the next layer and trained against the previous layer. The first layer, therefore, extracts low-level features from input, and the next layers extract middle-level features, and the last layer extracts the high-level features. Traditional BP based DNNs lack this layer-wise training, hence all the features are represented in single or multiple hidden layers. In DNNs fine tuning with BP is used for evaluating the accuracy and back propagating the error to improve accuracy, a process similar to traditional ANN training.

When compared to earlier approaches, traditional approaches are capable of learning linear features within labelled data (supervised learning) whereas deep learning is capable of extracting non-linear features from complex multi-dimensional data with unsupervised training [16]. Diverging from the conventional training procedure of ANNs, deep learning adopts a new layer-wise training procedure (typically unsupervised) followed by a supervised fine-tuning using BP. Deep learning and deep architectures are discussed in depth in Chapter 2, Section 2.5. However, a general overview of deep learning is provided here for completeness.

Since the deep architectures are a form of ANNs with more hidden layers, knowing the importance of the 'depth' will provide an insight into how deep learning is efficient than the traditional learning approaches [3, 17]. ANN with one hidden layer lacks the ability to group similar features or correlate a set of low-level features as a representation of high-level features. This is due to the presence of features in the form of condensed representations in one hidden layer [17]. This also restricts ANNs to explore the hierarchy of features to discover hierarchical relationships between the features to extract lineage between them [29]. In contrast, DNNs can preserve the feature space and provide multi-level feature extraction to know the relationship between the features [29]. DNNs allow different functions to be assigned to different layers and even at different levels to construct a hierarchical problem structure [30]. Consequently, DNNs are able to deal with highly complex and domain specific problems as a set of smaller individuals, but related, sub-problems or multiple domain problems [31].

Thus, it can be concluded that the main strength of deep learning is its ability to extract features from input data at various levels/stages in a hierarchical fashion which is not possible with single-layered ANNs [29]. This makes DNN suitable for exploring weights in the layers to identify and extract knowledge. This is the motivation to use DNNs to identify/establish a relationship between input features and DNN layers as well as the weights.

In spite of being a slow process, deep learning is considered as the most successful approach in 2020 with the-state-of-the-art results demonstrated in a variety of application domains [32-44]. There are several successful unconventional deep learning approaches based on various established ML methods [45] including: Support Vector Machines (SVMs) [46, 47]; deep kernel machines [48]; deep spiking neural networks [49]; genetic algorithms [50]; evolutionary computation principles [51, 52] as well as a combination of traditional components like Boltzmann Machines and autoencoders [45]. These hybrid approaches are used for optimising

deep learning training mechanism and in some cases just to speed up the process. The hybrid deep learning approaches showed promising results for domain specific problems. Some of the recent unconventional approaches are presented briefly in the Chapter 2 Section 2.5.

The success of deep learning emphasizes the importance of architecture [15]. However, the deciding factor for architecture, once again, takes us back to identifying the number of hidden layers required for the experiment or how many layers to start with. Moreover, even after the experiment is started, it is uncertain whether to increase the number of layers or to reduce them to achieve good results. There is no certainty in relating the number of layers or even weights with the results and the entire process is carried out on a trial and error basis. The topology is decided arbitrarily and is often problem specific which helps to see the relationship between input and topology. However, the lack of formal literature to identify this relationship supports the necessity of this research in exploring the knowledge that resides in the hidden layers of ANNs.

1.5. Knowledge Discovery in Neural Networks

Knowledge extraction is an important aspect of ML research. Every ML algorithm attains knowledge through training and learning algorithms which enables it to perform classification, prediction or other tasks required for the problem solving [53]. It is important to understand that the transfer of knowledge from a learned system to a new system will have several advantages including providing a warm start to the new systems [53-58]. Considering the current applications of DNNs (as well as traditional ANNs which was discussed earlier), training time is one of the major drawbacks [59, 60]. DNNs require a huge amount of training data which in turn increases the training time and non-availability of every type of training data is also a major problem which could not be tackled just by training again and again [61]. Therefore, a simple research step towards transferring knowledge would provide a giant leap towards optimising DNNs which indirectly helps its deployment in small scale systems [53-55, 58, 62-64].

The process of knowledge extraction from ANNs is directly associated with investigating how the features in the input are represented in the hidden layers (topology) and neurons (weights). The research on extracting knowledge from ANNs was popular in the late 20th century [65-71]. A notable account of extracting knowledge from ANN weights is by removing irrelevant attributes from the input and training the model with known data (supervised) [69]. In the

removal process, the attributes that influence the classification were identified, and the rest of the attributes were removed. The importance and the influence of an attribute can be identified through various attribute selection methods from both statistical and ML approaches [72, 73]. Knowledge extraction through attribute removal is generally successful for linearly separable data. But for nonlinear multi-class data, the importance of an attribute may not be evaluated correctly in relation to other attributes [74].

Knowledge discovery in ANNs attained limited success for ANNs that are trained for a specific problem. For instance, the successful attempt to extract symbolic rules from ANN by Abruzzian and Monirul is confined to the single digit hidden nodes with a small dataset [75]. In other words, most of the earlier attempts for knowledge discovery from ANNs are confined to small data sets consisting of linear data and was not compatible with current DNNs and big data. This suggests that the extracting knowledge from DNNs could be considered as an area of research importance.

It is evident that there is some unknown relationship between the input, DNN's parameters and topology and the problem solving capability of DNNs since DNNs are optimised through parameters. The research adopted in this thesis tries to explore the relationship between input and the neural network topology (particularly weights) through a systematic research design, study and experimental evaluation. The importance of the topological structure and the individual parameters have the potential to provide a basic understanding of the key influential factors for training ANNs and DNNs and their learning processes. Furthermore, these key forces internally present in the DNNs that drive the process of deep learning should provide a starting point to explore the internals of DNNs.

1.6. Preliminary Investigation - Establishing Research Problem & Direction

To establish a better prospective of the research problem and the point of focus, some initial exploratory experiments were conducted on ANNs. These experiments can be categorised into two parts. Firstly, the evaluation of ANN parameters, namely, learning rate and momentum and their influence on different types of topologies. The second category is changing the topology of ANNs (addition and removal of layers). All experiments are performed using simple yet widely used benchmark datasets iris and wine.

The first set of experiments is conducted by changing learning rate and momentum one at a time with various types of architectures in two different strategies. The first strategy uses the architecture with symmetric node count (same number of nodes in the hidden layers) whereas the second strategy uses asymmetric node count. In the first set of experiments, for a fixed architecture, the learning rate and momentum are adjusted one at a time. In the second set of experiments, the architecture is adjusted keeping learning rate and momentum at constant values. Both strategies were used for the experiments. The influence of topology for a fixed learning rate and momentum is the same (if not more in some cases) compared to fixed topology and varying learning rate and momentum. Further, better results are obtained with symmetric node count in the hidden layers. The second set of experiments were conducted in order to gain familiarity about the influence of ANN architecture with respect to the problem space. These types of experiments require a high number of hidden layers. Thus, the experiments are performed on DNNs using layer-wise training.

The majority of ANN implementations are primarily one or two layered as a universal approximators [76]. The weights inside the hidden layer that are responsible for the functionality possess the problem solving knowledge in the form of representations. Since the number of layers are limited, the representations that are present are in the form of condensed representations making it impossible to extract patterns or comprehensible representations. The limited literature exploration on identifying and extracting representations has motivated to undertake this research. The aspect of viewing the relationship between input and neural network weights which has been largely ignored since the beginning of ANN research, has influenced to further investigate the existence of any connotation either direct or indirect between input and the neural network topology particularly weights in the hidden layers.

To start with, the influence of weights on problem space is studied through several experiments using a synthetic hierarchical dataset (in line with the traditional ANN representation as a tree-structure) and DNNs. The experiments aim at determining whether the DNNs are able to preserve any notation on the relationships that exists within the input data and how the DNN accuracy is affected by choosing hidden nodes in a particular symmetry. The results obtained from the experiments concluded with a new outcome that an architecture with the equal number of (hidden) nodes at each hidden layer performs more efficiently than the one having an asymmetric node count [29]. In other words, the results from the initial experiments showed

that irrespective of feature reduction, the number of hidden nodes should be kept the same as we traverse up the tree-hierarchy.

Since the mapping of the hierarchical structure of data with the hierarchical representation of architecture was found to be unsuccessful, the research investigation is directed towards representation of features in the neural network weights. There is no known study or evidence in the literature on feature representation being organized or structured in such a way that it reflects the topology [16]. However, the changes in the input demands considerable changes in the topology for obtaining efficient and accurate results [3, 17]. Moreover, there is no generalised neural network that could tackle the changes in the problem or variation in the input without modifying the topology or training parameters [77]. This led to the supposition that there is an unknown relationship that exists between structure representation of the input (problem) and the architecture of DNNs.

In spite of the success of deep learning, the fundamental questions about the functionality of ANNs particularly the internal knowledge that ANNs acquired by optimising weights through training is still a puzzle. The successful knowledge transfer approaches for DNN are based on transfer of weights or hidden layers and sometime whole topology without knowing on what is being transferred and why it is effective [53, 55]. Also, there is no known attempt to perform a systematic study on how the input is connected to DNN weights. This research is focused on investigating this aspect of DNNs by attempting to build a mathematical expression of the relationship that exists between a problem and the ANN/DNN topology. This mathematical expression will be useful for efficient knowledge extraction and helpful in gaining familiarity with the factors influencing the efficiency and performance of ANNs, especially the ability of DNNs to produce good results for previously unseen data.

While there has been some theoretical research published indicating that any problem can be solved by an ANN with one hidden layer [76], this is not the focus of this research. Instead, the aim of the research is to help future researchers identify suitable and appropriate DNN architectures that resemble in some form the type of problem being tackled. Such help may include identifying important hidden layers so that the DNN is an ‘analogue’ in some sense of the problem. If such an approach can work, it may then be possible to address one of the oldest problems in ANN research, which depicts how to extract symbolic knowledge from an ANN. The ‘analogue’ aspects may be providing a novel way to extract such knowledge. The unknown

question at this stage is as to how the choice of representation is related to DNN architecture. The research aim is to shed light on these important questions using the recent growth of interest in deep learning as a motivation. Further, the extraction of transferable knowledge components can provide an efficient transfer learning mechanism where the knowledge transfer is systematic and explorable instead of being unknown and closed. The practical application of the research would be extracting transferable knowledge from a trained DNN to provide a warm start to an untrained DNN.

With deep learning, DNNs are capable of learning discrete representations at multiple levels which paved the way for exploring the learning capabilities of DNNs. Recent advances in ANN research, particularly deep learning, has inspired to investigate ANN behaviour in general by unleashing deep learning. The exploration of ANN functionality provides an inspiration to get closer towards exploring and experiencing the consciousness of the ANN 'blackbox.'

1.7. Research Problem

The main purpose of this research is to examine:

"why artificial neural networks behave the way they do"

Traditionally, the focus of neural network research is on "How" to optimise a neural network for a given problem. Recent developments in neural networks, particularly deep learning models, also follow the same path of investigating how DNNs can be optimised for a particular problem. Since the implementation of multilayer neural networks, the fundamental problem has been determining a correct topology and weights. Neural Network weights are the functional building blocks of an ANN and these weights have been limited to representation as simple numeric values. The basic operation of a Neural Network depends on these weights. From the onset, the representation of these weights and their influence on ANNs has not been sufficiently explored.

There is a wide range of literature on 'how' to solve a particular problem using DNNs which includes different types of algorithms, ML approaches etc. However, there is only minimum information on **'why'** a topology with particular number of layers or nodes is able to solve a problem. The literature provides no standard or rationale on number of hidden layers and their importance to provide a viewpoint on which layer or layers are influential for accuracy and

efficiency of DNNs. Traditionally, the success of deep learning in a wide range of domains and applications is attributed to the availability of powerful hardware and efficient software resources. Due to this, the investigation into the reason on no formal or decisive procedure for studying the impact of adding or reducing layers has not been undertaken.

The increasing application of deep learning for various problems has forced researchers to identify new methods and approaches for choosing initial topology and parameters. Therefore, the importance of a systematic approach in identifying number of layers and the effect of changing layers is becoming more significant which is the core aspect of this thesis.

The most successful deep learning models are constructed by adding and removing layers and testing them on a specific problem, thus identifying near optimal topology without knowing **'why'** it is working. The academic literature and industrial white papers are full of deep learning implementations and their success on benchmark problems. There is no consistency or systematic research on knowing **'why'** a DNN's functionality is affected by the addition or subtraction of layers.

This research endeavours to provide a systematic approach in considering both theoretical and practical aspects of neural network learning, particularly deep learning, and towards exposing the internal mechanics of neural networks.

Therefore, the research problem is positioned as **'why'** rather than 'how.'

1.8. Research Focus

The focus of the research is on exploring the patterns that exist in the neural network weights to establish a relationship between input features and neural network weights. This would help explore the internal representations in the neural network weights that are influenced by the changes in the input; thus, providing an insight on impact of input features on the neural network topology.

Since deep learning is based on a form of feature learning through various hidden layers [16], this systematic research on extracting weight patterns enable presentation of ground work on how input features are transformed through each layer which will help to identify important layers that constitute knowledge and provide maximum impact on accuracy of the model.

Identifying these influential layers will enable investigate the functionality of DNNs by explaining how features are represented in neural network weights and how these features are transmitted from the input layer to the classifier through training. The learning that is attained by the neural network (through training) is the knowledge that may be transferable or that is what has essentially been transferred unknowingly.

The exploration for the knowledge in the neural network weights to identify and establish a profound relationship between input features and neural network weights is the key aspect of this research. Since, this type of research requires comprehensive study, the following research questions are identified. The contribution of the research can be established by the outcome of the following research questions:

1. **Exploring the representation in the weights**
    a. Why do changes in the weight values impact the classification?
    b. Do changes of weights in some layers have more or higher influence than other layers?
    c. Why does increasing or decreasing number of layers impact efficiency of neural networks?

2. **Establishing the relationship between input features and neural network weights**
    a. Why do any changes in the input require changes in the neural network topology/weights?
    b. What are some reasons that the changes in the relationships in input features present a clear change in the patterns in the neural network weights in various layers?
3. **Extracting transferable neural network model**
    a. Why is there no systematic research of extracting knowledge from the weights as a neural network component knowledge model?
    b. Are these knowledge components transferable (from one DNN to another DNN)?

The outcomes from the research questions will able to provide an interpretation of

- The representation of features in neural network weights in various layers
- Movement of features from one layer through another by neural network training
- Impact of altering the topology of a neural network by adding or removing layers
- The relationship between input features and weights
- Knowledge representations in the weights

1.9. Scope of the Research

The research evaluates the proposed model on deep architecture learning and is confined to deep neural networks (feed forward), deep belief networks and deep (stacked) autoencoder networks. However, some brief experiments are also conducted to ascertain the proposed model on Convolutional Neural Networks.

The research predominantly uses the benchmark datasets of character recognition, image, speech along with simple yet popular datasets like IRIS, Wine, and gene expression. Based on the requirements, several new versions of datasets are also created by modifying these benchmark datasets. Two synthetic datasets, one hierarchical and another random valued dataset are also used for experiments. The evaluation of the proposed model is performed on a variety of datasets to provide a generalisation on the datasets used in this research.

1.10. Thesis Contributions

The key contributions of the thesis are summarised as follows.

- **Exploring Neural Network Behaviour:** As mentioned in Sections 1.2 & 1.3, there is an almost non-existent research on the internal functionality of neural networks and its effect on DNN behaviour. In this thesis, a systematic approach and methodology are followed towards investigating the influence of position of hidden layer on the functionality and behaviour of ANNs. This will directly impact the practical applications of DNNs, and other deep learning based research by providing a method to add or remove layers based on a systematic approach rather than trial and error. The research will answer some the key questions in neural network and deep learning by

providing an insight of the representation of weights and learning the relationship between input features and neural network topology and weights.

- **Deep Learning:** This research will provide the first insights as to how neural networks are able to learn discrete features.

- **Exposing Deep Representations:** Identifying how features are represented in weights (knowledge) will provide a new direction in transfer learning. This work delves and exposes the deep representations buried in the neural network weights. This will allow the segregation of significant features and help identify their contribution to overall accuracy.

- **Neural Network Knowledge Models:** Identifying what exactly constitutes the knowledge in a neural network provides an insight into the DNN functionality. This discovery will establish a profound relationship between input features (feature components) and DNN weights (knowledge components).

- **Deep Transfer Learning:** The systematic approach for extracting transferable models developed in this research will contribute to improve the performance in deep transfer learning. This research, in turn, assists the development of approaches to improving deep transfer learning which may result in reducing the time needed for neural network training.

- **Identifying the importance of a layer:** Exploring the process of transformation of features from one layer to another layer will provide information on the importance of each layer. It will help in knowing the importance and contribution of each layer in terms of overall accuracy. This helps to address the topological dependencies and has the potential to help estimate accuracy fluctuations based on adding or removing layers.

1.11. Thesis Organisation

This thesis is organised as follows and the structure is shown in the **Figure 1-1.**

Chapter 2: Knowledge Discovery in Deep Neural Networks

This chapter is divided into two parts. The first part reviews the known available literature to explore the methods of feature extraction using both statistical and ANN based approaches. This is followed by examining the layers and weights of a DNN to understand what exactly has been learnt and how this knowledge exists in the DNNs. The second part highlights various categories of feature representations in the literature related to learning representations and knowledge transfer in line with the research focus. This is followed by defining deep learning and introducing major deep architectures. This chapter also presents knowledge representation and transfer learning aspects available in the literature.

The outcome of this chapter is identification of research gaps, the grey area that has been not yet explored which serves as the motivation for this research.

Chapter 3: Preliminary Investigations

This chapter presents an initial hypothesis on identifying the importance of a layer towards exploring the layer with necessary knowledge.

To investigate the feasibility of the research goals, it is necessary to identify the importance of various layers and their impact. An initial hypothesis is proposed stating that the middle layer(s) is significant and possesses all knowledge. The hypothesis is evaluated through a series of experiments, and the results suggest the existence of transferable knowledge components.

Chapter 4: Hypothesis and Research Approach

This chapter proposes the main hypothesis categorised into two hypotheses based on the research gap in the literature (Chapter 2) and the findings of initial experiments (Chapter 3). This chapter works towards establishing an appropriate research method for answering the research questions posed in Chapter 1 in line with the proposed hypothesis. The detailed acceptance criteria and evaluation criteria are laid-out for testing the proposed hypothesis. The research framework is illustrated in the **Figure 1-1**. The research framework gives an outline of investigation and proposed approach towards providing a feasible and reliable criterion for solving the research problem.

Chapter 5: Transferable Knowledge Component Model

This chapter proposes a Transferable Knowledge Component model by extracting feature components from the weights of the hidden layer(s). The proposed model is expressed in the

form of mathematical model and the relationship between the hypothesis and the proposed model is established. Initial experiments are performed, and the results suggest that the proposed transferable knowledge component model is successfully implemented, and the Blossom Effect could be evaluated.

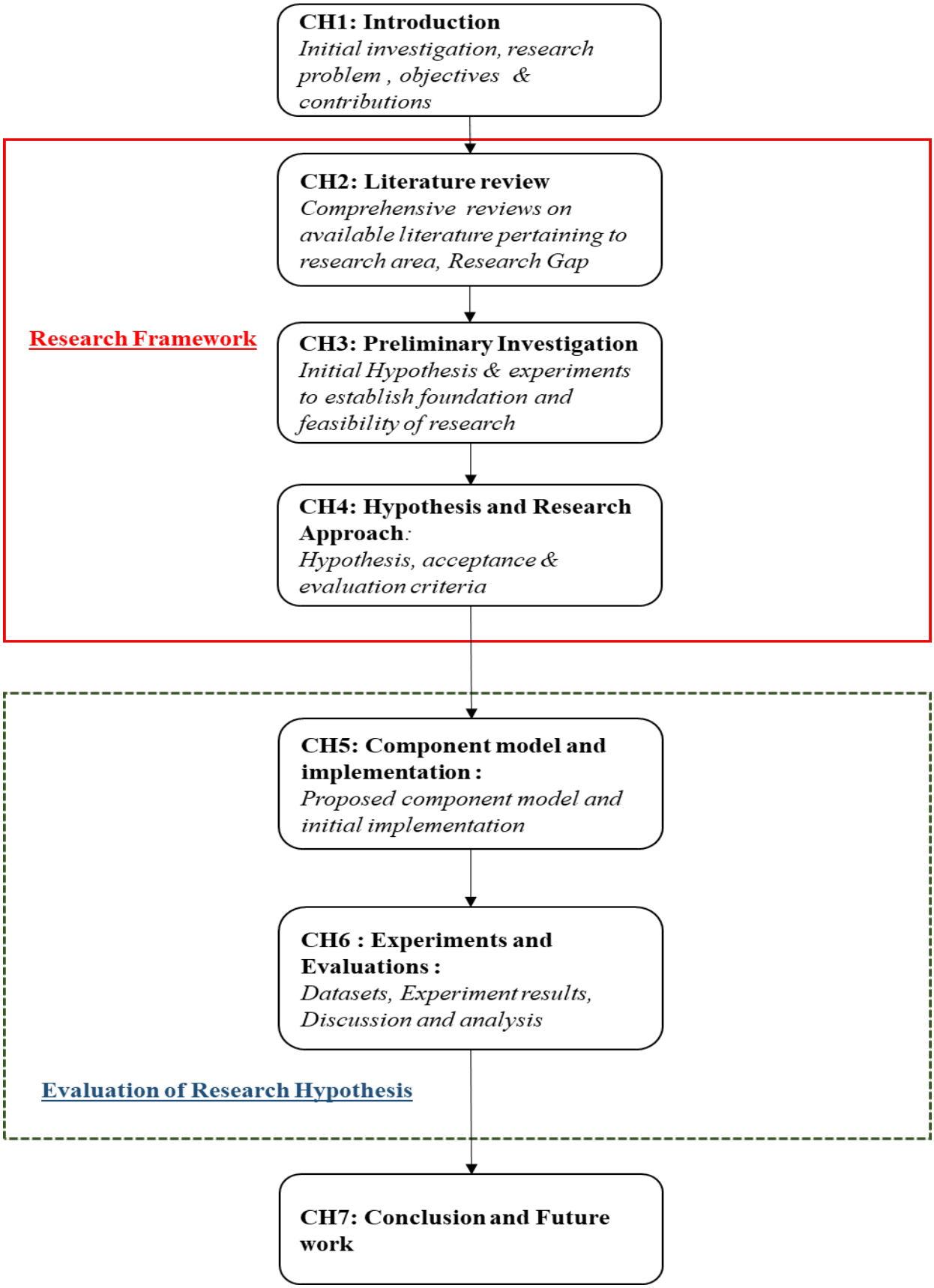


Figure 1-1: Thesis Framework and organization: Flow chart indicating the sequence of chapters in the thesis and their purpose

Chapter 6: Experiment Results and Evaluations

This chapter provides a comprehensive experimental evaluation of proposed Transferable Knowledge Component model. This chapter starts by presenting the details of various datasets used for the experiments. The experiments are carried out using a variety of deep architectures and strategies to eliminate any possible bias. The second section details the experiment results used for the proposed approach followed by a reconciliation and analysis of the results in the third section. This chapter concludes with the details on how research objectives are met based on theoretical and experimental evidence. The proposed hypotheses are tested and found to be true. The experiment results support and validate the proposed Blossom Effect.

Chapter 7: Conclusion and Future Work

This chapter presents conclusions obtained from this research and the direction for future work.

Chapter 2 : Knowledge Discovery in Deep Neural Networks

2.1. Introduction

This chapter presents an investigation of the literature on feature learning and knowledge discovery in ANNs followed by a deep exploration on representation learning which is the key factor for the success of deep learning. This chapter also presents the mechanism of feature processing that is responsible for knowledge extraction. Further, this chapter investigates literature pertaining to deep learning's ability to learn underlying representations present in the input that are spread into the neural network weights. This chapter concludes with presenting a review on traditional transfer learning approaches and their variants.

This chapter is divided into two parts. Part I introduces different aspects that exist in the literature associated with features, feature construction, and feature extraction to present a clear idea on what a feature is, and the characteristics of different types of features. Since features have highest importance in deep learning, it is necessary to know various aspects of features in

order to assess the transformation of different types of input features through neural network layers.

This is followed by an account of statistical feature extraction approaches including PCA, FA, and ML based ANN approaches. Part II presents categories of significant priors (features) extracted from the literature followed by a discussion that provides insight into the deep learning/deep architectures in the literature. Three main types of deep architectures namely DNN, DBN and DAE are briefly explained in this chapter.
The necessity and the justification for the research undertaken is identified through critical review of literature on knowledge representation and transfer learning with deep architectures in line with the proposed research question.

Part I: Feature Extraction Approaches

The efficiency of machine learning tasks is based on the learning capability of the classifier [16]. Typically, a classifier is trained to learn the characteristics of the data to effectively segregate the datasets into different classes for classification, clustering, regression, and pattern recognition, and identification problems. A feature is commonly recognised as single attribute or a set of input attributes or variables [78]. The terms property, attribute and feature are often used interchangeably. However, there is a subtle difference between these three terms. A property is the characteristic of an object or variable whereas an attribute is the additional information apart from standard characteristics. The most common definition of a feature is *"an attribute which is unique and individually differentiable."* However, this may not be true for each case. Sometimes a feature can be a single attribute in a dataset, but not always, as it cannot be generalised as proposed in some of the earlier literature [79]. An attribute is simply a variable and often treated as raw data, whereas it becomes a feature when 'processed'.

The widely accepted definition of a feature is proposed by Christopher M Bishop who states that:

"a feature is a measurable individual property." [80]

In some cases, discrete features (may be low-level features) are sometimes grouped together to form a high-level feature. If an attribute is influential enough to designate a class to the record in the dataset, it can be considered a feature. In this research, a feature is considered as

"an attribute or group of attributes that constitute a characteristic property or set of properties which is unique, measurable, and individually differentiable. A feature may exist as a single attribute or set of attributes grouped together."

In other words,

"a feature is a representation of related attributes (data) with underlying similarities."

In the literature examined for this research, the majority of ML approaches have treated the terms features and representations as interchangeable and therefore, often not differentiated sufficiently. However, it is important to realise that a representation is a property or characteristic of a feature that differentiates it from other features. For instance, a binary representation of decimal 22 is 10110 where 22 represents a value, which is characterised by a binary representation of 10110 with each binary digit with its own value determined by rules of the binary system. To study and understand the features in a dataset, it is equally important to explore and expose the underlying representations that constitute the nature of a feature. The underlying or hidden representations consist of the individual values that represent a particular characteristic and constitute the features when grouped together. In other words, a feature is a composition of low-level characteristic traits, to be specific, appearances at discrete level. According to literature, a feature can be either isolated or overlapping.

Isolated Features: A feature can be considered as an isolated feature when it does not influence any other category, or labelled data. [81]. In other words, an isolated feature has zero commonalities in the feature model. It is important to note that an isolated feature may be non-influential by itself but may be part of a feature along with multiple attributes.

Overlapping Features: When multiple attributes constitute a feature, there is always a possibility of the same set of attributes existing in more than one feature. These common attributes themselves constitute an overlapping feature.

Features may also be classified based on the structure of the input data, typically defined by the domain or the application. For instance, the structure of an image dataset is different to that of speech and an image feature cannot be compared to that of a speech feature as the

representations are different. But it is interesting to observe how this difference is represented in neural network weights which is discussed in this thesis. The research on changes in the weights for any alterations in the input features will provide an opportunity to identify the relationship between input features and neural network weights.

There are two significant categories of features used in pattern recognition: high-level features and low-level features. A high-level feature is a humanly readable feature that is important by itself. A low-level feature is a discrete feature that may exist as a fundamental block and can be decoded only through ML algorithms. A high-level feature is a reconstruction of a set of low-level features.

A pattern recognition task, like classification, is the process of reducing the gap between high-level and low-level features so that ML algorithm can precisely relate the low-level features with high-level features. The accuracy of ML algorithms depends on how the efficiently the low-level features are learnt by the algorithm through which the high-level features are classified. The success of deep learning is also attributed to this particular aspect of learning discrete features at the deepest level [3, 15, 17].

2.2. Feature Construction, Selection and Extraction

The fundamental approaches in ML and pattern recognition that involve features and manipulation of features are Feature Selection, Feature Extraction, and Feature Construction. Feature selection is the process of selecting a subset of features from the original feature-set without any manipulation or processing [78]. Feature extraction and construction often create new feature(s) from the existing features by combining, isolating, and cleansing of original features [82]. The importance of a feature may be determined by its relevance or redundancy, and when combined, they determine the efficiency of a feature [83, 84]. Features may also be categorized based on their relevance in the feature-set as (i) irrelevant, (ii) redundant, (iii) weakly relevant but not redundant, and (iv) relevant [84]. This section presents the details of various feature processing approaches available in literature.

Feature construction involves building feature subsets from the existing features or attributes to improve the efficiency of classification [85]. Combining a subset of features to construct a super-feature is a form of dimensionality reduction. Reducing dimensionality has proven efficient in various applications with high dimensional datasets like gene expression dataset

[86]. However, a 2017 survey on feature selection states that there is only a nominal difference between dimensionality reduction and feature selection which is debatable [87]. Feature construction can also be used for standardising the feature building process to achieve uniformity in the features. It is not always feasible to depend on hand-crafted features in the case of datasets with high volume and many dimensions. Feature construction is also used for converting multi modularity data to single modularity in health informatics [88]. Modularity of features often refer to features that work individually as well as in combination of multiple features to evolve a new feature. Automating the process of feature construction was also proven efficient and been applied on various datasets across multiple domains [89, 90]. The main application of feature construction is prediction. The automated feature construction approaches surpassed the handcrafted features for object recognition particularly for segregating objects in image and video datasets [88-92]. Another important application of feature construction is combining multiple cross domain features to construct a generalised feature-set [93].

Feature selection is the process of selecting relevant attributes in the input data that represent a feature or set of features [85]. Feature selection may also be defined as identifying features that have less influence and has minimal effect on ML tasks [94]. Therefore, feature selection can sometimes be considered as feature elimination, i.e., eliminating irrelevant attributes. The relevance of attributes can be determined by identifying their dependency, for instance, in determining a class in a classification problem [74]. Selecting appropriate features is substantially useful for reducing training times, data storage, data visualization and presentation [94, 95]. Feature selection also helps to understand the importance and influence of features and to optimise training and improve the efficiency of pattern recognition tasks [96].

For the input dataset *D* with *M* samples with the feature set *X* consisting of *N* features,

$$X = \{x_i\}, i = 1 \ldots . N \tag{1}$$

the target classifier *c*, the feature selection aims to find the subspace/subset S^n with *n* features from observations S^N such that S^n that achieves a minimum classification error for *c*.

Feature selection approaches may also be classified depending on the type of search mechanism applied [72, 74, 94, 97-100]. Considering the type of search strategy, feature selection approaches can be either wrapper based, filter based, or embedded [72, 74, 94, 97-100]

Another classification [74] separates feature selection approaches into two categories, label based, and search based [95-98, 101]. Label based feature selection approaches uses machine learning (supervised, unsupervised, semi-supervised) whereas search based approaches are (i) wrapper, (ii) filter, or (iii) embedded [100]. However, there is an overlap between the labelled and search-based criteria. As a result, the categorisation followed in this thesis is supervised, unsupervised, and semi-supervised [16, 17].

Wrapper methods are based on evaluating a subset of features on a predefined model [102]. Once the model is trained with the subset, the importance or influence of features is evaluated by adding and/or removing features and comparing the accuracy with the previous results. Since the approach is like a sequential search which needs to be tested for each and every attribute/feature, it is computationally expensive. Moreover, the predictor/classifier is based on predefined model and in this case it is similar to a neural network blackbox [74].

Implementing a wrapper method is not complex since it is based on a predefined model with labels which is similar to supervised learning. When no labelled data is available, the attributes are grouped based on no criteria, i.e., grouped randomly. This is similar to implementing unsupervised learning for clustering with no guaranteed qualitative results [103]. However, it is complex and more time-consuming to identify ‘good’ and ‘bad’ clusters since there is no direct approach to evaluate the quality of the selected attributes (clusters) other than using an objective function with predefined criteria to differentiate the attributes.

The feature selection process in wrapper methods is based on either sequential search algorithms or heuristic search algorithms. Sequential Feature Selection (SFS) is the most basic approach which starts with an empty subset and adds one feature at a time in the later stages. This process is iterative and each addition is evaluated against a predefined model [104, 105]. Other sequential methods are Sequential Floating Forward Selection (SFFS), Sequential Backward Search (SBS) and Sequential Backward Floating Selection (SBFS) [74, 104, 106] .

Wrapper methods can also be implemented using heuristic search approaches since sequential search is operationally expensive [74]. Hence, nature inspired approaches like genetic algorithms and particle swarm optimisation are also used for search in the wrapper-based feature selection approaches.

The second category of feature selection is filter methods that consider the feature selection process as a rank based scenario for listing features based on their scores [107, 108]. Filter based feature selection is based on evaluating features using classifiers. Initially, the features are ranked based on a criterion followed by an evaluation. The feature can be ranked using one of the two methods namely univariate or multivariate ranking. In the univariate approach, the ranking of a feature is independent of search space and the process is performed on one feature at a time. This makes a univariate process very stringent. Multivariate approach evaluates the features in groups or batches which allows it to incorporate redundant features and handle them efficiently. This is followed by the second stage which is an evaluation process using a classifier. The ranked features obtained from the first stage are sorted in ascending order based on scores. The features with highest ranks are selected to induce classifiers for evaluation. There are several successful filter methods for feature selection approaches such as Fisher Score [109, 110], lapsian score [111], single [112] and ensemble learners [102], graph theory, and other models [113, 114].

Embedded methods use a linear model using L1 regression by adding a penalty based on the complexity [115]. The importance of attributes is categorised based on degree of overfitting which is controlled by adding additional bias. The penalty is directly used as the cost function. This is an intrinsic way of selecting features through the L1 vector [116]. SVM based approaches are also used for embedded feature selection as an alternative to regression [117].

There are several ML approaches for feature selection. These are categorised in this thesis based on their learning scenario (supervised, unsupervised or semi-supervised) as mentioned earlier in this chapter. The ML based feature selection approaches are often domain or dataset or problem specific, which makes it difficult to represent in a generic categorisation [118-122]. The principle segregation feature selection based on filter, wrapper, and embedded methods is apt, widely accepted, and supported in literature.

The purpose of feature extraction is often confined to dimensionality reduction for optimising the learning process. However, feature extraction involves identifying and extracting characteristic features in the form of input attributes that influence the process of learning or other ML problems that the model is designed for. This includes the transformation of input into a format that represents a collective information referred as knowledge.

Consider a labelled dataset *D* with *x* as data, *y* being a label and *C* is the number of classes associated with data. Dataset *D* is domain specific which belong to a particular domain *X*, such that,

$$(x_1, y_1), \dots, \dots, \dots (x_{n,} y_n) \in X \; for \; \{c_1, c_2 \dots \dots \dots, c_n\} \tag{2}$$

The aim of the task is to predict (*p*) for an unknown value of *x* with a label *y*. The label *y* is associated with one of the classes in *C* and the challenging task is to identify the closest possible association.

For identifying the $y \in \{c_1, c_2 \dots \dots \dots, c_n\}$ or the closest possible value for *y*, it is important to identify the pattern of *y* which is similar to a class c_y but quite distinct to other classes in the dataset. In other words, identifying the dissimilarity is as important as identifying the similarity. Therefore, the feature extractor can be defined as:

$$J = \frac{1}{2} \sum_{i=1}^{n} p_i p_{i+1} d\,(c_i, c_{i+1}) \tag{3}$$

the distinction of the classes is based on distance between the mean values,

$$d(c_i, c_{i+1}) = d(m_{i,} m_{i+1}) \tag{4}$$

where *m* denotes the mean vector of *C*.

The main aim of feature extraction is to identify, and map measurement space *M* to the feature space *F* with either linear mapping or non-linear mapping.

The aim of linear mapping is to maximize *J(M)* for a linear mapping of individual *M* with *X*. The linear feature extractor *J(M)* is maximised when *M* achieves largest Eigen value for scatter distances between each class that belongs to *C*.

Through the details presented in the sections, feature extraction can be defined as

"a process of identifying, highlighting, and segregating to create a representation of interesting characteristics in the data"

2.3. Statistical Approaches for Feature Extraction

In the literature, there are several successful and efficient approaches for extracting features. The main purpose of these approaches is to identify the characteristic features that help to optimise the learning process. Feature extraction is also used to categorise and group features with similar or related characteristics. The main purpose of feature extraction is pattern recognition, recognised and emphasized in ML and its applications as early as the 1980s.

There are numerous feature selection and extraction approaches proposed ever since - as early as late 1800s. The feature extraction/selection approaches can be broadly classified into two categories; linear and non-linear, based on the dimensionality of the data. Typically, if the data is represented in more than two dimensions, it is very difficult to interpret which makes it high dimensional. Linear approaches are based on the assumption that data are present in the linear subspace at lower dimensionality and can be projected linearly. Some linear dimensionality reduction techniques include Principal Component Analysis (PCA), Linear regression, Factor Analysis (FA), Single Value Decomposition (SVD) and Independent Component Analysis (ICA).

In non-linear methods, the data is projected in the form of a non-linear manifold with the feature space. Some of the non-linear methods are Sammon's mapping, curvilinear component analysis, self-organising maps, diffusion maps, and ANNs. The machine learning based feature

selection/extraction approaches are uniquely distinctive to other feature selection/extraction approaches.

One of the objectives of this research is to explore and extract feature components from the neural network weights through feature extraction approaches. Since, the mechanism of extraction of feature components from neural network weights is attempted for the first time, the literature could not assist in any experimental method or procedure. Due to this, it would be reasonable to start with an existing component extraction approach which is popular and widely accepted, yet simple to analyse. Principal Component Analysis (PCA) is academically acclaimed and statistically proven approach for extracting components from dataset which makes it the first choice. This section presents an analysis of PCA and its implementation. This section also presents a description and review of Factor Analysis (FA) which is used for analysing correlation between components. PCA is considered as a special case of FA. This section also attempts to examine the capability of autoencoders which are used for feature extraction through dimensionality reduction. The importance of domain specific feature extraction and its application for evaluation is presented in the research methodology (Chapter 4).

2.3.1. Principal Component Analysis (PCA)

PCA is a statistical approach for dimensionality reduction by projecting the correlated observations into a non correlated point called principal components [123, 124]. PCA can also be described as an algorithm that reduces the dimensionality of a dataset by identifying the directions called principle components [125]. PCA is a linear transformation approach that uses orthogonal transformation to identify principal components such that the first component comprises of the variables or attributes with maximum variance followed by the components with variances in reducing order. PCA is based on the statistical influence of the features in the feature vector. PCA evaluates a feature based on its statistical dependence to eliminate least discriminative features among the feature-set. For any feature extraction approaches, the influence of one feature on another feature cannot be ignored. PCA mainly uses either an Eigen vector or a single-valued based decomposition according to the type of data. PCA, in which the principal components are used as transformed features, is a widely used method for feature selection and feature extraction [126].

The capability of PCA to automatically determine the number of principle components is purely based on variance and is considered to be main reason for its wide acceptance. This strength of PCA eliminates any bias since the process is completely dependent on the dataset and the type of projection used. The results from the PCA are analysed based on the component scores and weights or loading.

In PCA, the input variables are projected into an output feature space with a new coordinate system. These transformed variable values associate with a data point in the newly projected space. The position of the variables is based on the variance. In the new coordinate system, PCA transforms the variables such that the variables with maximum variance are placed in the first coordinate (first component) followed by other components based on variance (component score).

The component score in the projected space is achieved by multiplying the original variable with the weights calculated based on variance, type of rotation, and other parameters. The transformed variables are grouped into components based on their component score. Traditionally, PCA results are presented as a linear combination on the transformation of original variables. In case of variables with different measuring units, some unwanted features are also included in the results. As variance is strictly based on units of single measurement (common scale) of the input covariance matrix, this challenges the principle theory behind constitution of principal components, since the key aspect of PCA component categorisation is variance [127]. If the unit of measurement is different or changes for one or more variables, this will lead to a new change of scale/component scores and data points. To overcome this, the input variables are standardized so that a common measurement and scale can be maintained [127].

Rotation methods play a vital role in determining the data points in the projection space making them highly influential on PCA results. There are six rotation methods that can be applied for PCA. The five types of rotation methods of PCA are namely Varimax, Quaritmax, Equamax, Direct Oblimin and Promax. Sometimes, PCA can be used without any rotation method to observe the generic PCA results (non-rotational) for preliminary analysis which is the sixth type of rotation method (No rotation).

Another important statistical method is Factor Analysis (FA) which is often considered as a domain specific PCA. In FA, the Eigen values are calculated slightly differently based on domain specific assumption about the structure of data. Since its first invention in 1901, PCA is used to solve various types of problems and has been the most popular statistical approach that has challenged many ML algorithms. The primary application of PCA is classification/prediction [124, 128]. PCA is also used to visualise the data points (high dimensional) in a low dimensional space for clear representation using special tools [129-131].

However, the most powerful aspect of PCA is its capability to project the values of the variables clearly, which guides in analysing the hidden patterns in data. PCA's unique capabilities enable to identify, explore and expose the underlying relationships between variables [132]. The application of PCA includes face detection [133], anomaly detection [134], e-learning [135], healthcare [136], speech recognition [137] and many more. Attribute selection or extraction in multivariate attribute datasets is complex and often involves multiple criteria. PCA is also a top contender for working on multivariate variable datasets [138]. PCA is most successful and widely accepted approach for exploring attributes and identifying the relationships, and dependencies. PCA is proven efficient in extracting features from image data [139], speech data [140], Real-time (temporal) data [141], Geographical (Global Positioning System or GPS) data [142] and Gene expression data [143, 144]. It is noteworthy that PCA is used extensively to identify and examine the correlation between attributes [145, 146].

2.3.2. Factor Analysis (FA)

Common Factor Analysis (CFA) or simply Factor Analysis (FA) is a statistical approach for attaining knowledge on the variance-based relationship among correlated variables from the observed variables. The variability among the correlated variables is measured with respect to unobserved variables. This will help to expose and present the underlying factors that are strongly influential among correlated variables which are potentially not been observed to the necessary extent. FA is based on theoretical factors which propose a formal model on the observed variable. FA is similar to PCA, but it is not identical to PCA [147].

FA is important because of its ability to categorise factors as a superset of attributes based on underlying relationships. The correlation aspect of FA is powerful and reliable particularly for overlapping features reflected by highly correlated variables.

The PCA is principally based on variance and is used to extract linearly separable components of the variables whereas FA is based on covariance between multiple components called factors. In other words, the factors in FA are formed based on the linear combination of underlying variables that maximise the shared portions of variance. The extraction of a PCA component is based on the values and the starting points of the algorithms used for extracting components whereas FA optimisation methods are based only on the values. PCA and FA complement each other based on a common rotation method. For instance, FA and PCA produce the same results when the extraction method is based on maximum likelihood. The most widely used IBM's statistical software SPSS [148, 149] treats PCA as a simple case of FA and the results are varied based on the rotation method used. In contrast, the statistical package 'R' primarily uses FA and treats PCA as a simple case of FA.

PCA is usually regarded as component based on the uniqueness of attributes, whereas FA uses both uniqueness and commonality. FA models are widely accepted, particularly for datasets with overlapping variables since FA overtly accounts for errors in the measurements which is not the case for PCA.

In case of datasets with minimal overlapping of features as well as with attributes based on pure covariance, PCA could be used to produce components by dimensionality reduction. FA is used for much complex datasets where the variables are overlapping and highly correlated with underlying relationships which could not be seen merely by looking at variance.

2.3.3. Extracting Components

Implementing statistical methods such as clustering is one possible way of analysing the weights of individual layers of a DNN. Further, to help determine the association of weights and the relationships between features, PCA and FA can be used. These statistical methods can be used to propose a component model of extracted weights that can be associated with features. Therefore, in this research, PCA and FA are explored as a means of investigating the weights in each layer and for the extraction of feature components.

A rotation for factors is defined as *"performing arithmetic* [analysis] *to obtain a new set of factors"* [150]. Rotation methods can be categorised into two types: orthogonal and oblique. Orthogonal rotation assumes that the factors are uncorrelated whereas oblique approaches treat the factors as correlated. The details of rotation methods used to explore PCA and FA are

presented in **Table 2-1** which describes the characteristics of each rotation methods and type of rotation. The adaptation of rotation methods is important since the proposed component extraction is based on PCA / FA and may require a selection of rotation method.

The process of selecting a rotation method is a critical aspect of component analysis. Oblique methods are typically used to identify correlation between attributes by predefining the number of components to extract [151, 152]. If the correlation exceeds 0.32, the rotation can be considered as being orthogonal [153]. Though correlation between the attributes is important, it is not the only important factor particularly in analysing the characteristics of data at the initial stages [154]. The experiments presented in Chapter 6, therefore, involve a variety of datasets with correlated and non-correlated attributes.

Table 2-1: List of rotation methods and their characteristics for Principal Component Analysis (PCA) and Factor Analysis (FA). The description provides the characteristics of the rotation method which will help in analysing the experiment results and to explore the relevance of rotation methods.

Serial No.	Rotation Method	Description	Orthogonal/ Oblique	Correlated
1	No Rotation	No rotation method	N/A	N/A
2	Varimax	Actual coordinate – Not changed. Rotated to align with those coordinates. maximizes the sum of the variances of the squared loadings	Orthogonal	No
3	Quaritmax	orthogonal alternative which minimizes the number of factors		
4	Equamax	conciliation of varimax and Quaritmax		
5	Direct Oblimin	Similar to varimax and tends to produce varimax like factors which not orthogonal but are oblique		

6	Promax	Produces group factors similar to Oblimin typically used very large factorings	Oblique	Yes

The two important reason to choose varimax over other rotation methods are:

(i) Other orientation methods are not applied as they are not reliable [121].

(ii) Varimax is most popular, reliable and widely used rotation method [153]

Varimax, an orthogonal method, is more effective across a variety of datasets when compared to oblique methods [153]. Varimax is capable of dealing with correlated attribute datasets and is better than oblique methods for non-correlated datasets. Thus, varimax is considered to be more generalised approach than other rotation methods. The majority implementations of PCA and FA use varimax rotation method [153].

2.4. ANNs and Feature Extraction: Autoencoders

ANNs with one or two hidden layers are attributed to the success of AI approaches for classification based on features [78]. However, there was minimal success for ANNs for classification particularly when attempting to classify large datasets with complex features since ANNs are unable to learn the features at discrete levels. The features in shallow ANNs (ANNs with one or two hidden layers) are present in hidden layers in a condensed form since there are limited number of layers. The majority of feature extraction approaches associated with ANNs involve single or multivariate data projection [155]. One layered ANNs (based on multilayer perceptron) often use a nonlinear input feature in high dimensional space projected into an abstract low dimensional feature space [156]. Classical ANNs employs a wide variety of data projection algorithms and techniques for feature extraction [157-163].

Some of the initial works on pattern recognition and classification have used ANNs for feature extraction or dimensionality reduction [164]. Feed-forward ANN with one hidden layer has become a widely accepted method for extracting sets of features [131]. This method uses a form of dimensionality reduction by optimising number of hidden nodes [165]. There are some noted works that use a data projection based approach for feature identification and extraction [166, 167]. One of the earlier works in this area uses Sammon's nonlinear projection based

ANN (SAMANN) [155]. Sammon's projection tries to preserve the inter-pattern distances and, thus, lacks the capability to use or introduce new data to the ANNs for projection process. In Sammon's projection, for each *d*-dimensional pattern, there exists *n* patterns when *d* is projected into a space with *m*-dimensional, where $m < d$. The mapping difference that occurs between *m* and *d* is called the Sammon's distance.

The majority of approaches reported in the literature however use Euclidean distance for projection and inter-pattern distance estimation. The projection is performed from high dimensional space to a lower dimensionality without disturbing inter-pattern distances. The main disadvantage of Euclidean-based approaches is lack of a mapping function between the original space and the projected space. The limitations of both SAMMAN and Euclidean distance approaches can be addressed by using a BP based ANN with SAMMAN projection. This approach is more generalised and can cope with different datasets and allows for the inclusion of new data in the projection process.

An autoencoder is a special type of ANN for learning representations from unlabelled data and operate by reconstructing the input. In other words, the principle task of an autoencoder is to reproduce the input as output, i.e., to copy the input to output. Kohonen's auto-associator is the earliest known linear network for reproducing input as output [168, 169]. The autoencoder is an unsupervised learner since it requires only input (unlabelled) data to learn the representations.

First proposed by Rumelhart et al. in 1989 [12], the primary functionality of an autoencoder is dimensionality reduction. Autoencoders are knowns to be a bottleneck in a network since there are typically fewer hidden nodes than input nodes [170]. Autoencoders are now predominantly used for feature learning.

Autoencoder consists of an encoder function that creates a new coded representation of input. This representation is presented as a separate encoder layer. This is followed by a hidden layer (middle layer) that filters the identifiers to a required number, representing the essence of the input. This is followed by a decoder layer that fragments the features which are used to reconstruct the input. This process is pictorially represented in **Figure 2-1**.

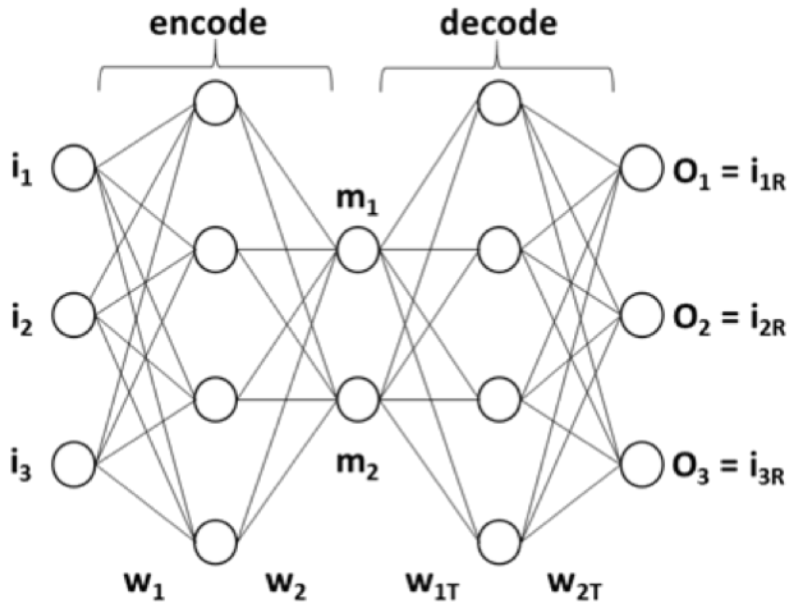


Figure 2-1: The representation of an autoencoder with encoding and decoding layers with middle layer represented by m1 and m2. The input i is passed through encoding layer and into the middle layer (dimensionality reduction) followed by the decoding layer to reconstruct the input as i_R.

An autoencoder tries to learn a function h_W, $b(i) \approx i$ for reconstruction so as to minimise the mean square difference:

$$L\,(i,j) = \sum (i - h_w\,.\,b(i))^2 \tag{5}$$

where *i* is the input data, and *j* is the reconstruction value.

Typically, for a sigmoid activation function, a cross-entropy loss reconstruction function is used. The reconstruction of weights is done by optimising the weights in the hidden layer that represents the input encoded using encoders. The significant application of autoencoder is data compression due to its ability to represent input in a compressed format. When the number of hidden nodes is less than the number of input nodes, the hidden nodes represent a compressed format. This compressed format enables an autoencoder to be used for data compression. In some cases like speaker identification, autoencoders enable the extraction of important features that represent majority of the input representations [171]. For compressing image representations, an autoencoder is a natural choice. When an image of 1024 x 1024 is

compressed and learnt as 256 x 256, it is easy to handle the data in terms of size, memory and speed.

One notable application by Kramer [172] was the use of a nonlinear PCA (NPCA) approach to construct an auto-associative neural network. This approach was then further enhanced, as h-NPCA, by introducing non-linear nodes into an autoencoder with hierarchical training [173].

One of the earlier applications of feature extraction using an autoencoder was reported by Cottrell in 1991 and was employed for facial recognition through extracting facial features from the image [168]. The experiments were carried out by reducing a 512 x 512 pixel resolution image into 64 x 64 pixel by decreasing the number of hidden nodes. This work uses associative network architecture that creates an association between two ANNs: one a regular network and the other, a compression network. The association is carried out by using an ensemble of networks, a face compression network, a compression network and a network with hidden nodes in the middle layer to learn the compressed features. This approach has given way to later implementations particularly in deep learning presented in the next section.

Shallow networks can be used only as non-linear modelling systems with numeric data. Processing high dimensional data with ANNs has two major issues. Firstly, ANNs incur the curse of dimensionality for large amounts of data. High dimensionality of data makes it impossible to attain efficient results. Secondly, due to a smaller number of hidden layers, the entire set of features are condensed in one or two hidden layers with extremely overlapping features. Increasing the number of hidden layers to address this issue has resulted in slowing down the training process, particularly when trained using BP. To mitigate this, a separate feature extractor is used to extract and fine-tune features before feeding them into the classifier. Firstly, feature extraction approaches are used for extracting the relevant features followed by manually hand crafting them to ease the learning mechanism [174]. However, this process is quite time-consuming in itself, and is not robust. Consequently, it is practically impossible to implement feature extraction for large datasets. However, for deep learning this is not necessarily the case since deep learning learns the representations hierarchically rather than features as a whole. Feature extraction is not necessary for deep learning mechanism as deep learning learns representations and avoids the requirement of feature extraction as well as fine-tuning them.

2.5. Summary

This section introduced the standard definitions of features and knowledge, followed by a discussion of the statistical and neural network based feature extraction approaches reported in the literature. The exploration of literature for the theoretical definitions of features and feature extraction will help in modelling a feature extraction approach using neural network weights. Considering the fact that there is no sufficient literature on identifying features in the neural network weights it is important to acquire sufficient background on features and feature extraction approaches that are successfully implemented. The information on types of features and their contextual definitions will guide the process of design and implementing component model based on features.

Continuing from here, the next section explores the literature on various approaches for knowledge discovery and transfer learning in deep neural networks.

Part II: Learning Representations & Knowledge Discovery

ML is completely reliant on how clearly the features are learnt through training. Clarity of features is often achieved by manually fine-tuning the features to create a clear distinction among each feature, which helps classifier to identify the class that is associated with the feature [175]. This fine-tuning of features termed as 'feature engineering' is the key for achieving accurate results. Feature engineering is quite costly and needs human intelligence and prior knowledge about the data. Traditional algorithms are greatly reliant on feature engineering. In other words, to be efficient, ML algorithms expect features to be hand-crafted or fine-tuned before submitting them to classifiers [176].

Shallow architecture based algorithms like ANNs, SVMs [177], and other kernel algorithms are unable to handle complex and high volumes of data and are proven inaccurate due to the lack of efficient training mechanism. For ANNs, increasing number of hidden layers will solve the issue of learning representations through multiple layers [178]. The main hindrance in this research direction was determining methods of training these ANNs and provide them with clean and efficient features which is possible only by manually handcrafting the features with human intervention. To reduce this dependency on human involvement, it is, therefore necessary to work towards learning algorithms that can learn features from the data itself. The efficiency can further be improved by making the training algorithms learn the representations by themselves.

The input features are spread across attributes of the dataset in the form of one or more attributes [179]. These features are leant by the DNNs and are represented in the form of underlying representations in weights across the layers of DNN. The representations are patterns or characteristics that are present in features. The uniqueness of a particular representation depends on the uniqueness of the feature. For instance, consider a dataset of three attributes out of which one is unique and associated with only one particular class. The DNN identifies the unique class based upon unique attribute associated with the class. Since DNN learns through weights and its internal representations, the attribute certainly needs to be represented differently in an exclusive format. The more exclusive and clear the representation is, the clearer DNN can learn [16]. Therefore, learning representation is the key aspect for DNN accuracy.

Representation learning or feature learning is the process of determining the representation that constitute features [16]. In other words, feature learning is the process of linking low-level features extracted by ML algorithms to high-level features that guide the classifier. The efficiency of feature-learning depends on the feature extraction capability of the ML algorithm used. Representation learning is what resides inside the human cognition to know, learn and experience various real-world scenarios. To start with, human learning is based on extracting and learning core representations and using them when required to perform various identification and recognition tasks [16]. The core representations learnt through human cognition and experience are often considered as core knowledge that exists across multiple task and domains. The core knowledge is sometimes referred as common or generic knowledge that can be used across different tasks. For instance, a person who learns how to play a piano can utilize the fast finger movement for typing. Here the finger skill is generic and not task specific whereas as the implementation may be task specific.

Traditional ML algorithms are task oriented and are designed particularly for a specific task or problem which makes these algorithms weaker and non-generic. Further, traditional ML approaches and methods lack the capability of learning representations at discrete levels due to technical and computation hurdles. For instance, ANN with only one hidden layer is unable to demonstrate the same capability that of DNN with multiple hidden layers, since DNN is able to learn features at a discrete level through weights present in various hidden layers which the ANN lacks. Another advantage of learning representation is that the learned representations can be used to express generic priors across multiple problems which is a common case for real-world applications to achieve near human accuracy.

Learning representations, particularly with deep learning, is proven to assist the AI systems to amalgamate or segregate features efficiently based on a problem [16]. The underlying representations or the pattern of priors are most important and influential in the learning process. These underlying ‘deep’ representations assist the learner (in this case DNN or ANN with sufficient depth) to pursue the ‘deep’ knowledge that enables the achievement of near human accuracy for various AI tasks. There are some important priors in the representation that are generic and can be used to perform a different task other than the task for which it is trained. This is particularly applicable in the case of transfer of knowledge from one DNN to another DNN where the second DNN receives the ‘learning’ (similar to experience) from first DNNs to perform a different task.

Representation learning came into focus with the success of deep learning which attained the-state-of-the-art results in various real-world implementations and applications. However, the investigation on what and how these representations are present in the form of patterns in the hidden weights has not been explored which is the significant factor of this thesis. To examine the weights for the patterns based on input features, it is necessary to explore more about how data is represented in various types of priors. The investigation of how ANN weights are impacted by the changes in the input representations requires a systematic experimental examination which is undertaken in this thesis. This also demands sufficient knowledge on underlying representations that are generic (significant priors) and can be categorised as identified by Bengio, Courville and Vincent [16]. The categorisation of significant priors based on the type of representations that exists in the data is presented in the following section.

2.6. Categories of Priors

2.6.1. Smoothness

Traditional linear models have limited success in handling complex data such as the data involved in computer vision and Natural Language Processing (NLP). These linear parametric models were initially replaced by kernel machines, but, these were limited to local generalisation [180] under the assumption that the target function is smooth to learn. Both linear models and kernel models assume this smoothness, failing to overcome “the curse of dimensionality”, since the generalisation is limited to local neighbours. Furthermore, the raw data representations particularly for complex data gives rise to many fluctuations in volume, complexity and training samples. However, the importance of linear models cannot be ignored. The combination of linear models and representation learning enables researchers to explore and expose the feature space in order to possibly improve the efficiency of AI algorithms.

2.6.2. Distributed Representations

To express the true nature of the input, representations have to be clear enough. This hints at the size of the learned representations based on numerous combinations of input attributes. Traditional clustering algorithms require *n* parameters (examples) to distinguish *n* outputs in the region or feature space since the input data cannot be expressed as distributed representations. For example, to distinguish three outputs in results, it is necessary to have at least three different parameters where each parameter is exclusively linked to a single output.

In the case of DNNs, a total number of 2^k features can be represented using distributed representations with the same number of n parameters where k is the non-zero element for sparse representations. In the case of such dense and non-sparse representations as RBMs (presented in the Chapter 2 Section 2.4), $k=n$ [181]. Clustering is generalised for distributed representations using sub-clustering or multi-clustering where representations are identified and distinguished in the form of small clusters distributed within the input or groups of clusters in parallel.

The important aspect of distributed representation is the nature and presence of representations. Some representations can be reused for multiple samples as well as associated with different regions of input. Distributed representations enable the association of individual features with multiple hidden units in the case of a single-layered ANNs, whereas in the case of non-distributed representations, the input feature is always associated with a single identifier. For instance, in clustering algorithms a feature in a non-distributed representation can only be associated with the most suitable cluster. In contrast, in a distributed representation each feature is involved in more than one concept and each concept is represented by multiple features, thus, making features mutually exclusive and also be independently verified.

2.6.3. Relational

The underlying patterns that determine a representation in abstract high-level features are interrelated since they represent a specific feature. These underlying patterns are also reflected in relating various features in a good and clear high-level representation. Further, there may exist some linear dependencies between these patterns (representations) that are reliable and definitive for identifying a (characteristic) feature.

2.6.4. Shared

The underlying patterns for a representation may be shared across multiple representations and substantially across multiple features. In real world examples, particularly in computer vision and image processing, core representations are shared across multiple features which helps in creating a generalised model.

2.6.5. Sparsity

Every representation possesses a group of dependent patterns that are most influential, and some definitive patterns are in the form of least-effective features. In an observation x, there

exists a set of features that are tolerant or insensitive to minimal changes. This tolerance can be identified through the hidden variables whose values are often flat/0 or linear/non-linear or a calculated Jacobian determinant through mapping inputs with representations.

2.6.6. Hierarchical

Groups of low-level discrete representations may constitute a high-level representation in an abstract form. These core representations are underlying representations which can only be exposed by 'deep learning.' Many real-world applications for text mining, NLP, face recognition are built in the form of a hierarchy with abstract representations at the higher levels. Learning of these low-level underlying representations (aka deep representations) is necessary to achieve high accuracy in any ML task. DNNs are able to learn these deep representations through layer-wise training and for the first time, the underlying core features that are responsible for better accuracy have been identified [3].

2.6.7. Sequential & Temporal

The nature of similar observations in sequential and temporal data is associated with a common observation value in the high dimensional feature space. This relevance of the category in a high-dimensional space can be reached through a small move. In case of diversified values, the associated value in high dimensional (feature) space is not close. In order to determine the representations for diversified categories, it is necessary to take a big leap which is guided through spatio-temporal aspects of the data. The search process for identifying associated sequential and temporal representations for a target has often proven to be costly in multiple-manifold search space.

2.6.8. Manifolds

The concentration of probability for machine learning algorithms always go around low dimensional regions in the output space. It is necessary to provide a thrust to overcome these manifolds and explore the entire data space to determine the deep representations, which to some extent, is accomplished through deep autoencoder networks.

2.6.9. Semi-supervised

In supervised learning, for an input x and a target to predict y, the characteristic factors that represent x can be used to explain the pattern of distribution of y. Henceforth, when the representations of $f(x)$ is determined, this can be useful to define and learn the function $f(x|y)$ (unsupervised) of an unknown target. Therefore, learning representations using supervised

learning may help in analysing and extracting patterns with no classes/targets. This is the core concept of deep learning where the DNN tries to learn the representations without target classes (unsupervised).

2.7. Deep Architecture Learning (Deep Learning)

Representation learning often requires multi-level or hierarchical architectures with significant depth. The deeper the architecture, the easier it is to learn representations in detail. However, it is challenging to train deep architectures, for instance, ANNs with significantly large number of layers. The training multilayer ANNs began over three decades ago. A considerable amount of progress has been made in recent decades and this development, as documented in the literature, is presented in the next section.

Deep learning is a unique algorithm that is capable of learning representation through layer-wise training of a very 'deep' topology. Deep learning provides the following three significant and important advantages over traditional learning approaches:

1) Deep Learning enables to learn features using unsupervised learning thus eliminating the requirement for training of classes.

2) Deep learning enables the extraction and immediate use of features at various intermediate levels

3) Deep learning provides the reusability of features which is enabled via empirical learning through samples.

De-noising is one of the important characteristics of representation learning via deep architectures particularly with DNNs. More abstract features can be extracted as high-level features that are more significant and contribute to the accuracy of the classification task. This de-noising process for the features has been used as a way of removing insignificant and unnecessary features and is somewhat similar to dimensionality reduction.

These aspects provide a theoretical advantage of deep learning over other learning algorithms. The next section presents a brief account of deep learning with respect to different types of deep architectures and their implementations.

2.7.1. Deep architectures

ANNs are once again popular due to the success of deep learning involving multi-layer neural networks for solving tasks that are too complex for single-layer or dual-layer neural networks.

Common problems pertaining to ANN learning mechanism are also persist in DNNs [29]. If training is too long, test results can be poor because the weights have become too specialized (overfitting). If training is too short, training results can be poor, leading to poor overall results on the full dataset (underfitting). Introducing a recalibration training dataset (i.e. training with an additional dataset after initial training) as an additional means to deal with overfitting or underfitting can lead to oscillation of weights and unlearning of initial samples [29].

To some extent, the problem of overfitting was addressed in DNNs using thinned DNNs [4]. The idea of thinning DNNs is to randomly drop units during learning to prevent unit over-adaptation. But this requires a number of different DNNs to be trained and then converged through averaging at the final stage, which results in a thinned DNN. Also, it is not clear what the implications of thinned DNNs are for data representation [29].

The number of layers of an ANN constitutes its depth. The ANN architecture is considered as 'Deep' when multiple hidden layers are used in its architecture [17]. Feed-forward ANNs with more than one layer of connections can solve more problems and be more accurate than one-layered ANN [17]. In this thesis, a 'hidden layer' in a DNN is defined as any layer of connections or units/nodes apart from those at the input and output stages. Throughout this thesis, the context determines whether we refer to hidden connections or hidden units/nodes [29]. Theoretical studies also support the statement that DNNs have the advantage of more efficient representation when compared to shallow networks and with fewer hidden units [17] [29]. Unlike ANNs, the layers of Convolutional Neural Network (CNN) have neurons arranged in three dimensions for overlapping purposes.
In 1980 Fukushima proposed Neocognitron using Convolutional Neural Networks (ConvNets) [25] [29] which served as a successful model for later works on deep architectures [182]. This first attempt at ConvNets still had an inefficient training mechanism. Fukushima's work was later improved by Lecun [22] who also proposed the theoretical concepts of deep architecture in 1998 [21]. CNN, being the first form of DNN, uses the standard BP algorithm for training, and the weights are updated according to equation 6.

$$\Delta w_{ij}\,(t+1) = \Delta w_{ij}\,(t) + \eta \frac{\delta c}{\delta w_{ij}} \tag{6}$$

where η represents the learning rate, c is the associated cost function, w_{ij} represents the weight between the units i & j *and* t represents time.

Historically, the concept of DNNs was proposed in 1989 as Convolutional Neural Networks (CNNs) without using the word 'Deep'[29]. Back Propagation (BP) was used to train CNNs and was known to be not so effective because of the limitations of BP [29]. For instance, feedback is applied only to the immediately previous layer. After the introduction of a new greedy layer-wise training, ANNs once again became popular in the form of DNNs [28]. The innovation in the research on the training mechanism of deep architectures was achieved in 2006 when Lecun, Hinton and Bengio proposed three different types of deep architectures, each with an efficient training mechanism. Lecun expanded on his earlier work on ConvNets by adding an efficient training mechanism [22]. Hinton's Deep Belief Networks (DBNs) [23] and Bengio's stacked autoencoders [28] were the other two implementations. However, the work of Jürgen Schmidhuber cannot be ignored as it is considered to be the first study that trained ANN with large number of layers well before the term DNN was used [17].

A fundamental form of deep architecture i.e. DNN, is a feed-forward ANN with more than one hidden layer that make them more efficient than a normal ANN [24]. DNNs are trained with BP by discriminative probabilistic models that calculate the difference between target outputs and actual outputs. The weights in the DNNs are updated using stochastic gradient descent defined in equation 6. For larger training sets, DNNs may be trained in multiple batches of small sizes without losing the efficiency [183]. However, it is very complex to train DNNs with many layers and many hidden units since the number of parameters to be optimized are very high.

Deep architecture is a hierarchical structure of multiple layers with each layer being self-trained to learn from the output of its preceding layer [182]. This learning process which is deep learning is based on distributed representation learning with multiple levels of representations for various layers [182]. In simple terms, each layer learns a new feature from its preceding layer which makes the learning process concrete. Thus, the learning process is hierarchical with low-level features at the bottom and high-level feature at the top with intermediate features in the middle [182]. From these features, the greedy layer-wise training mechanism enables the extraction of only those features that are useful for learning. Along with this, a pre-

unsupervised training makes deep learning more effective. Shallow architectures have only two levels of computation and learning elements which make them inefficient to handle large amounts of training data [28]. Deep architectures require fewer computational units that allow non-local generalization which result in increased comprehensibility and efficiency that has been proved with its success in NLP and image processing [182]. According to complexity theory of circuits, deep architectures can be exponentially more efficient than traditional narrow architectures in terms of functional representation of a problem [4]. Traditional ANNs are considered to be the most suitable type of neural network for implementing deep architectures [182].

2.7.2. Convolutional Neural Networks-ConvNets

ConvNets can be considered as a special type of DNNs that perform extraction of features using a mechanism called convolution and a process of subsampling. The principal application of ConvNets is feature identification [182]. ConvNets are biologically inspired Multilayer Perceptron (MLPs) based on virtual cortex principle [25] and the earliest implementation is by Fukushima in 1980 [25] for pattern recognition followed by Lecun in 1998 [24]. ConvNets diversify by applying local connections, subsampling and sharing of weights which is similar to the principal approach of ANNs in the early 1960s [182]. In ConvNets each unit in the layer, in a manner similar to the earlier MLP model, receives input from a set of units in small groups from its neighbouring layer. The usage of local connections for feature extraction has been proven successful, particularly for extracting edges, end points and corners. These features extracted at the initial layer are combined subsequently at the later layers to achieve higher or better features [182]. The features that are detected at the initial stages may also be used at the subsequent stages. The training procedure of the ConvNets is shown in **Figure 2-2**. The first layer takes a raw pixel with 32 x 32 from the input image. The second layer consists of six kernels with 5 x 5 local windows. From this, subsampling is done in the 3rd layer (subsampling) layer. For the 4th layer, another ConvNets with 16 kernels was exploited with the same 5 x 5 windows. Then the 5th layer is also constructed using sub sampling. This procedure continues till the last layer and the entire structure is developed as Gaussian connections [182].

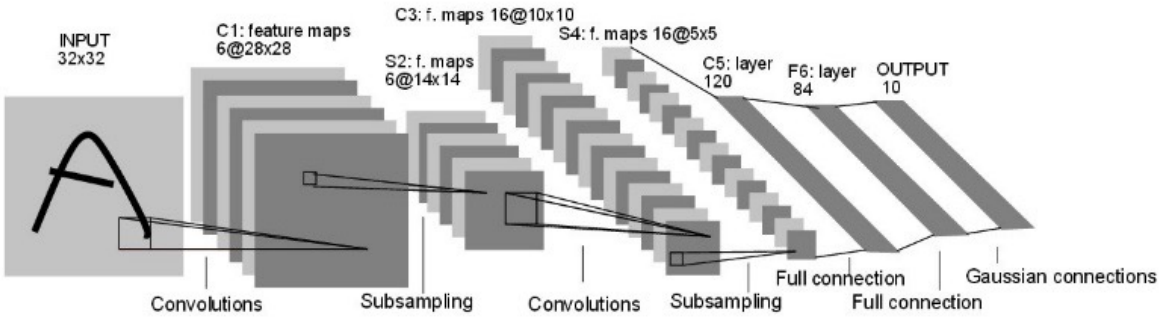


Figure 2-2: Architecture of the ConvNets CNN as proposed by Lecun [15]. Used with permission (open access license).

2.7.3. Deep Belief Networks - DBNs

The Deep Belief Network (DBN) is a form of deep architecture designed and developed by Hinton [26]. DBN is based on MLP model with greedy layer-wise training. DBN consists of multiple interconnected hidden layers with each layer acting as an input to the next layer and visible only to the next layer [182]. Each layer in a DBN has no lateral connection between its nodes present in that layer. The nodes of DBN are probabilistic logic nodes thus allowing the possibility of using an activation function. Restricted Boltzmann Machine (RBM) is stochastic ANN with input and hidden units connecting every hidden and a visible unit [182]. RBMs act as the building blocks of DBNs because of their capability of learning probabilistic distributions on their inputs. Initially the first layer of a DBN is trained as RBM that transforms input into output. The output thus received is used as data for the second layer which is treated as an RBM for the next level of training. Similarly, the output of the second layer will be the input for the third layer, and the process continues as shown in **Figure 2-3**. The transformation of data is done using activation function or sampling [182]. In this way the subsequent hidden layer becomes a visible layer for the current hidden layer to train it as an RBM. An RBM with two layers, a visible layer as layer 1 and a hidden layer as layer 2 is the simplest form of DBN [182]. The units of the visible layer are used to represent data and the units (hidden with no connection between them) will learn to represent features. If a hidden layer 3 is added to this, then layer 2 will be visible to only layer 3 (still hidden to layer 1) and now the RBM will transform the data from layer 2 to layer 3. This process is illustrated in **Figure 2-3** [182].

In DBNs, the lower-level features of the input are extracted at the lower layers and an abstract representation (high-level features) of the input is performed at the higher layers.

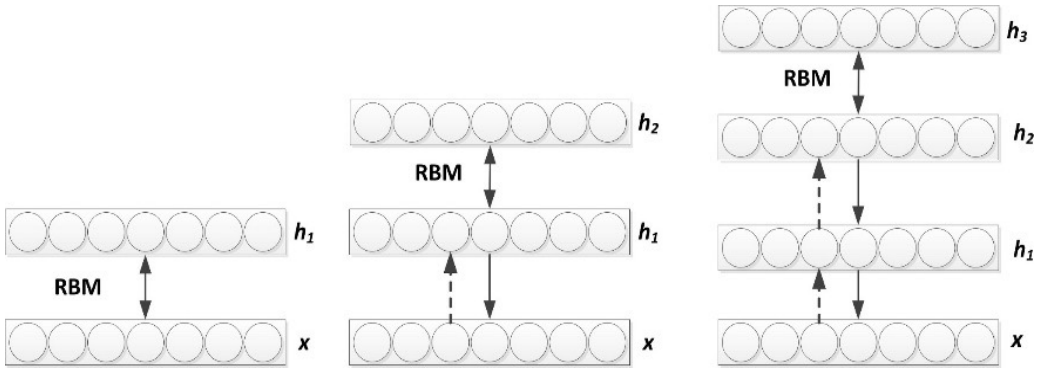


Figure 2-3: Pictorial representation of the first three layers of a Deep Belief Network where each layer is an Restricted Boltzmann Machine (RBM).

The training procedure of a DBN is carried out in three phrases. Each layer of the DBN is pre-trained with greedy layer-wise training followed by unsupervised learning for each layer and finally training the entire network with supervised training. The importance of this training procedure is determined by the generative weights. After learning, the values of the latent variables in every layer can be inferred by a single, bottom-up pass that starts with observed data vector in the bottom layer using generative weights in the reverse direction. DBNs proved to be the most efficient architectures in image recognition [23], face recognition [27] and character recognition [28].

2.7.4. Stacked Autoencoders – Deep autoencoder networks

The principle of autoencoders, using an encoding multilayer ANN, evolved from the attempts to reduce the dimensionality and find efficient methods to transform complex high dimensional data into lower dimensional code [182]. A decoder network is used to recover the data from the code [182]. Initially, both encoder and decoder networks are assigned with random weights and trained by observing the discrepancy between original data and output obtained from encoding and decoding. After this, the error is back propagated firstly through the decoder network, followed by encoder network. This entire system is known as an autoencoder [26].

An autoencoder with input $x \in R^d$ is "encoded" as $h \in R^{d1}$ using deterministic function defined as $f_\theta = \sigma(Wx + b)$, $\theta = W, b$. To "decode", a reverse mapping of f: $y = f_\theta(h) = \sigma W^1 h + b^1$ with $\theta = (W^1, b^1)$ and $W^1 = W^T$ is performed in which encoding and decoding process are performed using the same inputs. This process continues for every training pattern. For training, x_i is mapped to h_i with a reconstruction y_i. Parameter optimization is achieved by minimising the cost function over the training set. However, optimizing an autoencoder network with more

than one hidden layer is difficult. Being similar to DBN's layer-wise training procedure, this approach replaces RBMs by autoencoders that perform learning through reproducing every data vector from its own feature activation [4]. The considerable change that has been applied in this model is changing the unsupervised training procedure to the supervised mechanism in order to identify the importance of training model [182].

The training procedure of DAEs is as follows. In an autoencoder, three layers are considered at a time with the middle layer as the hidden layer. In the following instance, the middle layer becomes input layer and the output layer of the previous instance become hidden layer (the out parameters are now the training parameters) and the layer next to it will be the new output layer. This process continues for the entire network [182]. However, the results were not efficient since the network becomes too greedy [4]. It can be concluded that, the performance of stacked autoencoders with unsupervised training was almost similar to that of RBMs with similar type of training whereas stacked autoencoders with supervised pre-training is less efficient. Stacked autoencoders were not successful at ignoring random noise in their training data as such their performance is slightly poorer (almost equal performance but not the same) than RBM based deep architectures. However, this gap in performance is narrowed using the stacked de-noising autoencoder algorithm introduced in 2010 [28].

2.7.5. Unconventional Deep architecture

Hybrid and unconventional deep architectures are designed either combining/altering traditional architectures or by implementing new training mechanisms based on ML approaches [45]. In majority of the cases, the design of the deep architectures is problem specific and is based on the implementation. This section will provide a brief overview of some of the unconventional deep architectures and how these implementations guide the learning paradigms and problem solving.

Convolutional Deep Belief networks

Convolutional Deep Belief networks are based on Convolutional Restricted Boltzmann Machine (CRBM) which are a variant of traditional RBMs. Unlike RBMs, the weights of CRBMs are shared between the layers. CDBN implements probabilistic max-pooling similar to CNNs which allows to shrink the representation thus reducing the computational cost

without losing efficiency [184]. CDBN produced better results than traditional DBN for CIFAR dataset [185].

Deep Kernel Machine (DKM)

Deep Kernel Machine (DKM) is constructed by stacking kernels as layers for a deep architecture similar to stacked autoencoder approach. In DKM, each layer (kernel) is associated with the next layer similar to DBN based architecture and each layer is optimised using kernel based optimisation methods [48]. Each layer of DKM goes through a feature selection using supervised learning to eliminate unwanted features followed by a classifier layer at the end (usually kNN). DKM, for the first time, implemented the use of a kernel based deep architectures that attained considerable success particularly for multiclass problems and outperformed other kernel based deep architecture approaches [186]. DKM provided a new direction for implementing kernel based methods for optimising deep architectures through efficient feature learning mechanism [187].

Deep Coding Network (DCN)

Deep Coding Network (DCN) is the extension of traditional sparse coding one-layered ANN using DBN based architecture with hierarchical structure [188]. The hierarchical multilayer structure is proposed to provide local search for avoiding overfitting. The hierarchical nature and the local search reduce the computational cost compared to a generic sparse coding multilayer network. However, the performance becomes reduced as the number of layers is increased making it inefficient for problems with large datasets.

Tensor-Deep Stacking Network (T-DSN)

A Deep Stacking Network (DSN) uses parallel weight learning process where the weights are updated by grouping them into separate blocks, unlike BP (all weights at once). A DSN consists of one hidden layer and stacking a minimum of 3 such DSNs forms Tensor-DSN [189]. The input layer of the DSN is linked to the hidden layer by lower weight matrix and the logistic hidden layer that uses sigmoid is connected to the output layer with an upper layer weight matrix [45]. At present, T-DSNs are implemented principally for big data analytics [190] and data pattern analysis [191].

Deep Q-Networks (DQN)

Q-learning is a variant of Reinforcement Learning (RL) that adopts the process of selecting most suitable action for a finite process instead of reward based RL approach. Deep Q-Networks are designed by applying Q-learning principle for fine-tuning weights in a traditional DNN [192]. Traditional deep architecture based approaches sequentially update the weight, which is costly, whereas DQN uses random weight updates which reduces the frequency of updates thus reducing computation cost. DQN is proven efficient over the traditional RL-based approach for gaming and has provided a new research direction for RL-based deep learning [193-196].

Deep Support Vector Machines (DSVMs)

Deep Support Vector Machines (DSVMs) are designed by stacking SVMs as individual layers of traditional DNN [197]. DSVMs are also referred as SVM based deep stacking networks [198]. The difficulty in design and implementation of DSVMs is the major reason for its limited implementation as DSVMs cannot be created by simply stacking SVMs which is a common approach for DNN based implementations [187]. Another possible variant is using SVM as classifier (Softmax layer) in different types of deep architectures [199]. A recent implementation of Deep CNN architecture with SVM as Softmax layer is used for automatic mass detection for breast cancer which has toppled the benchmark results for the same problem using traditional CNN application [200].

Evolutionary Deep Neural Networks

Application of evolutionary strategies for improving the efficiency and accuracy of ANNs has inspired implementation of evolutionary strategies for optimising DNNs. Initially, the genetic algorithm (GA) assisted approaches are used for optimising DNNs [201] and these approaches are still quite popular [202, 203].

In spite of the success of Neuroevolution (evolving ANN), evolving deep architecture is considered as complex and time consuming. However, applying various evolutionary strategies for evolving deep architectures attained considerable success in recent times [51, 204, 205]. Among all the evolutionary DNN approaches, recently proposed neuroevolutionary based method is popular and widely accepted [51] due to its easy adaptation and application. A new multipopulation based coevolution strategy for evolving DNNs is also proposed [52, 59]. Use

of multiple sets of population and separate coevolutionary strategies for each set of population is useful for generating diversified solutions in short periods of time.

Apart from the unconventional approaches stated above, there are other methods implemented which are essentially an amalgamation of two or more ML strategies applied for different types of deep architectures.

2.8. Knowledge Representations in Artificial Neural Networks

The knowledge attained by ANNs resides in their weights. An individual weight by itself may not be significant, but in combination with other weights it is responsible for the problem solving.

Given that the first research objective is to identify a relationship between input features and ANN weights, the second research objective is to identify whether deep architectures, particularly DNNs, possess knowledge representations in the form of patterns. Answering this question will also provide a new opportunity to extract symbolic knowledge which would help to explain, to some extent, the internal operations of deep learning that provide a new direction in transfer of knowledge.

"knowledge in a neural network is the expertise attained by training."

For further investigation, it is important to define what is meant by knowledge and what it is made of. To investigate on how ANNs works and to explore the factualness, it is necessary to expose hidden representations in the weights of ANNs that constitute knowledge. These representations in the weights are significant, as weights by themselves are nothing but simple numeric values. To extract a meaningful pattern from the weights, it is necessary to establish a relationship between weights and the core components that are responsible for the change of weights, their values and their patterns.

Weights are optimised for a given task and are often optimised based on accuracy of the results. However, the constitution of weights is achieved through input features. To know about the patterns of the representations in weights that constitute knowledge, it is essential to know the relationship between input features and hidden representations. With one-layered or two-layered ANNs, it is difficult to identify representations from the weights since all the features

are condensed within these layers. This will increase the complexity of extracting knowledge. For instance, **Figure 2-4** presents a pictorial representation of an ANN's weights projected on to 3-dimensional space. The ANN consists of one hidden layer and is trained on the MNIST dataset. The ANN is able to achieve a classification accuracy of 69.2%. When the weights of the only hidden layer are projected, it may be noticed that no visible patterns are present. However, a clear difference between various types of weights can be observed with colour coding. It can be assumed that the colour actually represents a particular feature and these features are spread across multiple weights and scattered across the layer.

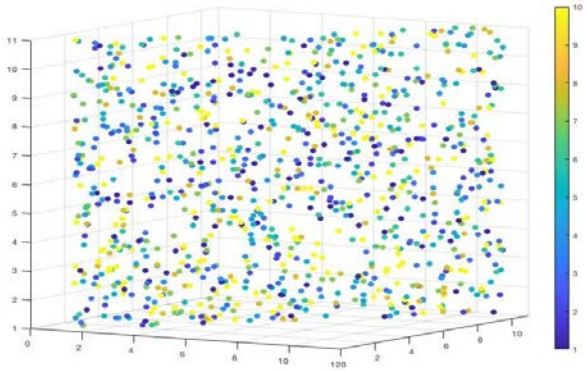


Figure 2-4: The 3D Projection of weights of the hidden layer of a fully trained ANN using MNIST dataset. The colour bar indicates the attribute to which the weight belongs to. The x, y, and z axes are based on the values automatically determines by MATLAB.

When a DNN with seven layers is used to classify the same MNIST data and trained to achieve the same accuracy of 67.2%, the weights in the middle layer reveal an entirely different pattern when projected (**Figure 2-5**). The MNIST data set consists of 10 digits which are distributed across four different features A, B, C and D. In the **Figure 2-5**, the features A, B, C and D are projected in four different colours and the features are grouped in patterns that are recognisable. The patterns possess knowledge perhaps in the form of partial class labels that assist classification after few more transformations through other layers towards the last layer.

Further, this type of projection particularly with DNNs gives an opportunity to investigate the influence of input on the weights which in turn can be constituted as features. The mechanism of identifying the relationship between input features and weights is the first one of its kind and is being attempted for the first time in this research. Further evaluation and explanation will be presented at the later stages of this thesis.

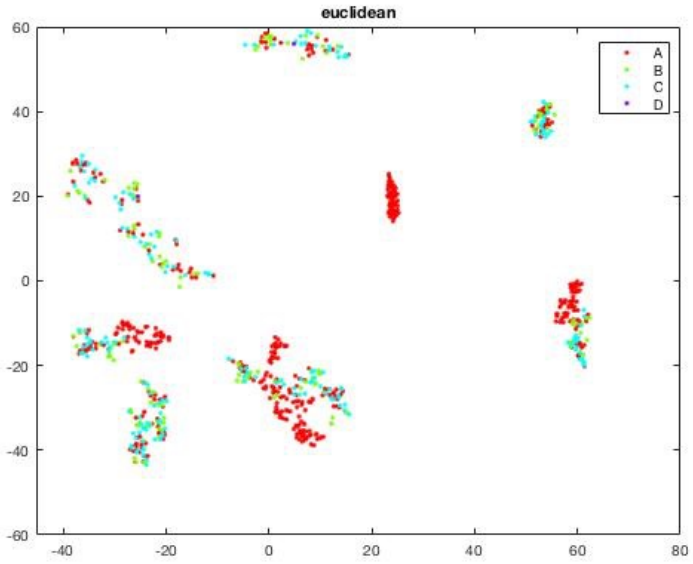


Figure 2-5: Euclidean valued projection of weights of the middle layer of a 7-layered DNN trained using MNIST dataset. The features indicated by A, B, C and D are clearly separated with A and B being strong features.

Knowledge extraction is another major point of research for neural networks. The task of knowledge extraction from ANNs can be directly associated with exploring how the problem is represented in the hidden layers (topology) and neurons (weights). The research on extracting knowledge from ANNs was widely popular in the late 19th century [67-70]. A notable account of extracting knowledge from ANN weights is done by removing insignificant attributes from the input and training the model with known data (supervised) [206]. This attempt may be successful for linearly separable data, but for non-linear multi-class data, the importance of an attribute may not be evaluated correctly with respect to other attributes [65, 71]. Further, knowledge discovery attempts have attained limited success in the problem specific experiments. For instance, the successful attempt to extract symbolic rules from ANN by Kamruzzaman and Monirul is confined to a single digit hidden nodes with a small dataset [58].

The common association of knowledge extraction and knowledge transfer for ANN is its weights. Transfer of learning and transfer of knowledge are often used in the same context. Simply, knowledge in a neural network is the learning and experience attained by training. Transfer of learning will enable a transfer learning mechanism in the form of parametric values as well as knowledge attained, whereas transfer of knowledge is the experience that the neural

network gained through the learning. There is no guarantee that efficient learning will result in efficient knowledge.

Though layer-wise training contributed to the success of DNNs, there has only been a few attempts to investigate the possibility of knowledge transfer between two DNNs. All these methods implement transfer of weights between DNNs without any established systematic approach. One such attempt using deep convolutional neural networks achieved limited success with small datasets [58]. The second notable implementation involved classifying upper case Latin characters using an ANN that was trained on Chinese characters [64].

There have been numerous attempts for improving the performance of DNNs. Transferring knowledge is one such approach where features learnt by one DNN are transferred to another DNN to improve performance and accuracy. The importance of knowledge transfer between ANNs was identified as early as the 1990s [207]. The process of knowledge transfer involves identification, extraction and transfer of knowledge which is also referred as 'Transfer Learning'. Deep Transfer Learning (DTL) was hypothesized by Bengio in 2013 [16]. DTL attempts to identify transferable features in DNN and copy them to another DNN to improve performance and accuracy. The earlier attempt towards feature transfer between DNNs is attempted by Yosinski [54]. This approach investigates on identifying layer(s) where generalization is occurring. In this approach, two Constitutional Neural Network ($CoNN_1$, $CoNN_2$) are trained on two equally divided parts of ImageNet dataset. Then, the weight vectors of $CoNN_1$ and $CoNN_2$ are copied to a new network $CoNN_3$ and $CoNN_4$ three layers at a time while randomly selecting the weights of the other layers. After several experiments, Yosinski concluded that generalization was occurring in the first two layers of the CoNN. The transferable features exist only in the first two layers of a DNN for same dataset. However, these results were only repeatable when experiments were conducted on similar datasets (dataset with similar structure of ImageNet) which raise questions about the existence of transferable features only in the first three layers.

The second notable implementations is the classification of the upper case Latin characters using an ANN that is trained on Chinese characters [64]. In an another approach presented in 2014, ImageNet dataset is used for classification of images which concludes that first three layers consists of more generic features that can be transferred to another DNN for image

classification problems [54]. The most recent work by Terekhov uses an alternate approach in which a block of weights are introduced between a trained DNN to obtain a set of weights that are optimized with the values between the layers [206]. The new DNN is trained after introducing this set of blocks between the layers which reduces the training time. The approaches mentioned above depend on transferring a set of layers (weights) from one DNN to another. Further, there was no clear indication on what knowledge is being transferred. The first two approaches transfer a set of layers that are identified by comparing classification accuracy by freezing the weights of two to three layers at a time on a trial and error basis.

A transductive transference approach was proposed to implement transfer learning by reusing extracted features [208]. A transductive learning approach examines and learns from the training on a specific task on the same dataset. So, in the case of source and target with different distributions, a transductive transference approach improves the classification results by transferring exploited labelled training instances from trained network to an untrained network. Experiments for detecting Latin digits using the weights obtained from a DNN trained on Arabic digits dataset has improved performance and accuracy of the classifier in detecting Latin digits. However, the place or layer where the generalisation is occurring is still unknown.

The Deep Adaptation Network (DAN) architecture is the first attempt for exploring the process of DNN learning and the generalization of deep CNNs [57]. DAN generalises CNNs towards domain adaptation state where task specific features are identified and transferred. DAN also confirms that generic features are transferable, whereas task specific features need to be tailored before transferring to solve a different task.

The efficiency of CNNs for image classification has motivated many researchers to work on transfer learning using CNNs which attained a considerable success rate [63, 209, 210]. However, these approaches using CNN are limited to image classification and their methods of transfer learning is mainly parameter based and confined to small datasets. For instance, the work by Wang et al. on crop yield prediction using remote sensing images of soya crops in Argentina was quite successful [211]. They used the same parameters to train a different network with a small amount of data consisting of remote sensing images of a Brazilian crop. Although this attempt was able to produce good results, it is simply copying of parameters and hand-crafting input features. For these types of implementations, there is a high chance of failure when small but variant changes are made to the input data.

For DBNs, an interesting transfer learning approach called Growing DBN with transfer learning (GDBN-TL) was proposed [212]. In this, the DBN initially has only one layer and is trained to learn the features. Then the weight parameters are frozen, and an additional layer is added which will be with the parameters copied from the frozen layer. This process continues for a considerable amount of time. This is further followed by another round of top to bottom layer-wise training. In the work on GDBN-TL, the authors claim to have reduced training time considerably by transferring (or perhaps copying?) knowledge and parameters in a process they call instantaneous transfer [212]. However, this is similar to the previously mentioned CNN works. Another interesting work uses DBN transfer learning for speech classification pertaining to emotion or sentiment recognition [213]. The DBN is trained on one/two language datasets and tested on the rest of the datasets. The authors claim to achieve good accuracies for five different datasets comprised of three languages, namely German, Italian, and English with the DBN trained only in German and English. The comparison of accuracy is with autoencoder networks and the DBN has only three layers out of which two layers are transferred (copied) which makes the results questionable.

Transfer learning scenario of DAE is similar to the transfer learning principle of CNN / DBN and has been successful in improving efficiency and accuracy of the classifier [214, 215]. A knowledge transfer approach using DAE for hierarchical data was proposed which attempts to transfer the knowledge attained by a DAE to another DAE that has already been partially trained. The first DAE was trained on a good dataset (uncorrupted) to achieve maximum accuracy. The second DAE was trained on a distorted dataset with which it could achieve a maximum of 56.7% accuracy out of possible 80% accuracy which is achieved with uncorrupted dataset. When the weights of first DAE are transferred to the second DAE and tested on the same corrupted dataset, there was an improvement of about 22% in accuracy (from 56.7% to 79.6%). This approach not only proves the efficiency of transfer of knowledge but provides important information about knowledge attained by DAE [56]. When a DAE is trained with a good dataset, it learns the features in such a way that it could re-construct the features accurately in spite of missing values in the damaged test dataset.

2.9. Transfer Learning and Knowledge Transfer

Transfer learning and knowledge transfer are equivocally presented in the literature creating some uncertainty on a profound definition. Knowledge transfer is a part of transfer learning

but, it is not ‘the only’ part. Transfer learning is definitively presented in the literature as transferring knowledge attained by one system to another untrained or dumb system resulting in reducing the training time or improving the performance or sometimes both [55]. However, there is a conceptual mismatch with these definitions as industrial requirements often demand a variety of practical practices. Considering some cases, transfer of parameter value may be enough to produce sufficient classification or required efficiency. In some cases, the execution time is important and the decision making can be achieved with minimum results. For instance, identifying a single negative result among multiple results may not require the entire execution mechanism.

Considering conceptual confusion and contradicting definitions, this research proposes a categorisation of transfer learning based on what is been transferred. The types of transfer learning proposed in this thesis are as follows.

1) Transferring only parameters.
2) Transferring only neural network weights (layers) and choose other parameters arbitrarily.
3) Transferring both parameters and weights.
4) Transferring some parameters and/or weights based on the problem.

The key issue with the mainstream literature is lack of clarity and consensus on what exactly knowledge is. The question of what is to be transferred and what is sufficient to produce desired results also needs to be answered. There are some works in literature which are purely based on transferring only weights (hidden nodes/layers) to improve efficiency, whereas other approaches are based on copying parameter values. To attain some clarity, it is necessary to pursue an investigation as to how knowledge is represented and what needs to be transferred as knowledge.

2.10. Research Gap

From the literature, it is evident that learning representations is significant and contributes to the success of deep learning. However, majority of the research concentrates on optimising the learning algorithm to achieve higher accuracies. In spite of a clear knowledge of learning algorithms and advantages of learning features, there is, to my knowledge, no notable work on exploring the underlying representations in the weights. The majority of the ANN and now

DNN research is confined to optimising learning by introducing new training algorithms and parameters which are problem specific.

The ambiguity of transfer learning and knowledge transfer has been a point of discussion that appears throughout the related literature and is one aspect that needs to be resolved and studied.

From a comprehensive review of the known literature undertaken as part of this research, the unexplored research areas highlighted in Section 2.8.1 are identified.

2.10.1. Research Gap in line with the Research Problem

- The relationship between input features and neural network topology has not been explored to the extent of studying the impact of addition or deletion of layers.
- There is an absence of empirical study on extracting the underlying representations in neural network weights that can be mapped to input features; i.e., how changes in the input features are reflected in neural network weights.
- There is a lack of a systematic approach to explore how features are represented in weights which are simple numeric values.
- Research is lacking on the influence of topology, their limitations and boundaries while designing a neural network.
- There is no reflective research on the importance of a layer in a deep neural network and its influence on overall accuracy.
- Transfer learning is merely transferring weights with no true information of what is been transferred.
- The absence of a transferable neural network model (statistical or mathematical) which established the knowledge in the form of a feature or group of features.

2.11. Chapter Summary

This chapter's contribution can be summarised as follows:

- Feature processing involves feature construction, selection and extraction out of which feature extraction is attributed to knowledge extraction and there are limited works in the literature on feature extraction using artificial neural networks.
- Deep learning is the process of learning underlying representations that exists in input features through neural networks weights.

- The underlying representation in the neural network weights constitute knowledge that is acquired by deep architectures through deep learning.
- The transfer learning approaches existing in the explored literature are just copying of layers without knowing what exactly is getting transferred
- There is no known work on a systematic approach or mathematical model for transfer of knowledge

The next chapter presents a preliminary investigation into these questions and attempts to identify the existence of knowledge in a DNN followed by some initial experiments on feature extraction and transfer learning.

Chapter 3 Preliminary Investigation

3.1. Introduction

The current neural network research concentrates predominantly on how to optimise topology and other parameters for problem solving. The literature review presented in the previous chapter provides an insight into the neural network research pertaining to the optimisation as well as feature learning. There is very limited research on the impact of modifying topology, particularly adding or removing layers. Problem specific neural network models are also designed based on trial and error, thus, provide a very little knowledge on why a topology with a particular number of layers is able to produce better results.

It is evident that each and every layer of ANN model is important, and has some influence on the overall efficiency and accuracy of the ANN. The impact may be positive or negative which can be related to the improving or reducing accuracy when a layer is added or removed. By removing or adding layers it should be possible to determine which layer is significant and has the highest impact on the outcome of the ANN. The conjecture that is presented in this thesis that a particular layer is more significant than other layers, will provide an insight into the existence of significant knowledge in a particular layer or layer(s).

This chapter proposes the initial hypothesis towards attaining the research goal and investigates its feasibility through a systematic experimental evaluation. The research feasibility is studied by exploring the following aspects:

- Impact of adding and removing layers at various positions
- Importance and influence of middle layer
- Impact of a position of a layer on accuracy and efficiency

There are five sets of experiments carried out in this section. Firstly, the relationship between input features and neural network weights are investigated followed by a second set of experiments to know the importance of a layer by exploring layers for transferable knowledge. A set of transfer of layers experiments is carried using different scenarios to investigate the impact of transferring weights (layers) from a trained DNN to an untrained DNN.
Feature extracting and transfer learning experiments are carried out to demonstrate the transfer of knowledge and its impact on efficiency of neural networks. Finally, DNN optimisation experiments are carried out using knowledge components and a newly proposed concept called Weights of Weights (WofW).

3.2. Initial Hypothesis

"The middle layer(s) of a deep neural network is significant and has highest impact on the accuracy of the neural network. "

$H_{1\text{-}1}$: The weights extracted from middle layers are significant.
$H_{1\text{-}2}$: The weights extracted from the layers near input are overlapping and contribute less to classification

3.2.1. Evaluations:

Hypothesis is evaluated using Synthetic Hierarchical Dataset, CIFAR, TIMIT and MNIST datasets. The experiments are carried out in two different scenarios.

Scenario 1:

Firstly, a set of layers are removed, and classification accuracy is compared to the best accuracy to test the importance of layers. This is followed by a transfer learning scenario to test the classification accuracy when the weights from a particular set of layers are transferred.

Scenario 2:

For each layer, the relationship between features and weights is evaluated in each layer by adding and removing features one at a time.

The hypothesis states that when trained, a DNN tries to learn the representations in the input from the first layer and while reaching the middle layer, the weights are optimised in such a way that the middle layer possesses the knowledge of all the features in the form of neural network weights. From the middle layer, the features start getting unfolded towards being problem-specific or class-specific high-level features by the time they reach the last layer of the DNN. In other words, the middle layer acquires the information through training and holds significant knowledge in the form of underlying representations which will pass through the rest of the network constructing problem specific (class based) high-level features.

The null hypothesis in this case is to provide sufficient evidence that the accuracy of neural network is least impacted when the middle layer is removed when compared to other layers of the neural network.

3.2.2. Experiments & Evaluation

The initiative to test the proposed Blossom Effect is to investigate the importance of an individual layer based on its position. Another consequence of this hypothesis is that the middle layer(s) should contain significant knowledge that is attributed to the success of the neural networks. The objective of this chapter is to study the influence of a layer on the accuracy of the neural network. This, in theory, will help in knowing the importance of a layer which in turn provide an explanation of the impact of inclusion and exclusion of a layer(s) on classification accuracy.

A systematic experimental process is adopted to provide an evidence on the importance of middle layer. To start with, a set of experiments is carried out to examine the capability of neural networks to learn hierarchical features. This is followed by testing the impact of having equal number of nodes in all the hidden layers on the classification accuracy for hierarchical dataset.

This is followed by identifying the importance of a middle layer and its impact on accuracy through experiments based on multiple strategies. Experiments are carried out by removing layers from different positions of a trained neural network and testing the accuracy without re-training. The chapter also tries to establish the importance of middle layer through transfer

learning experiments i.e., transferring the weights of middle layer to different layers of an untrained neural network.

3.2.3. Expected Outcomes

The experiment results will provide:

1) The impact of removing a layer from different positions
2) The importance of middle layer(s)
3) The impact on accuracy when a middle layer is transferred to various positions

3.3. Relationship between Input Representations and DNN Topology

The main purpose of this experiment is to discover the relationship between topological hierarchies of layers in a DNN (DNN hierarchies) and the hierarchies in the input features called Features Hierarchies (FHs), if any. It is not clear whether modelling FHs with a hierarchically organised DNN conveys has any benefit over using non-hierarchical neural networks. It is also not clear whether the representation of the input reflects exactly in the topology.

Taxonomy based datasets are available with biological hierarchies as benchmark for various bioinformatics algorithms. However, it is not practical to use them in this preliminary investigation due to their size and complexity.

To overcome this, a biological synthetic dataset is constructed using representation with known FHs. Connectionist methods of data representation can be categorized into two types: Localist and Distributed. In localist representation, each unit is associated with a single feature or concept and each concept is represented by one and only one unit [216]. Localist representation is simple to use and easy to code but not feasible for a component, structure-based data such as FHs. In distributed representations, a single concept is represented by a combination of multiple units and each unit can be a part of multiple concepts [216]. For instance, if an organism has a subset of 3 features, there is possibility that one of these features may also be present in another organism. An organism possesses a lineage based on features which form the hierarchy of that organism and distributed representations are suitable for representing such taxonomic (hierarchical) data. It is noteworthy to observe that in a distributed representation,

an isolated or independent unit has no meaning by itself, and it is valued only when it is in a group. With binary encoding and distributed representation, *n* neurons can produce 2^n patterns with which small number of units can represented huge amounts of data.

A localist representation represents a single feature or concept on its own, such as an organism having backbone, hair, and other unique features which determines the uniqueness of the organism. If the organism has multiple features that are represented as 8 localist bits, each bit will indicate either the presence or the absence of a concept or feature. For instance, an organism that has a backbone is coded as C1 with last bit as 1 and is represented as 0 0 0 0 0 0 0 1. Similarly, other features may be represented as shown in **Figure 1-1 (a).** The remaining bits of a localist representation can be used to identify individual organisms.

Features	Representation
C1 (Backbone)	0 0 0 0 0 0 0 1
C2 (Hair)	0 0 0 0 0 0 1 0
C3 (Hands and Feet)	0 0 0 0 0 1 0 0
C4 (hair on hands)	0 0 0 0 1 0 0 0

(a) Localist representation

Organism	Features	Representation
O1	C1 and C2	0 0 0 0 0 0 1 1
O2	C1 , C3, C4	0 0 0 0 1 1 0 1

(b) Organism: Multiple features

Figure 3-1: Representations: (a) Localist representation of features where each feature is represented by 8 bits. (b) Representation of organisms: Every organism is comprised of multiple features.

FHs classification of organisms into taxa is based on the features they possess. Since each feature of an organism is represented in bits, organisms with multiple features are represented in localist form as a combination of binary bits. For instance, the organism O1 has backbone and hair which are C1 - 00000001 and C2 - 00000010 as presented in **Figure 3-1 (a).** Therefore, the features of the organism O1 are represented as 00000011 with combined features as shown in **Figure 3-1 (b)**. Similarly, the organism O2 has backbone, hands and feet, and hair on the hands (C1, C3 and C4) which is represented as 00001101. This pattern of features of the organism determines its sub-group.

Hierarchical data can be defined as data units with hierarchy-based interrelation among them. A taxonomic dataset is taxa-based data with FHs to represent organisms organized by species for easy and efficient management of data as well as retrieval. The hierarchical tree is

constructed from a synthetic dataset of organisms. An organism is represented as a stream of binary data of 20 bits categorized into Rank (4 bits), Group (4 bits), Subgroup (4 bits) and features (8 bits) as shown in **Figure 3-2**. The taxonomic Rank is determined by the shared features, Group, and Subgroup making this a hierarchical representation. The cophenetic correlation coefficient is useful to realise the efficiency of hierarchical structure based on the similarity between two values obtained by calculating the distance between a pair of unmodeled data within a dendrogram [217]. The typical value for the cophenetic correlation coefficient is around 0.8 with values above 0.95 considered as more efficient [218]. The cophenetic correlation coefficient for this data is 0.9934. This value highlights that the dataset created using synthetic data is efficiently structured with considerable accuracy.

Organism	Rank	Group	Sub-Group	Features
1	0001	1101	1101	11011101
2	0010	1010	1010	10101010
3	0011	0101	0101	01010101
4	0100	0100	0100	01000100
5	0101	1001	1001	10011001
6	0110	1011	1101	10111001

Figure 3-2: Binary representation of organism with 20 bits in distributed format with 4 bits each for rank, group and sub-group and 8 bits for features

The main purpose of this set of experiments is to examine the learning capability of DNNs for hierarchical data with known feature hierarchies, as the first step towards identifying the plausibility of this research. The aspect of presence of knowledge and its identity in the DNN weights is based on input, and its representation will help in exploring the relationship between input features and their representation in DNNs weights. A set of experiments are designed aimed at extracting information and developing a new direction in the research to realise the importance of the input, weights and DNN topology.

There are 90 organisms in the synthetic dataset categorised into six different species. For all experiments, the dataset is divided randomly with the first 60% for training, the next 10% for calibration/validation and the remaining 30% for testing. After some initial trials to identify appropriate parameters, the initial learning rate was determined as 0.01 with a step-ratio

(incremental learning step size) of 0.001 and a momentum of 0.3. To reduce the complexity and irregularity which may be caused by large weights and weight-decay, a simple penalty function is introduced to penalize large weights. Weight-decay is calculated as the half of the sum of squared weights times a coefficient termed as weight-cost which is 0.0002 for this experiment (a typical starting value for weight-cost is 0.0001). The objective function for the experiments is 'Cross Entropy.' Each experiment was performed 10 times with 100 epochs, different weight initialisations and the results are averaged.

This experimental study is divided into two categories. In Experiment I, a DNN is trained to classify the species depending on the features. In the second experiment (Experiment II), a second set of data is used to identify whether any two given organisms are related (belong to same species) or not. Four different types of strategies are adopted for each experiment.

Table 3-1: Results of Experiment I: Percentage of accuracies of training, validation and testing using four strategies along with the rmse values for training.

No.	Hidden layers	Train	Validation	Test	Total	Avg. Train rmse
1A	3 (30,30,30)	100%	100%	100%	100%	0.023
1B	3 (30,40,50)	100%	100%	81.5%	94.4%	0.021
1C	4 (30,30,30,30)	100%	100%	92.6%	97.8%	0.0499
1D	4 (30,40,50,60)	13%	55.6%	14.8%	17.8%	0.0448

To start with, the number of hidden nodes is chosen as 30 which is 1.5 times the number of inputs (nodes). There can be three types of scenarios for selecting hidden nodes, same number as number of inputs, double the number of inputs or any other number. The main reason for selecting 30 is that it comes in between the two options of same and doubled number of inputs. If the number of hidden nodes is doubled at the first hidden layer, there might be a possibility of dilution of features at the first stage which could affect the intended experiments.

For scenario A and B, a DNN with hidden nodes 30, 30, 30 and 30, 40, 50 is used whereas for C and D, a DNN with hidden nodes 30,30,30,30 and 30,40,50,60 is used. This scenario is designed to help determine the influence of symmetric (same number of nodes in all the layers) and asymmetric node count. For Experiment I, 20 inputs representing the 20 bits of the

organism are used with six separate outputs for determining the species of the organism and the results obtained are presented as **Table 3-1.** Experiment 1A, in which the hidden nodes are 30, 30, 30, shows 100% results for training, validation and testing. When the number of nodes in the hidden layers was changed to 30, 40, 50, there was a variation in the classification accuracy with testing dataset which is 81.5% constituting the overall results as 94.4% as shown in **Table 3-1.** However, when the depth of the DNN was increased to four, the confusion matrix showed little variation compared to DNN topology with a depth of three, whereas the results of the experiment with different number of hidden nodes (1D) showed a considerable reduction in the accuracy rate with 17.8% as an overall percentage. Inspection of the confusion matrix reveals that classification error has occurred for species 5 with three of class 5 being classified as class 4 due to similarity in majority of their features.

The performance difference (Total) between experiment 1A and 1C is 2.2% in the favour of 1A. However, the difference between 1B and 1D is 76.6% in favour of 1C. On the other hand, if analysed, the impact of the same number of nodes and different number of nodes with same number of layers, the difference between 1A and 1B is 5.6% in favour of 1A and 1C and 1D is 80% in favour of 1C.

The second set of experiments (Experiment II) were carried out to identify whether two organisms are related or not. For example, the tiger is related to cat as they belong to the same species whereas a rat which belongs to different species is not related to cat as defined in our synthetic data. The input, in this case, is a 40-bit binary number vector fed to the network (20 each for two organisms) resulting in either '0' for not related or '1' if related. 60 data samples, (10 from each species) are used for this experiment, and the results are shown in **Table 3-2.**

Table 3-2: Results of Experiments - II: Training, validation and testing accuracies along avg. rmse for the experiments to identify whether the two input organisms are related or not.

No.	Hidden layers	Train	Validation	Test	Total	Avg. Train rmse
2A	3 (30,30,30)	100%	100%	88.9%	96.7%	0.0491
2B	3 (30,40,50)	100%	100%	100%	100%	0.0027
2C	4 (30,30,30,30)	100%	100%	83.3%	95.0%	0.0497
2D	4 (30,40,50,60)	100%	100%	93.4%	98.3%	0.0428

In summary, better results for Experiment I are achieved when each layer in the topology has the same number of hidden nodes whereas in Experiment II better results are achieved for the topology in which each hidden layer has a different and incrementally increasing number of hidden nodes.

The difference between overall accuracy for the experiments, in Experiment II, with three hidden layers namely experiments 2A vs. 2B, is 3.3% in favour of 2B. In the case of experiments with four hidden layers, experiment 2D is 3.3% more accurate than 2C. When the performance difference is analysed in terms of depth, the topology with three hidden layers (2A and 2B) had better performance than the 4-layered topologies (2C and 2D) with an average difference of 5.6% and 6.4% respectively.

A taxonomic FH with associated data was generated, and a DNN was trained to classify the organisms into various species depending on their characteristic features. The ability of DNNs to identify whether or not two given organisms are related (depending on the sharing of appropriate features in their FHs) was then tested. The experimental results showed that the accuracy of the classification is reduced with an increase in 'depth' of the topology. Additionally, improved performance was achieved when every hidden layer had the same number of nodes (symmetric) compared to the strategy where hidden nodes are increased as they progress towards the middle layer. These experiments show that the relationship between DNNs and FHs is not simple and may require further extensive experimental research to identify the best DNN architectures when learning FHs. With the experiment results published in [29], it is concluded that, all the representations are present in all the weights and vice versa.

The conclusion is based on the evaluation of these specific tests on hierarchical dataset which are performed for the first time to test whether the DNN is able to classify and identify the ancestry based on hierarchical features presented in distributed representations [29].

3.4. Identifying the Importance of Layers

Identifying the importance of a layer to find its impact on an overall accuracy of the neural networks is one of the crucial aspects of neural network research which, to my knowledge, has yet to be investigated. Examining the impact of various layers will provide an insight into the

research of the impact of changing the number of layers in DNN. Further, this will help to identify important knowledge hubs, those layers that influence efficiency and accuracy the most, among various layers of topology. Considering the importance of this section, the experiments are carried out with three different and versatile datasets namely MNIST, IRIS and the synthetic dataset that was created and employed in the experiments reported in the previous section. The IRIS dataset [219] allows for the classification of flowers of three different species and contains 150 samples. The MNIST dataset [220] is a character recognition dataset with 60,000 training and 10,000 testing images (see Chapter 6 for a full description of these widely used datasets).

The experiments were carried out using MATLAB, Weka and the Microsoft.NET framework to investigate if there are any software bound limitations particularly considering the random initialisation of weights. For all experiments, the data sets were divided for testing and training to ensure that the testing data was not exposed at the time of DNN training.

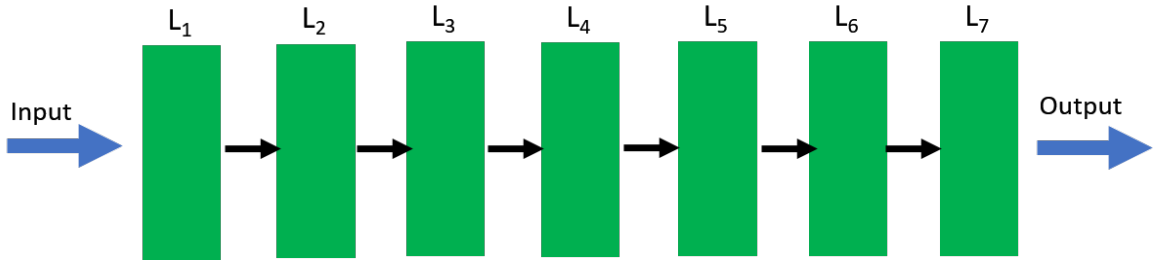


Figure 3-3: Architecture of initial 7-layered neural network. Each bar represents a layer in the network. Encoding a bar as green means that the weights in that layer are frozen (fixed). Thus, in the architecture in this figure all layers in the network have frozen weights.

To examine the impact of individual layers and their contribution (individually) towards overall accuracy and performance, the following three experiments are carried out. Firstly, classification experiments are carried out using above mentioned datasets with a 7-layered DNN with the weights of all the trained layers frozen to changes. Each experiment is carried out in ten tests with 25 runs per batch. From these experiments, three results are selected from highest to the least accuracies as follows: (T_{Best} (DNN_B)), 2nd best T_{SBest} (DNN_{SB}) and T_{Worst} (DNN_W). **Figure 3-3** presents the structure of DNN that achieved T_{SBest} results. These results are used to carry out two more experiments as detailed below.

3.4.1. Freezing Weights of DNN one Layer at a time:

To investigate the influence of each layer, classification experiments are then carried out with the original results T_{Best} (DNN_B), freezing the weights one-layer at a time. In the first run, weights of the layer-1 are frozen so that they cannot change and other weights are allowed to initialize with random values as per the regular practice as shown in **Figure 3-4** (top). For the second run of experiments, the second layer weights are frozen (**Figure 3-4** (middle)), and weights of the other layers are initialized with random values. This process is carried out for all the layers one-layer at a time which as illustrated in **Figure 3-4**. For now, there is no conclusive evidence on weights being ordered differently for different layers but the impact of freezing a layer could be assessed by the experiment results.

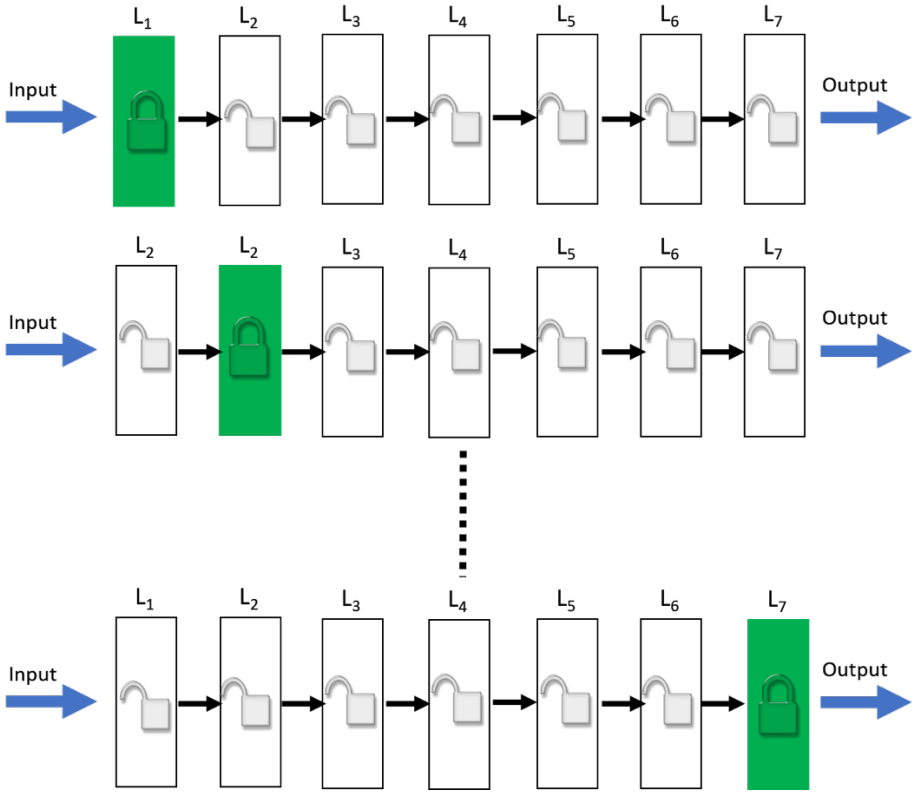


Figure 3-4: Illustration of the importance of layers experimental setup: Weight initialization with freezing of the weights of various layers, one layer at a time. The weights in the chosen layer are adopted from a trained network and all other (unfrozen) layers are loaded with random weights.

The scenarios where weights in the middle layer (L_4) are frozen have produced better accuracy when compared to freezing any one of the other layers (see **Table 3-3**). In other words, the

trained middle layer is able to produce better results, in spite of weights of the other layers being randomly initialised.

It is noteworthy to observe an increase in the accuracy when weights of layer seven (L_7) are frozen. This rise in accuracy is attributed to the fact that weights in L_7 obtained from the $T_{Best}(DNN_B)$ experiment have become too specific to the task. However, the overall accuracy is still less than the accuracy obtained with middle layer strategy.

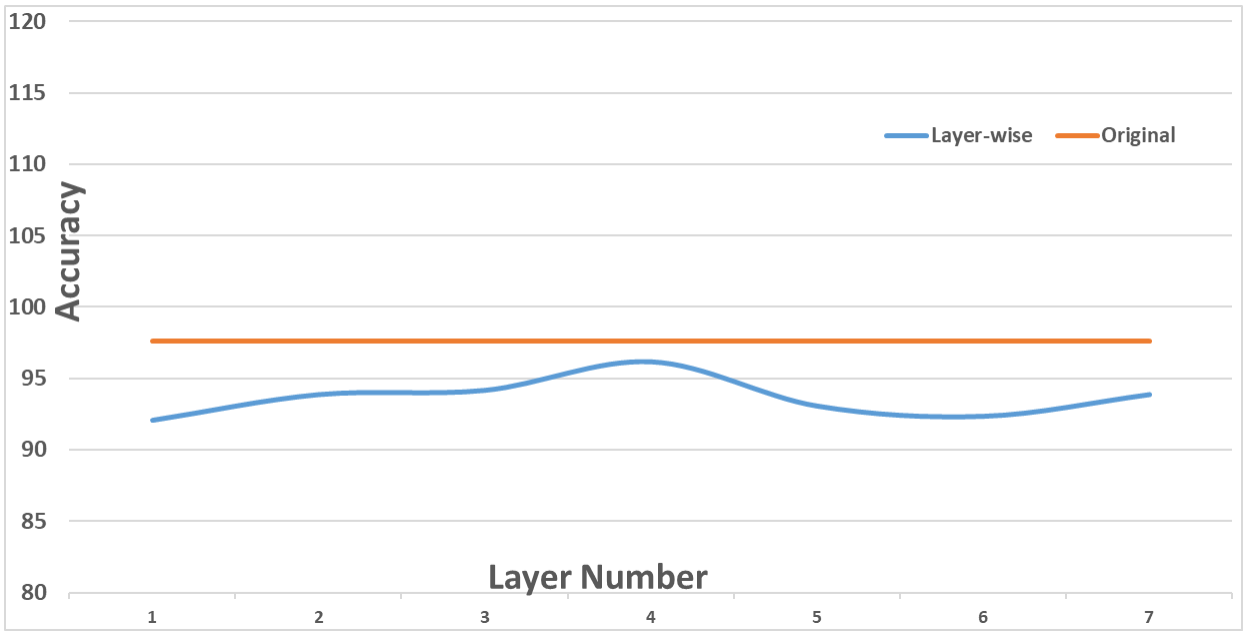


Figure 3-5: A comparison of classification accuracies on the synthetic dataset: The original classification accuracy is compared to the accuracies achieved when layers are frozen (previous experiment) one layer at a time. The highest (closest) accuracy to the original value is achieved when the middle layer weights are frozen. The experiment results are presented in **Table 3-3**.

The experiment was repeated with T_{SBest} (DNN_{SB}) and T_{Worst} (DNN_W) weights, freezing the middle layer strategy, had the highest impact on the classification accuracy once again.

To test the effect of the middle layer further, a weight value of 'zero' is randomly assigned to selected weights for every layer, one-layer at a time. From the results of this experiment, it has been concluded that the weights in the middle-layer are more sensitive and have highest impact on the classification accuracy.

Table 3-3: Experiment results: classification accuracies on synthetic datasets. The original classification results against results when weights are frozen, one layer at a type.

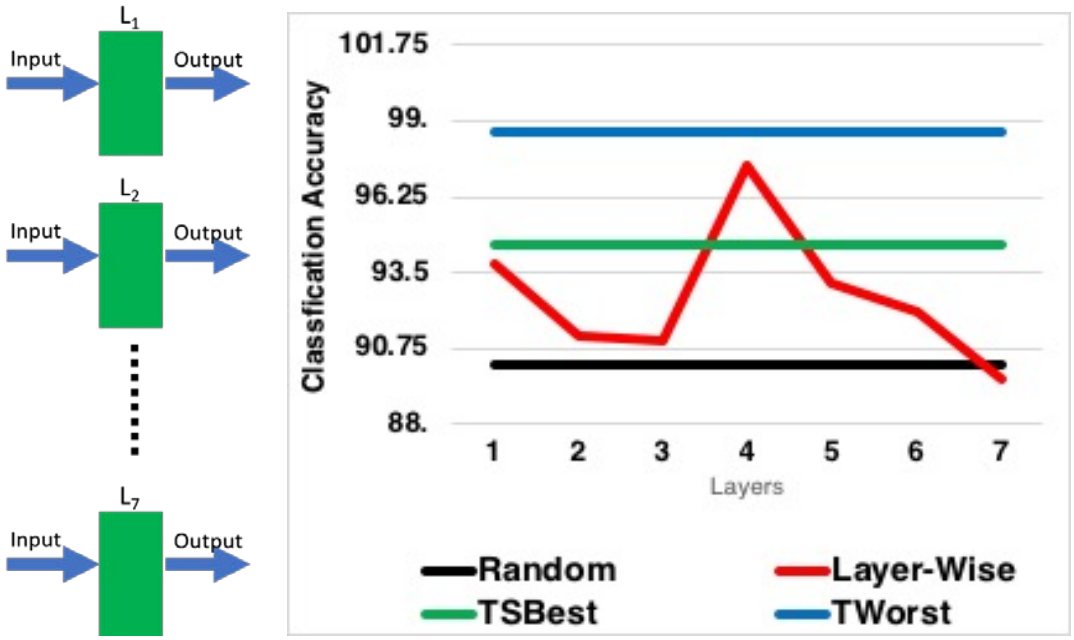
Layers No	Results (Layer -wise)	T-Test	Original Results	T-Test
1	92.1	0.011	97.6	0.0192
2	93.9	0.031		
3	94.2	0.054		
4	96.2	0.021		
5	93.1	0.05		
6	92.38	0.012		
7	93.9	0.091		

Similar results have been obtained for IRIS and Synthetic datasets with both the strategies. Each experiment is carried out 30 times, and the average classification accuracy is measured using T_{Best} (DNN_B). A comparison of classification accuracies between original value, and the layer-wise experiments are presented in Appendix J. The results are the averages of accuracies for all three datasets. The results further reiterate the importance of middle layer(s) irrespective of dataset used for the experiments.

3.4.2. Experiments with One-layered DNN:

As a result of the previous experiments, it is evident that some significant patterns are formed in the middle layer, the weights in the middle layer are comparatively more sensitive to changes and has significant impact on the accuracy of neural network than those in the other layers. A set of experiments are conducted that involve splitting of the 7-layered DNN into individual ANNs (seven ANNs, each consisting of one layer). The purpose of the experiment is to know which layer produces the highest accuracy among the seven layers. The layer that achieves highest accuracy could arguably be acknowledged as the most 'learnt' layer or layer that possesses 'better knowledge' when compared to the other layers in the network. This strategy is pictorially represented in **Figure 3-6 (a).** For each experiment, the classification accuracy is recorded, and the results are presented in **Figure 3-6 (b).**

From the experimental results it was observed that better accuracy is achieved for the ANN with middle layer weights of DNN.



(a) Experiment Design **(b) Classification Results**

Figure 3-6: (a) A one-layered ANN constructed by extracting individual layers from a trained DNN (b) A comparison of classification accuracies with original (random), best (T_{Best}), worst (T_{Worst}) and with one-layered ANN (Layer-Wise).

3.4.3. Removing One-layer at a time:

The purpose of this experiment is to study the impact of classification accuracy on a trained DNN when an individual layer is removed. The same trained DNN from the previous experiment is used for this experiment without any retraining. The strategy of removing the middle layer is shown in **Figure 3-7** where the removed layer is indicated in red. It was found that the classification accuracy is considerably reduced when middle layer was removed.

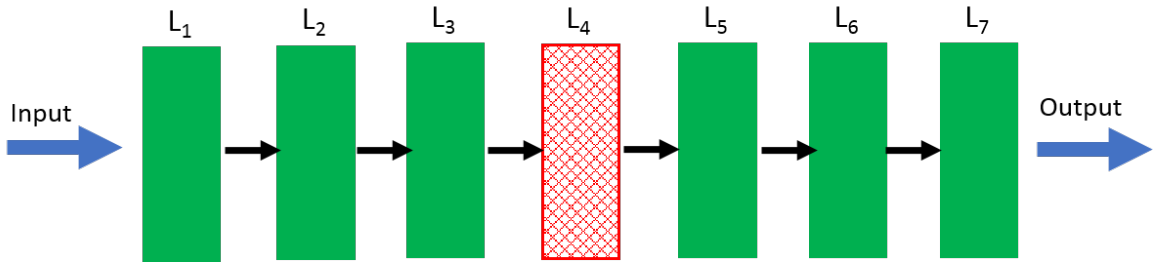


Figure 3-7: Experiment strategy where middle layer is removed from a trained DNN. The reduction in the classification accuracy when the middle layer is removed is far higher compared to the accuracy when any other layer is removed.

The reduction in accuracy was higher than for the removal of any other layer in the DNN. This reiterates the importance of the middle layer in a DNN.

3.5. Transferring Weights between two DNNs

The experiment results presented in the previous section suggest that there is a significant impact on the classification accuracy (positive or negative) when middle layer is manipulated. The first set of experiments where a layer's weights are frozen suggest that there is high contribution of middle layer weights to the overall accuracy compared to the other layers. Also, when the middle layer is removed, the classification accuracy is significantly reduced compared to the removal of other layers. The implications on classification accuracy is significant and has highest impact on the accuracy of DNN when the middle layer is removed. Considering the validation of the impact of middle layer, this section presents a set of experiments related to knowledge transfer between two DNNs to further investigate whether the weights in the middle layer actually contain any transferable 'knowledge'. To test this, a set of experiments are designed in which weights are transferred from one DNN to another DNN, one layer at a time, freezing all other layer weights to their original values (those obtained via the 7-layered DNN with no layer weights frozen). The details of various experiments and the results are reported in the following subsections.

3.5.1. Experiments with same number of layers:

For this set of experiment, the best (DNN_B) and worst (DNN_W) performing DNNs from the earlier experiments on MNIST dataset were used.

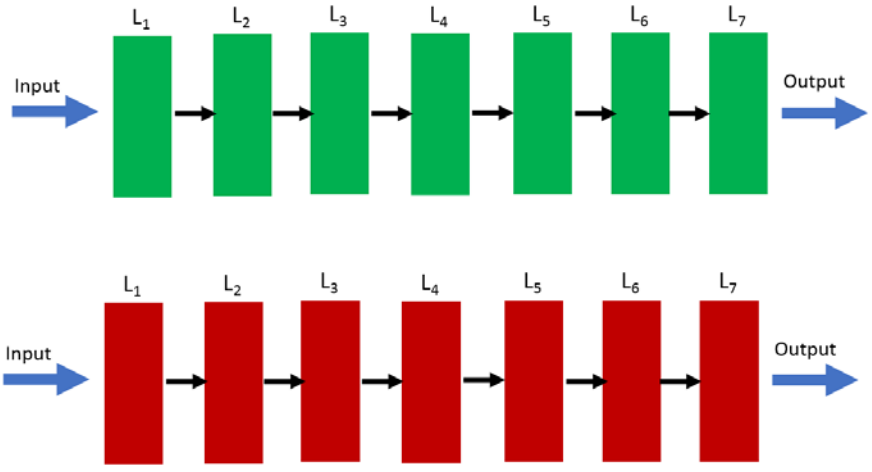


Figure 3-8: Representation of Best (DNN_B) and Worst (DNN_W) performing DNNs. The DNN with best and worst accuracies are shown in green and red respectively

In **Figure 3-8**, the best performing DNN_B with weights T_{Best} is represented in green whereas red represents the DNN with worst performance i.e., DNN_W with weights T_{Worst}. The weight values of the layer are transferred as it is, i.e., without any retraining.

The first set of experiments are carried out by replacing the weights of DNN_B with DNN_W weights, one layer at a time leaving all other layers as they are. **Figure 3-9** represents this transferring strategy forming new DNN called DNN_{BW} for each layer. The experimental results show that DNN_{BW} has better accuracy (average 93.2%) and performance than the original DNN_W (90.1%). Furthermore, when the middle layer of the best performing DNN_B is transferred to DNN_W, there is a considerable improvement seen in classification accuracy and performance when compared to accuracy values (presented below in **Figure 3-9 and Figure 3-11**) from the results of original experiments carried out with random weights.

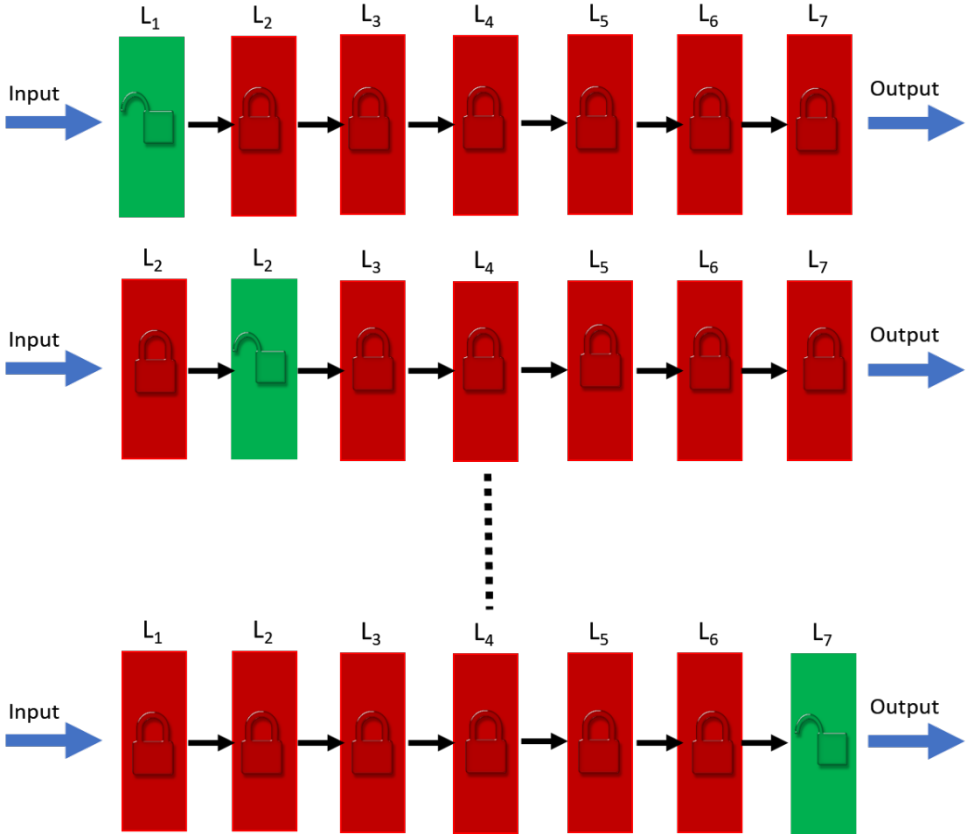


Figure 3-10: Transfer of weights (one layer) from the DNN with best classification accuracy (DNN_B) to the DNN with worst accuracy (DNN_W). The green bar indicates the layer selected from the DNN with the best classification accuracy (DNN_B).

The second set of experiments were conducted, with the opposite strategy, where the weights of DNN_W are replaced with those from DNN_B, one layer at a time. The results showed that there was a notable reduction in classification accuracy and execution time when the middle layer weights of DNN_B are transferred to the middle layer of DNN_W. Similar results are observed for the IRIS and Synthetic datasets. A comparison of classification accuracy and execution time is shown in **Figure 3-10** and **Figure 3-11** respectively. The detailed experiment results can be found in Appendix J.

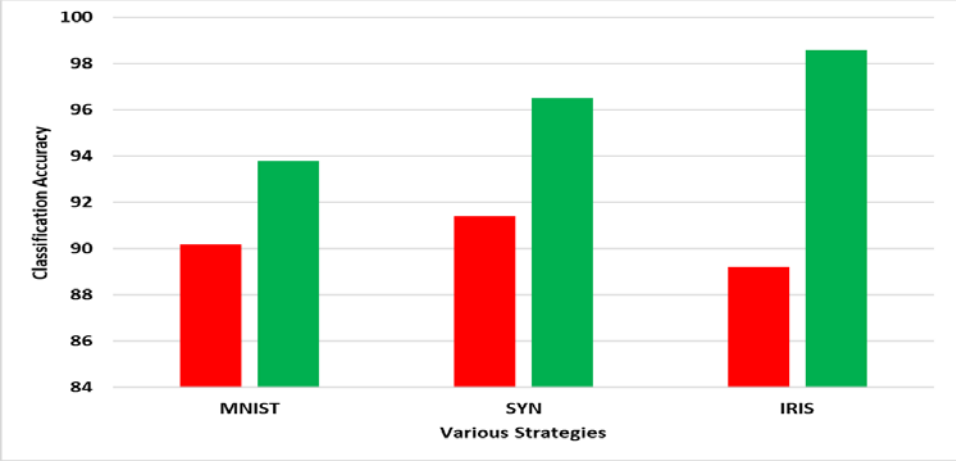


Figure 3-11: Classification accuracy for the three different datasets when the middle layer weights are transferred into an untrained DNN_B network. Red indicates the accuracy with regular (random) weights and green indicates the accuracy when middle layer weights are replaced with weights from the trained DNN_W middle layer.

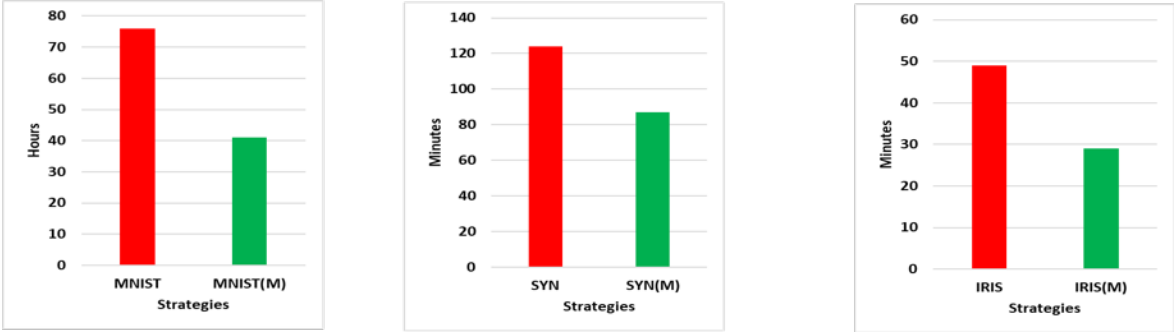


Figure 3-12: Execution time for the three different datasets when the middle layer (L_4) weights are transferred into an untrained DNN_W network. MINST (red) indicates the accuracy with regular weights and MNIST(M) green shows the accuracy when the middle layer weights are replaced with weights from the trained DNN_B (middle layer).

3.5.2. Experiments with different number of layers:

This experiment was conducted to investigate the topology based influence, i.e., number of hidden layers. A 7-layered DNN is used as the source from which the middle layer is transferred to 5-layered and 9-layered DNNs one at a time. The experiment scenario is presented in **Figure 3-12** and the results are shown in **Table 3-4**. Classification accuracies are measured before (without training) and after transfer. It was found that there is a considerably improvement in the classification accuracy of both the 5-layered and 9-layered DNNs after performing the transfer of weights.

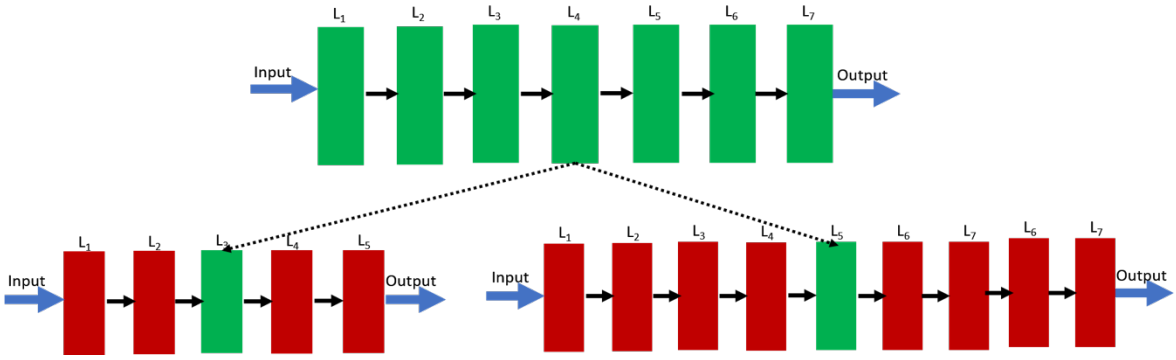


Figure 3-13: Transfer of weights strategy, from a 7-layer DNN, applied to a shallower (5-layer) and a deeper (9-layer) DNN.

Table 3-4: Experiment results for transfer of weights between different topologies. The classification results for 7-layered topology DNN after training is 98.5% for IRIS, 96.4% for MNIST and 98.3% for Synthetic dataset. The detailed statistics are presented in Appendix J3.

Topology	Classification Accuracies (%)					
	IRIS		MNIST		Synthetic	
	No training	Transferred	Random	Transferred	Random	Transferred
5-layered	51.2	76.5	33.5	80.2	44.6	85.2
9-layered	43.5	7.1.2	45.3	76.8	57.4	79.7

3.6. Feature Extraction and Transfer learning

This section presents various experiments designed to demonstrate the transfer of features through transfer of weights. The experiments are carried out using variety of datasets such as speaker identification [221], and a biological taxa-based synthetic dataset [29]. The experiments in this chapter are used to evaluate the feature learning capability of deep

architectures. The results of these experiments can be used to support the conjecture proposed in this thesis, that knowledge in a DNN exists in the form of representations in synaptic weights. The results also add further support to the theory that weights are transferable, and when transferred, they can provide a warm start, thus reducing the execution (training or testing) time of a DNN without negatively affecting the accuracy.

3.6.1. Feature Extraction for Speaker Identification

Speaker Recognition (SR) is the process of recognizing words or statements uttered. Automating this process, usually with AI techniques, is called Automatic Speaker Recognition (ASR). In terms of ML, SR is considered to be a pattern recognition problem. Speaker Identification (SID) is another NLP technique similar to SR but with a different objective. SID's objective is to identify a speaker based on their voice prints by comparing the voice profile of the speaker against existing profiles of various speakers [24]. SID systems have various applications such as user authorization (voiced password), personalized assistant, automatic mail direction, and many more.

The process of SID involves extracting and identifying unique characteristics of speech features from a group of speakers; hence, it is important to select the most efficient feature extraction approaches that best represent the speech features. One of the most complex aspects of feature extraction for SR is when the input utterances are infected with noise [38]. In layer-wise training, each layer of a DNN extracts features at different levels (hierarchically). A deep architecture is a hierarchical structure of multiple layers with each layer being self-trained to learn from the output of its preceding layer.

Deep learning algorithms were applied for hierarchical feature extraction to overcome the problem of noise in the utterance audio [13] and were found to be effective in improving SR performance [27, 21, 7]. Deep learning has been successful in various applications involving feature extraction for analysis and comparison [16, 35, 22, 12, 2]. The importance of feature extraction is well defined and implemented with DNNs. This makes it important to explore the various implementations of deep learning for SID.

Using DNNs to extract features from acoustic speech signals was initially proposed in 2014 [32]. This approach used a DNN to extract features instead of regular models for representing

voice frames (similar to an i-vector based approach). This DNN approach used a supervised training method instead of training with a CNN.

A DNN's topology is designed with each layer working at the acoustic frame level. Each frame of speaker's voice input is fed to this DNN and the output activation values of the last layer is accumulated as a representation of that particular speaker. This representation of a speaker is called a d-vector. Usually, DNN uses a Softmax (supervised) layer for output. In this approach, the output from the last layer is employed instead of a regular Softmax layer. Removing the need for an output layer enables the DDN's the size to be reduced by one layer. This might seem small, but the reduction in size by one layer is quite significant for DNNs. Further, removing the Softmax layer will enable better generalization and thus allow a DNN to extract compact speaker models for unknown speakers [32]. Unlike the regular SID approaches, this approach does not use any adaptation technique for extracting known features in the training phase. Instead, this approach uses a DNN model for extracting specific features in both the enrolment and matching phases.

A typical DNN classifier for speaker recognition uses a set of stacked features as input typically from feature extraction approaches such as MFCC [10]. The features used in initial DNN approaches are short frame based with 20 milliseconds (ms) with a context of ten frames for each segment of input. Each DNN is expected to predict a probability of the speaker for the input frames that are fed to DNN. To obtain the overall decision comprising of multiple frames, each prediction of DNN can be averaged out to find the speaker class. An alternate approach is using two different DNNs, one for frame level prediction followed by the second one for classification.

The speakers' data are extracted from the Census (AN4) speech database provided by Carnegie Mellon University [221]. The database consists of 1158 16KHz speech samples collected from 84 subjects of both genders (male and female) using 16-bit linear sampling. The dataset that is publicly available has only 948 samples for training (from 53 males and 21 female subjects) and 130 samples for testing (from seven male and three female subjects). For this experiment, the training and testing samples are combined, out of which 600 random samples are selected to avoid the imbalance in the gender and any other unknown factors since the criteria for the separation is not mentioned in the repository. The speech features are extracted as Mel Frequency Cepstral Coefficients (MFCCs) using open source code libraries in MATLAB

2016a [222]. Sixteen MFCCs were extracted from the inputs for each sample in the dataset. Once the features were extracted, different classifiers are defined using three DAEs with one, three, and five layers. Using multiple and divergent DAEs is necessary in order to test the accuracy rates and to determine the necessity of the number of hidden layers in DNNs as discussed in the literature [52, 182].

The DAEs are trained layer-wise using scaled conjugate gradient descent (SGD) whereas the baseline ANNs are trained using regular back propagation with a sigmoid activation function. The training, validation and testing datasets are divided as 70%, 15% and 15% of the data respectively. DAEs with one, three, and five layers are used with varying numbers of hidden nodes. The first layer always has 16 (same as the number of input MFCC features), and the subsequent layer has 20 hidden nodes (refer to **Table 3-5.**). The training time is set to 100 epochs. ANNs with one, two, and three layers are used with 16, 12, and 22 hidden nodes. The learning rate and momentum were fixed to 0.05 and 0.2 respectively for all the neural networks but 500 training epochs are performed for the baseline ANN based SID. All experiments are conducted in MATLAB 2016a using the Neural Network toolbox and deep learning modules on a Microsoft windows 7 PC with Intel dual core 3.4 GHz and 16 GB RAM.

Six different experiments are conducted: three experiments using the baseline SID systems and another three with DAE-based SIDs. Each experiment is repeated 25 times, and the average results are presented in **Table 3-5.** Along with model accuracy, the results are validated using T-Test to show that there is no significant difference among the results.

Table 3-5: Experiment details: Architecture and experimental results (accuracy and error) for the ANN and DAE classifiers.

Experiment No	Classifier	No. of Hidden Layers	Hidden Layer Number	Number of Neurons	Root Mean Squared Error	Accuracy (%)	T-Test
1	ANN	1	1	16	0.24	83.7	0.015
2		2	1	16	0.39	71.15	0.029
			2	22			
3		3	1	16	4.15	39.0	0.03
			2	12			
			3	22			
4	DAE	1	1	16	0.19	79.4	0.027
5		**3**	**1**	**16**	**0.112**	**98.8**	**0.003**
			2	**20**			
			3	**20**			
6		5	1	16	0.34	69.16	0.022
			2	20			
			3	20			
			4	20			
			5	20			

The experiment results indicate that the DAE with three layers outperformed all other classifiers. The lowest accuracy of 39% is provided by the ANN with three hidden layers with a difference of ~59% when compared to the highest accuracy experiment using a 3-layer DAE of 98.8%. The worst performing DAE achieved 69.16% which is over 30% better than the worst performing ANN (39%). The best results for an ANN are only 83.7%, 15% lower in accuracy than the best DAE results. It is noteworthy to observe that the number of hidden layers has a considerable effect on the outcome this effect which has also been speculated on in earlier research [29]. The reason for the trend of reducing accuracy with increasing number of ANN

hidden layers is attributed to the known convergence issues of BP. The BP convergence is due to the complexity of the SID task [223].

The results obtained using DAE employed by the feature learning through encoding features as condensed representations provides an evidence on the possibility of knowledge being existed as underlying representations.

3.6.2. Transfer Learning Experiments

These experiments are carried out using a synthetic hierarchical dataset with known feature hierarchies (see Section 3.2 for the detail of dataset). In this dataset each sample is a single 8-bit value. DAEN ($DAEN_1$) is trained with an uncorrupted dataset until 100% classification accuracy is achieved. The dataset is then corrupted to distort the existing relationship in the within the data by changing randomly selected values (bits). The dataset is further corrupted replacing some randomly selected values with NaN. Classification is again performed on the corrupted dataset using a second DAEN ($DAEN_2$). Then finally, the weights of $DAEN_2$ are replaced with the weights of first $DAEN_1$ making it $DAEN_R$ and classification is performed without training the second DAEN. In other words, the weights are transferred from first DAEN to second DAEN for all autoencoders.

A three-layered Deep Autoencoder Network (DAEN) is used to perform the experiments and SGD algorithm is used for training. A symmetric node count of 50 is chosen for each layer of all the autoencoders. Fifty symmetric nodes are chosen due to the efficiency observed in the hierarchical data classification experiments when compared to asymmetry node count i.e., when number of hidden nodes in the layers are not equal [29]. SVM is used as the classifier for the Softmax layer. Each autoencoder is trained for 400, 200, and 100 epochs and overall supervised training for the Softmax layers is performed for 100 epochs. The hierarchical dataset used for the experiments consists of 90 samples of six different species. Each experiment was performed 25 times. The main reason for selecting the hierarchical dataset is because it consists of known features. Further, the dataset is constructed using distributed representation which makes it easy to disturb (corrupt) the features and hierarchies.

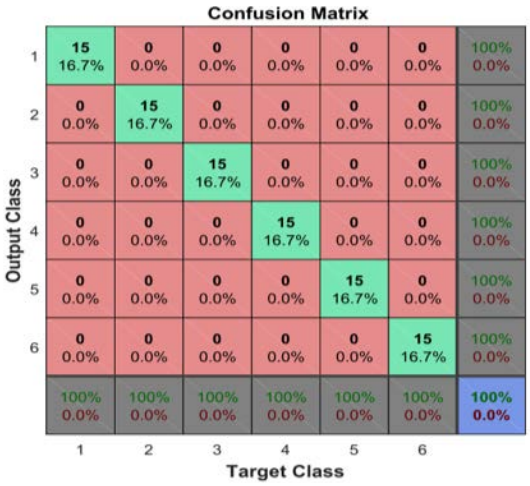


Figure 3-14: Confusion matrix for classification experiment with the original (uncorrupted) dataset

The classification results for various DAENs and data sets are presented in **Table 3-6.** The confusion matrix for the experiment results with the uncorrupted dataset and corrupted dataset are presented in **Figure 3-13** and **Figure 3-14** respectively. For the uncorrupted dataset the classification accuracy is 100% whereas when the dataset is damaged accuracy fell to 56.7% as a result of the corrupted data and distorted hierarchies (relationships).

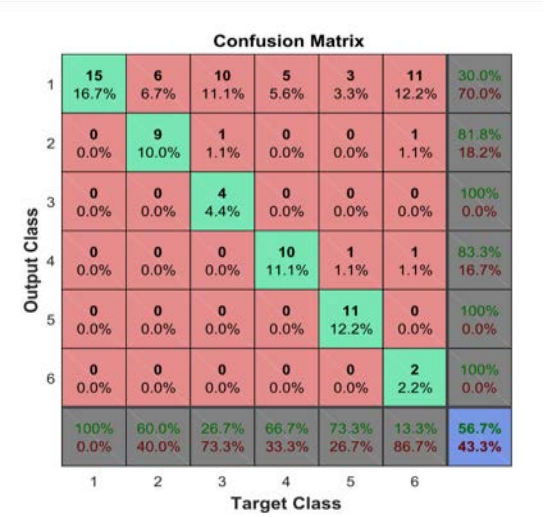


Figure 3-15: Confusion matrix for classification results with corrupted dataset. The data is distorted to reduce the classification accuracy

However, when the classification experiment is performed with the same corrupted dataset after transfer of weights (from $DAEN_1$ to $DAEN_2$) with new $DAEN_R$, a significant rise of 22.2% in the classification accuracy to 78.9% is observed as shown in **Figure 3-15.**

Table 3-6: Classification results for the corrupted (C) and uncorrupted (UC) dataset

Deep autoencoder	Dataset	Accuracy (%)	Train RMSE	Test RMSE
$DAEN_1$	UC	100%	0.003	0.0034
$DAEN_2$	C	56.7%	0.663	0.5113
$DAEN_R$	C	78.9%		0.252

One reason for this increase might be that 'some knowledge' is transferred unknowingly when weights are transferred. It is a fact that the principle components of any neural network are weights, and thus the conjecture that they contain knowledge seems reasonable. It is worthy considering that a weight is just numeric values and might not be significant in itself, but collective weights might have some kind of hidden representation(s) that may constitute knowledge and what is getting transferred in the knowledge transfer process. So, these hidden representations might constitute some form of knowledge which is being transferred and is responsible for improving the accuracy observed in the experiments undertaken in this research.

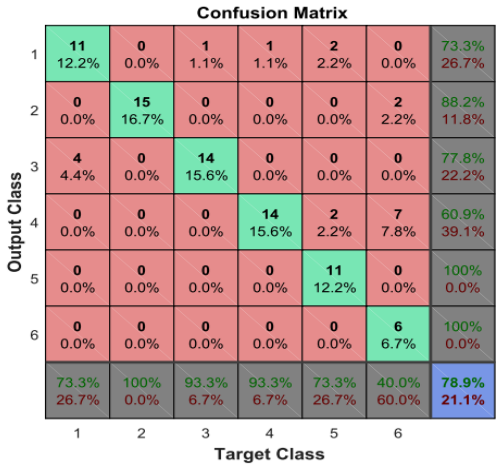


Figure 3-16: Confusion matrix for classification results for the corrupted dataset after transfer of weights from DAE. Note the improved accuracy which is comparable to that of the classification of the original dataset (**Figure 3-14**).

When an autoencoder is able to reconstruct the input, the weights might be storing a structure or some form of pattern in the weights. It appears that this structure, when transferred to another DAE, can be utilized to replace corrupted values such that the samples are classified correctly. However, it is still an open question as to how such representations can be extracted from weights.

3.7. DNN Optimisation by Reducing number of layers

This experiment is carried out using MNIST hand-written character recognition dataset. The MNIST dataset is divided into three parts 70% for training (TrainDs), 10% for testing (TestDs), 20% for recalibration (RDs). Each experiment is carried out 25 times.

Firstly, experiments are carried out using only 10,000 images from a total of 60,000 images of MNIST data. A 7-layered DNN_1 is trained using TrainDs and tested with $TestDs_1$ with a classification accuracy of 97.6%. A second DNN_2 with 1-layer is constructed by transferring the middle layer of DNN_1 is found to produce 97.2% accuracy with TestDs.

In the second set of experiments, the entire dataset of 60,000 images is used. DNN_1 is able to achieve an accuracy of 91.2% with sensitivity and specificity of 71.2% and 81% respectively. When the same experiment is carried on DNN_2 (1-layer), the accuracy reduced to 67.1%. In the next experiment, the three layers (1 middle and 1 each from either side) of DNN_1 are extracted and these layers made up a DNN_2 with 3-layers. There was a little improvement in classification accuracy (74.2% c.f. 71.2%). DNN_2 is retrained with RDs dataset allowing only 5% of variation in the weight values. This improved the classification accuracy to 91% with better sensitivity and specificity at 82.4% (11.2% more) and 89.8% (8.8% more) respectively.

When these experiments are repeated using the synthetic hierarchical dataset, DNN_2 with only one layer was able to achieve same classification accuracy with that of a 5-layered DNN_1. It is noteworthy that the classification accuracy is always better with only one layer. Further research and analysis are required to explore the reasons for this behaviour.

From the experiment results presented in Sections 3.1 through 3.6, it is highly likely that the middle layer possesses knowledge that significantly affects the overall accuracy of the neural network compared to the other layers. Therefore, this section establishes, at least in the context of the experiments presented, that the middle layer provides highest contribution to the overall efficiency of DNNs.

3.7.1. Knowledge Components and Weights of the Weights

The conjecture proposed in this section states that the *knowledge component* is a subset or a proportion of total weights in a layer and this component constitutes knowledge that

significantly impacts the classification accuracy. This aspect could be associated with the principle of dropout where (insignificant) weights are dropped randomly to improve learning capability and speed. The subset of weights, which are named as knowledge components are extracted from the fullest of weights from the middle layer. These components are built on the mathematical principles of PCA and FA but with significant difference which will be presented in next chapter.

The concept of Weights of the Weights (WofW) is quite significant in extracting the subset of weights (aka knowledge components). This section also presents experimental evaluation of the principle of WofW and the experimental results are presented at the later stages of this section.

The concept of the WofW is conceived and developed in this research and is founded on a principle of dimensionality reduction by submerging two layers of a DNN as shown in **Figure 3-16**. The weights in two layers are fed into an ANN to obtain a new set of weight values that possess the qualities of both weights. Based on the encoding principle of autoencoders, the weights are fed into a neural network and the network is trained for weight values that produce highest accuracy when used in a neural network middle layer. In other words, a large set of weights is encoded into a small set of weights similar to dimensionality reduction that occurs in autoencoders.

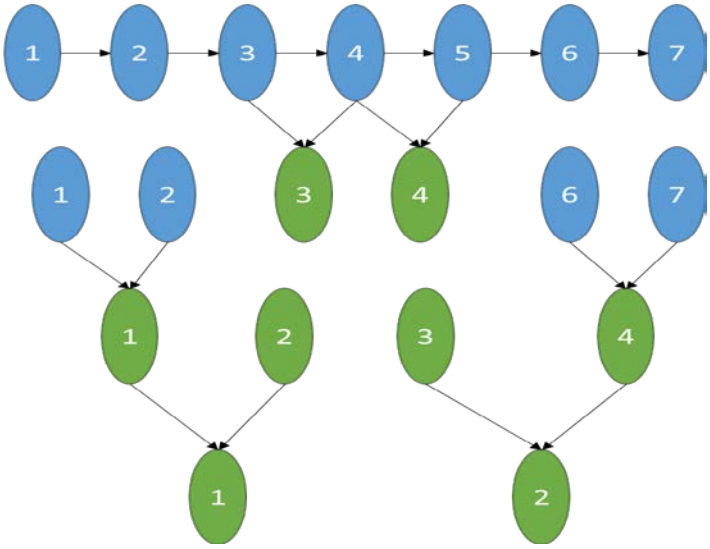


Figure 3-17: Representation of scenario where the combination of layers (weights) higher level to deduce weights of the weights. In this scenario, the weights of two layers are fed into neural network to generate a new set of weights (with same number as one of the layers).

Thus, the middle-layer weights in the condensed form are similar to WofW and an experimental verification using WofW scenario can re-affirm the proposed hypothesis that the middle layer consists of significant knowledge in the form of underlying representations in the weights. The aspect of WofW being efficient also strengthens the proposed Blossom Effect which states that the features are not lost but folded in as representations in the condensed weights of the middle layer.

Classification experiments with a varying number of weights extracted from each layer are conducted. This set of experiments follows the same concept of extracting weights of the weights (WofW) with different number of hidden nodes for each layer. In this case, the entire 7-layer DNN network is employed. The experimental scenario is presented in **Figure 3-17** and the experiment results are presented in **Table 3-7.**

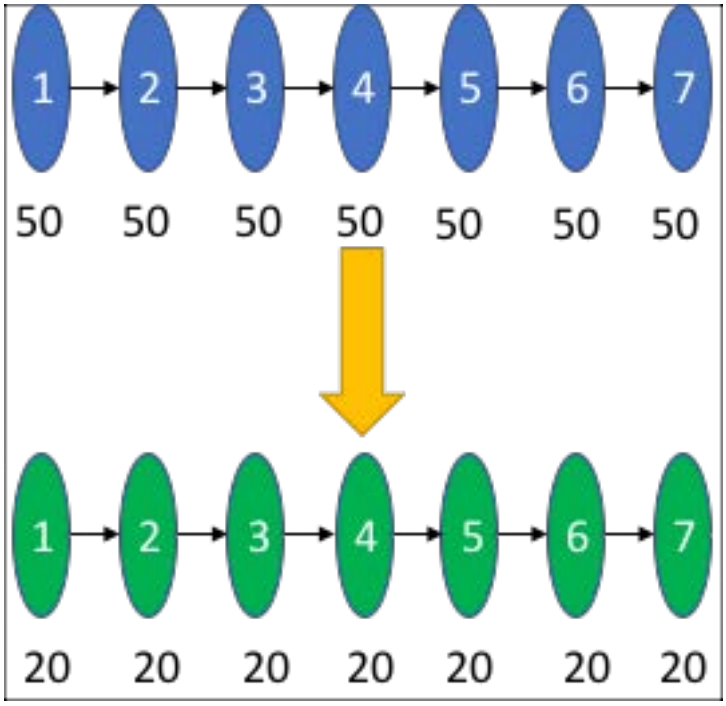


Figure 3-18: Experiment scenario for extracting weights of the weights (WofW). A 7-layer DNN with 50 nodes in each hidden layer is reduced to a seven layer neural network with 20 nodes in each layer.

From the results, it can be noticed that, there are some weights that are more influential than the others. It is to be noted that the number of nodes to be extracted are determined randomly. Thus, there is now a requirement to propose a systematic approach to determine the number of weights i.e., transferable components represented in weights as a model.

Table 3-8: Experiment Result using 'Weights of Weights' with reducing number of weights. The results show that when number of nodes are reduced, the classification accuracy without retraining is low since some nodes might have been lost. Whereas with WofW approach, , the classification accuracy is higher with the same number of nodes.

Nodes in each layer	**Classification accuracy (%) (without retraining)**	**Classification accuracy (%)**
50	94.2	-
40	87.2	93.1
30	61.6	94.6
20	44.8	78.1
10	32.1	82.1
13 (No. input attributes)	29.6	88.0

3.7.2. Efficiency of the Weights of Weights

The experimental results obtained shows that Weights of Weights (WofW) can retain the knowledge in that the use of WofW improves the accuracy of classification. The results also show that there exists some segregation between significant weights and insignificant or less influential weights. In other words, though the weights are just numerical values, they possess some internal representation in the form of patterns. These representations can be condensed into significant weights which may be less in number but holds majority of the knowledge.

In order to be able to extract the optimised weights between two layers, another set of experiments is devised and conducted. Starting from the centre of the DNN, optimised weights are generated to reduce the number of layers.

The middle layers of three, four and five of a 7-layered DNN are chosen for this experiment. To extract optimised weights, layer three is used as input for an autoencoder to obtain the weights of layer four as output. The hidden layer weights are fine-tuned to obtain new weights that are optimised weights for producing layer 4 weights from layer 3 weights. This is similar to the autoencoder experiment explained in Section 3.6.1 The experiment is carried out until only two layers are left.

The results of these experiments using different numbers of layers are presented as **Table 3-9**. With complete 7 layers the classification accuracy is 94.2%. Classification accuracy is reduced as number of layers are reduced using weight extraction. However, there is considerable improvement in the results when the entire network is retrained (supervised).

Table 3-9: Using Weights of Weights with and without retraining the weights.

Layers	Classification Accuracy (%) (without retraining weights)	T-Test	Classification Accuracy (%)	T-Test
7	94.2	0.018	-	-
6	72.5	0.005	93.1	0.0089
4	43.1	0.016	79.2	0.0051
2	22.7	0.021	81.1	0.0014

The results (**Table 3-9**) show that extracting weights by combining layers is of no use as the feature representations that are present in the weights are lost due to recombining or changing the combinations of the layers. This is further emphasised by the improvement of results when retrained (supervised) weights are used. It is worth noting that when conducting the experiments, the required number of retraining cycles are increased as the number of layers is reduced.

3.7.3. Relationship between input features and weights

The initial experiments presented in Chapter 3 thus far have provided the evidence that hidden representations exist in DNNs and these hidden representations are mathematical constructs in the form of transferable components. These components are a direct representation of input features that influence the learning mechanism of DNNs. From the experimental results, it is concluded that the middle layer(s) of the DNN is most significant, most sensitive to changes and it is this middle layer that influences the overall accuracy.

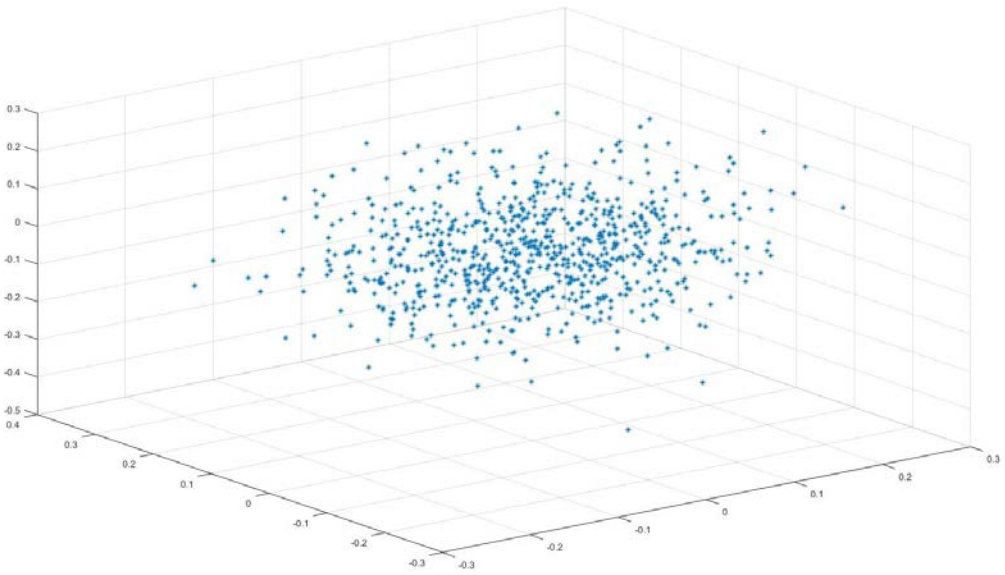
Further, these transferable knowledge components form a model, that when transferred, can lead to remarkable improvements in learning. The key challenge in this research is identifying the relationship between features and a DNN's weights. At this stage this relationship is unknown. However, with these initial results, the transfer of features is occurring when weights in the middle layer of a DNN are transferred to another DNN.

Since the features are spread across the various layers of DNN, it cannot be concluded that the weights in the middle layer are representing subsets of features or significant features directly. It is already proven with the set of initial experiments that weights in the middle layer have the highest influence on the accuracy. Since all the weights are represented in features and all features are represented in some form of weights, the only possible conclusion is that the weights in the middle layer are representing skeletal features or a prototype of features rather than features themselves. Then, is it these prototypes of features that constitutes 'Deep Knowledge'? It is my conjecture that it is. It is this 'Deep Knowledge' that acts as an underlying representation of a domain model that can be transferred to another DNN. Therefore, the middle layer is critical since it is representing the features that are necessary for the transferable DNN model. There may exist many such models in the DNN within the same domain. With

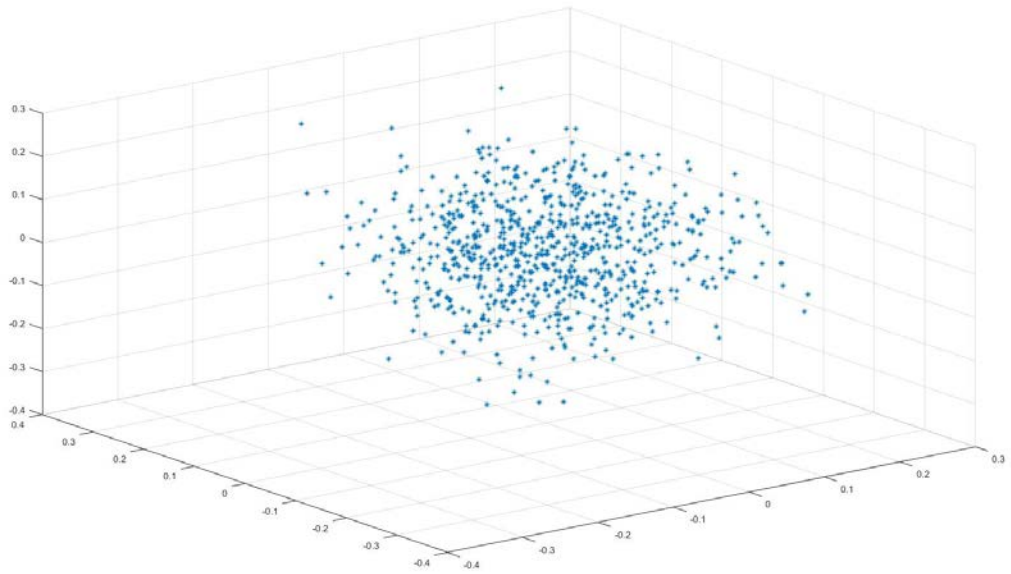
this, the DNNs can be considered as a model that can be generalised and a domain specific DNN model could be generated and implemented for various problems pertaining to the same domain. This is the model that is transferred as knowledge model between two DNNs.

3.8. Discussion

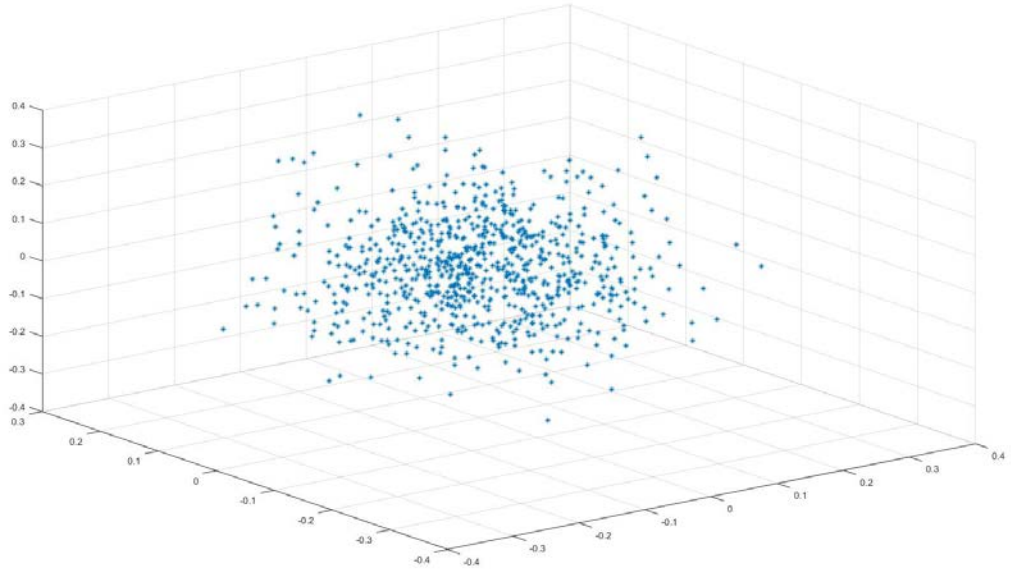
The initial projections using the weights obtained from the experiments (7-layered DNN using MNIST that attained best results) show that the middle layer of the 7-layered DNN is significant and seems to be performing some funnelling of features between the layers on either side. Considering the fact that the middle layer is equidistant from input and output, this behaviour can be justified to some extent. It may be argued that the layers near the input are too input specific to contain any useful 'knowledge' whereas the layers near the output are more problem-specific and therefore not useful. Hence, the middle layer may be acting as a transition layer where the transformation of weights is occurring. The hidden representation in the middle layer could be considered to be un-biased or neutral with respect to input values and problem specific classes.



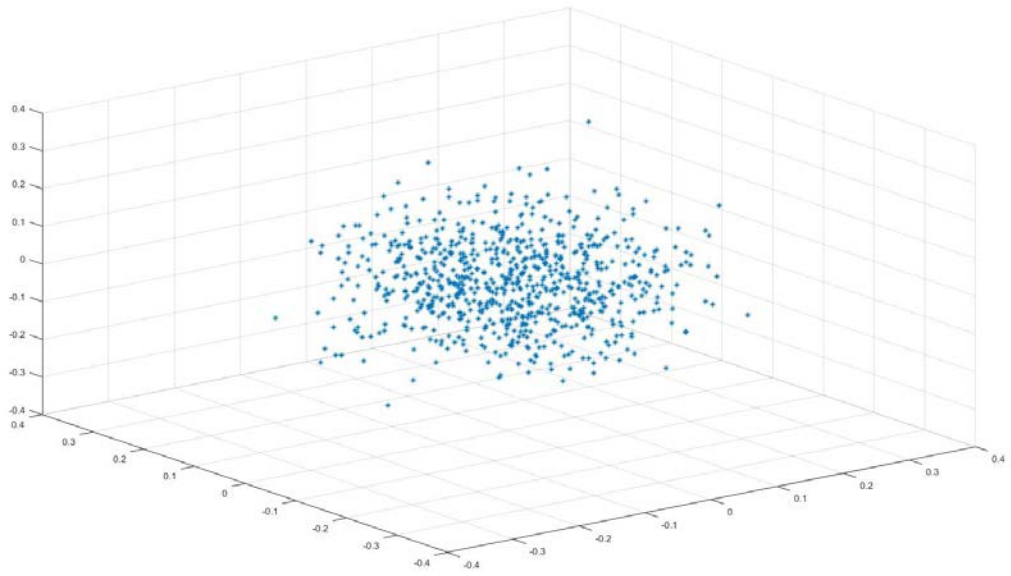
(a) 1


0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.4
-0.3
-0.2
-0.1
0
0.1
0.2
0.3

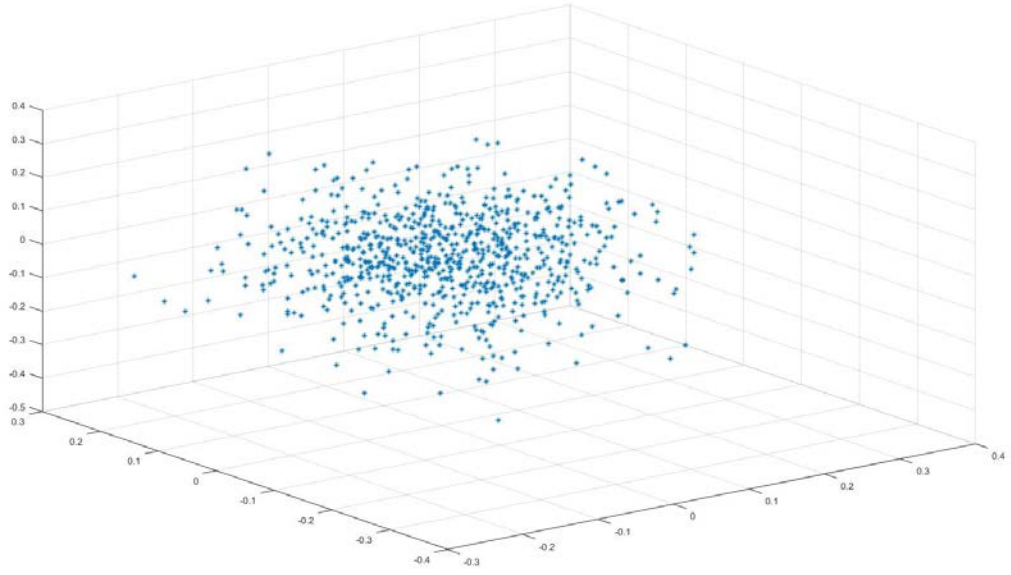
(b) 2


0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.4
-0.3
-0.2
-0.1
0
0.1
0.2
0.3
0.4

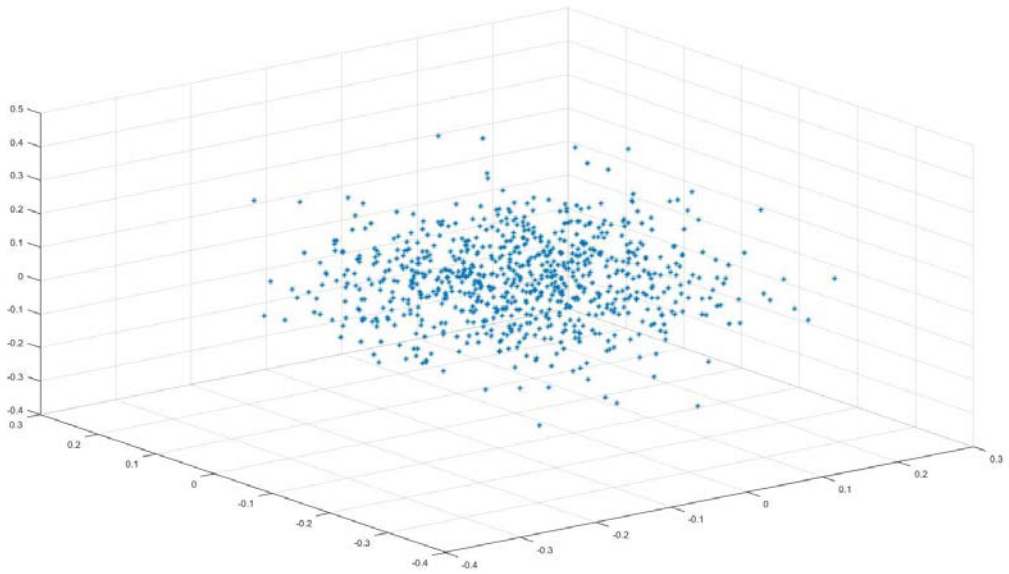
(c) 3


0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.4
-0.3
-0.2
-0.1
0
0.1
0.2
0.3
0.4

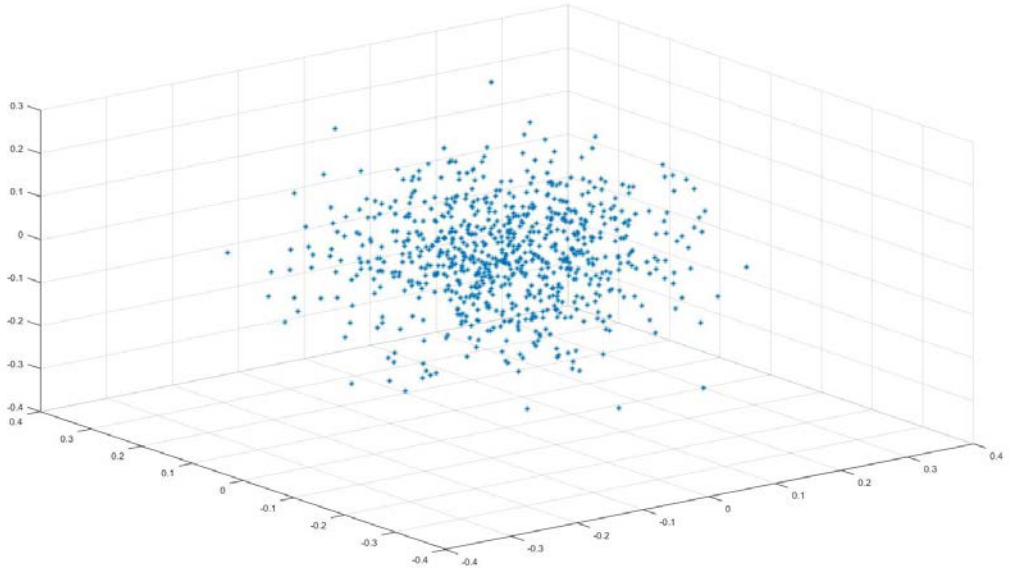
(d) 4


0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.3
-0.2
-0.1
0
0.1
0.2
0.3
0.4

(e) 5



(f) 6



(g) 7

Figure 3-19: Projection of weights of a trained DNN with 7 layers. The alphabets from (a) through (g) indicating the seven layers (numbered in the picture). The weights are projected in 3 dimensions to identify the relative distance. It can be noted that the weights are more concentrated (folded – the Blossom Effect) in layer four.

To establish the fact that the formation of weights in the middle layer is different to the other layers, the weight values of various layers are projected in a 3-dimensional scatter graph as shown in **Figure 3-18**. It is noteworthy to observe that, for all the layers, the weight values are dispersed in the layers except for the middle layer (**Figure 3-18** (d)**)** where the weights values are more concentrated (condensed representations). A graphical representation of variance for each layer of weights is presented as **Figure 3-19**. The variance graph clearly indicates that the weights in the middle layer have the least variance compared to other layers.

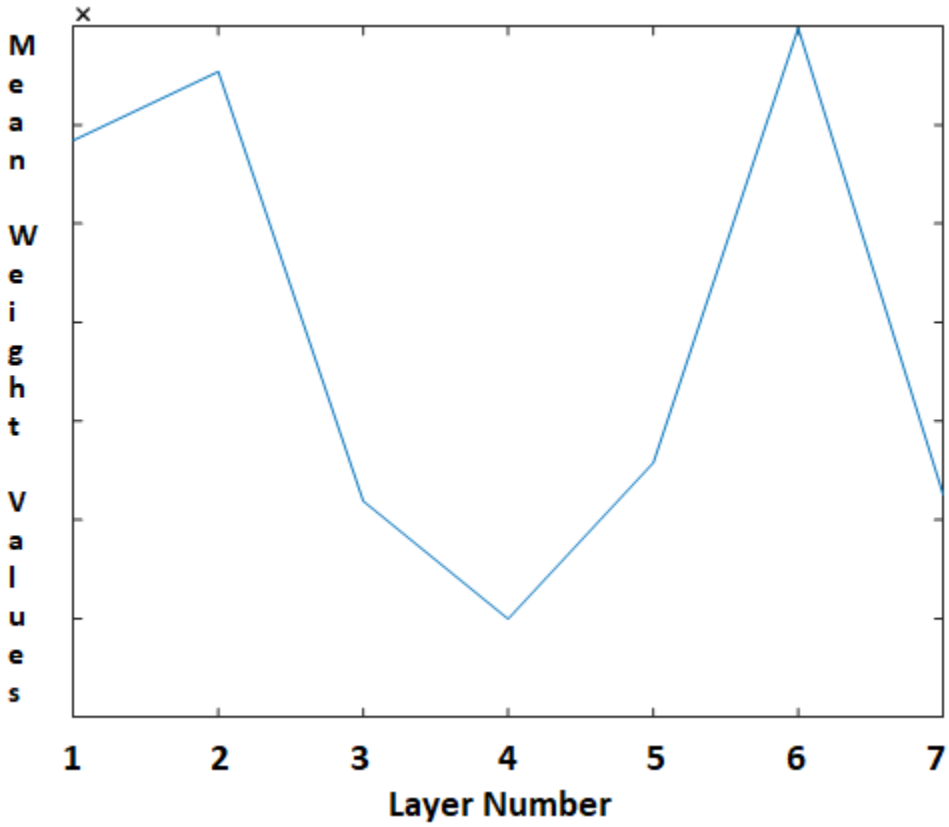


Figure 3-20: The graph portraying the projection of weights for each layer of the 7-layered DNN. The minimum variance resembles the variance of them being less that other layers. The variance value of the 7 layers is diminishing since the weights have become problem specific.

Though transfers of middle layer weights are significant, it is necessary to investigate what exactly (features or representations or yet unknown patterns) is being transferred when weights are moved from one DNN to another. The relationship between input features and weights of a DNN is established by the experiments in this study. Therefore, there must exist some hidden representations in the weights that will be named 'Deep Representations.' These representations constitute influential features that are affecting the accuracy and functionality of the DNN that is receiving weights. When weights are transferred, it is the features in the

form of ‘Deep Representations’ that are being transferred. It cannot be concluded that the features in the middle layer are ‘the only important features’, since all the important features may not be confined to a single layer. So, it might be skeletal features or prototype of features in some form that are being transferred unknowingly when weights are transferred. This specific set of transferable skeletal or prototype features is the ‘Deep Knowledge’ that is buried as ‘Deep Representations’ in the deepest layer(s) of DNN. A pictorial representation of this scenario is shown in **Figure 3-20.**

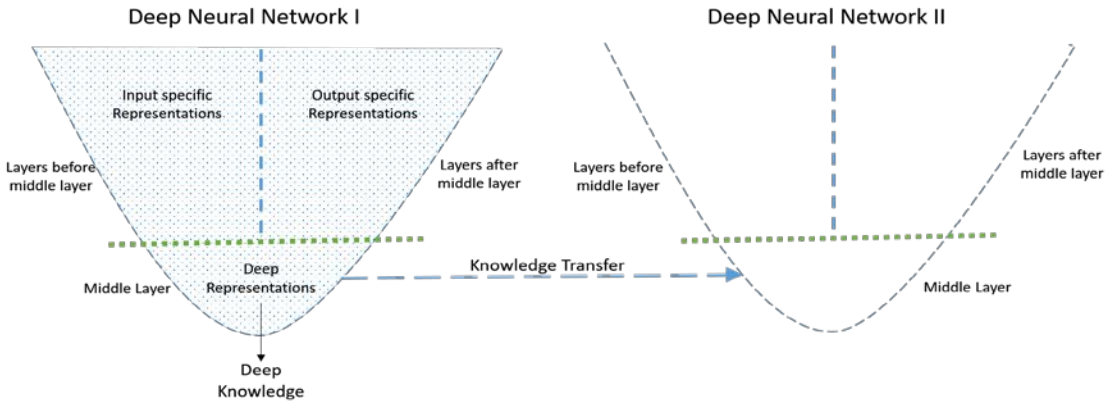


Figure 3-21: The Deep Representations and Knowledge Transfer scenario: The middle layer holds the knowledge as deep representations and as such yields the highest accuracy when transferred into another DNN.

The existence of multiple deep representations in the weights for the same problem cannot be denied. Another underlying fact is that these deep representations may be based on learning or the domain or any other factors that influence a DNN’s functionality. However, it is evident that hidden representations do exist and can contribute to a transferable model which influences the learning and operations of DNNs. Further empirical research is required to identify, extract and analyse such representations.

This chapter provides the foundations for the research carried out in this thesis. Identifying the impact of weights in various layers has paved way for further research on the transfer of knowledge. The importance of number of nodes and the capability of ANNs (or DNNs) is significant and could give a direction for establishing a relationship between input features and weights in a hidden layer. The weight variance graph (**Figure 3-19**) shows a clear indication of significant correlation of weights that constitute knowledge that is transferred.

To eliminate any technical or domain based bias, a variety of deep architectures as well as diversified datasets are used. The reason for not using CNNs is due to the fact that CNNs are purely used for image analysis and there has been a transfer of layers approach already undertaken in this domain [63, 209] (the transfer of layers in CNNs in image recognition was discussed in the Chapter 2 Section 2.6 of this thesis). However, the proposed model is tested using CNNs confined to the scope of the research (Chapter 6 Section 6.152 of this thesis). Furthermore, the architecture of CNN is different and has been a point of discussion on the category of deep architectures that CNN belongs to.

The main purpose of testing initial hypothesis is to provide the evidence of the importance of the middle layer. Initial experiments are carried out on MNIST, IRIS and the synthetic datasets for a DNN, and are the results are presented in this chapter. ImageNet and CIFAR-10 datasets are used for further experiments, and the results are presented in the next chapter. Experiments are also carried out using the TIMIT dataset and the results are presented in the Appendix E. The proposed hypothesis is thus tested and found to be true.

The experiment results presented in this chapter proved that when the middle layer of a trained network is transferred to an untrained network, the untrained network will produce significant improvements in the classification results. In addition, the execution time is also considerably reduced.

3.9. Chapter Summary

This chapter investigates the plausibility of the research through preliminary investigation. The outcomes of the chapter can be summarised as follows.

- All the features are represented in all the weights of DNN and vice versa. The middle layer(s) is(are) significant among the layers of the DNN, and weights in the middle layer are more sensitive to changes. The middle layer has highest positive impact on the accuracy and functionality of DNNs, and removing this layer reduces the efficiency of the DNN.
- The weights in the middle layer are in a submerged state where the features are folded in, and they gradually become folded out (re-emerge) as they move away from the

middle layer. These condensed middle layer representations can be verified through the experiment results of the Weights of Weights (WofW).

- Weights in the middle layers constitute features in the form of hidden representations. The middle layer possesses the knowledge in the form of underlying discrete representations in the weights and DNN is able to learn these representations through training. These features are transferred unknowingly at the time of transfer of knowledge.

In the next chapter, research hypothesis is proposed towards identifying the relationship between input features and DNN weights and the presence of knowledge components in various layers (in line with research problem in the Chapter 1 Section 1.7). The chapter also presents how the hypothesis is tested and states the research method adopted for testing and verification.

Chapter 4 Hypothesis and Research Approach

4.1. Introduction

The findings of the previous chapter evaluated the importance of various layers through classification experiments and the middle layer(s) and found to be of greater importance compared to the other layers: that the middle layer possesses transferable knowledge. The findings also provide a new research direction in the form of two hypotheses that drive the rest of the research presented in this thesis. This chapter presents the hypotheses, and an account of various experiments designed to test each hypothesis.

4.2. Proposed Hypotheses

The principle hypothesis of this research is that

"There exists a Blossom Effect in DNNs in which the input features are folded into the middle layer and then will be folded out through the layers thereafter resembling the sacred lotus flower."

4.3. Hypothesis 1 (H1)

H_1: For a given input with n features with x% of least relevant information (noise or distortion) that significantly effects the accuracy, x is distributed among L/2 layers in which there exists c components in the middle layer (L_m) such that when x is minimised, $c \leq n$.

4.3.1. Scenario 1: x = 0 (no noise, clean data)

When $x = 0\%$ there is no noise and only independent and clearly differential features then $n = c$ where L_m has near equal variance in a short time for all layers, then the funnel of layers will turn into a pipe with equally distributed components.

Result:

1. *All layers are equally important*
2. *Knowledge is distributed among all the layers.*

4.3.2. Scenario 2: x > 0 (some noise with few overlapping features)

When $x > 0$, i.e., there exists some noise which affects the classification task.

This causes an effect similar to the Blossom Effect with the variance value being smallest at the middle layer and subsequently increases towards output layer. This is similar to the observations from the Chapter 3.

Result:

1. *The middle layer(s) are significant and constitute core knowledge.*
2. *The neural network operates with middle layers.*

4.3.3. Scenario 3: x = 100 (full noise data with complete overlapping features)

When x = 100 i.e., the data is full of noise and overlapping features, this causes the minimisation of variation to occur in all directions producing a combination of the funnel (scenario 2) and pipe (scenario 1).

Result:

1. *Importance of a particular layer may not be determined.*
2. *Neural network operation is indeterministic.*

4.3.4. Evaluations:

The evaluation of Hypothesis 1 is performed using three datasets: IRIS, MNIST handwritten character recognition and Speaker dataset with MFCC coefficients.

Experiment Set 1:

The first set of experiments are performed with modified IRIS and MNIST datasets with three independent and non-overlapping features for Scenario 1: x = 0 (no noise, clean data).

Experiment Set 2:

The second set of experiments are carried out using an Air pollution dataset [224]and the MNIST dataset with two different setups, one with the existing dataset where there is some overlap and a modified setup to make sure there is a profound overlap to suit Scenario 2: x > 0 (some noise with a few overlapping features).

Experiment Set 3:

To evaluate Scenario 3: x = 100 (fully noise data with complete overlapping features) it is necessary to perform the experiments on a special type of dataset. Therefore, a modified MNIST, Synthetic image dataset and speaker datasets are used.

4.4. Hypothesis 2 (H2): The Blossom Effect

For an input with x features, when C_m components are extracted from the middle layer L_m of a neural network with l layers such that $C_1 < C_m > C_l$ where $C_1, C_2 \ldots C_l$ ae components extracted from layers 1…..l , The transformation of information into knowledge occurs at the middle layer where the variance is minimum, and number of components is maximum.

4.4.1. Evaluations:

Hypothesis 2 is evaluated using Synthetic Hierarchical Dataset, CIFAR, TIMIT and MNIST datasets. The experiments will be carried out in two steps:

Step 1:

Extract components from various layers and compared them according to the layer number.

Step 2:

Perform a comparison of variance of the weights with the variance calculated from the component features.

Testing on Random Dataset:

The Blossom Effect hypothesis is tested on a synthetic dataset with unknown values/random values without knowing the characteristic of the data (class) upfront. Since the values are generated randomly, the amount of noise and its category (low, medium or high based on percentage) cannot be determined (scenario 1, 2 or 3).

4.5. Research Approaches

There are two types of research approaches detailed in the literature: Inductive and Deductive. The process of selecting a research approach is highly dependent on the purpose of study and availability of resources and, in some cases, time frame [225]. In some cases, hybrid approaches based on selective aspects of both inductive and deductive methods are proposed and have been successful [226]. Before choosing a research method, it is necessary to understand key aspects of these methods.

4.5.1. Deductive

The initial investigation will enable the formation of a new theory. This new theory may be entirely new or an extension of or deviation from an existing theory. In some instances, it could be simply be a hypothesis generated based on a review of the literature or results in hand. In this case, the research design and experiments are framed to validate the theory and test the proposed hypothesis.

4.5.2. Inductive

Traditionally as well as in majority of instances, research is carried out by proposing a hypothesis and then providing sufficient experimental verifications to prove or disprove the hypothesis. While performing these verification experiments, researchers often come across some new theoretical principles of generalisations which were not proposed in the original hypothesis. In this case, the theoretical principles have been extracted based on experimental results. This type of approach is called an inductive approach.

4.5.3. Selection of Research Method

Though the research approach adopted in this thesis can be categorised primarily as deductive, the possibility of inductive hypothesis generation cannot be ruled out. In this research, the hypothesis was proposed built on an initial investigation (presented in Chapter 3) based on an inquiry into the published literature. The research design and experiments work towards validating the hypothesis. Since this research has both hypotheses and experimental verification, it falls into the category of a *deductive* approach. However, the hypothesis is based on the results of preliminary investigation which makes it *inductive*.

Another strong reason to support the inductive nature of the research is the lack of sufficient literature on how Deep Learning works (Chapter 2). Deep Learning is the core principle of this thesis (see Chapter 1's Research Focus, Thesis Contribution and Chapter 2's Research Gap). The *deductive* and *inductive* nature of this research means that a pragmatic combination of deductive and inductive approaches has been adopted as the research methodology for this thesis as presented in the next section.

4.6. Deductive-Inductive Research Approach (DIRA)

A combination of deductive and inductive methods is adopted for the research, and it is named as Deductive-Inductive Research Approach (DIRA). The DIRA approach consists of the following steps as presented in **Figure 4-1**.

As shown in the **Figure 4-1**, the hypotheses are based on the experiment results obtained from the preliminary investigation. However, before proposing the hypotheses, a detailed literature review is conducted to reaffirm the research gap identified which is the deductive part of the proposed research approach

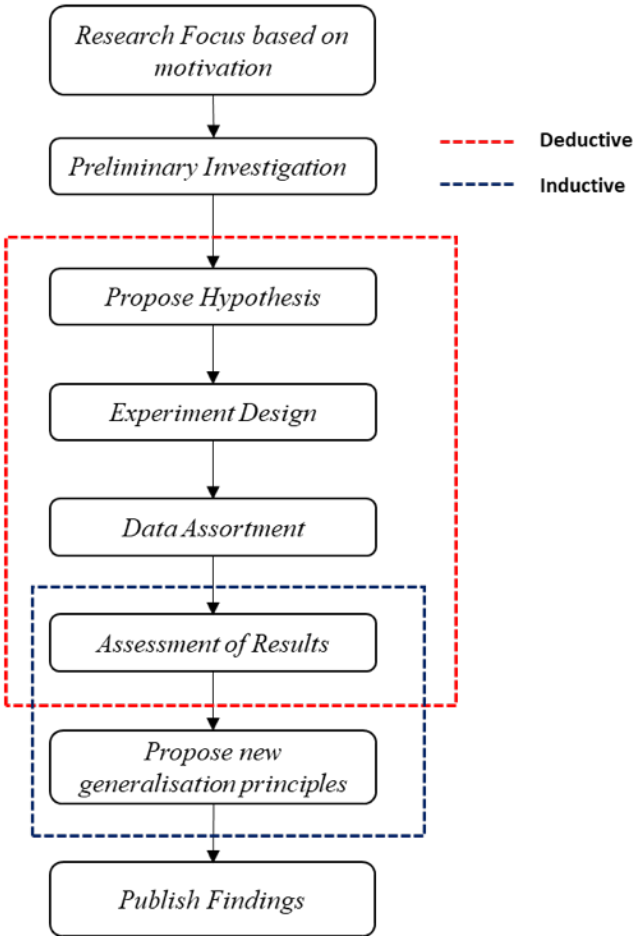


Figure 4-1: Structure of the Deductive Inductive research approach (DIRA) used for this research. .

The validation of the proposed hypotheses is carried out using various experiments. The experiment results are evaluated to provide the verification of the hypotheses. Further, the experiment results have provided some new principles which make this research inductive.

4.7. Chapter Summary

This chapter presents the main hypothesis with two sub-hypotheses designed towards achieving the research aim. The outline of experiments for evaluating the proposed hypotheses was presented along with the research method to be used. The next chapter, Chapter 5, presents a new component model for transfer of knowledge. This model serves as the framework for extracting and transferring knowledge components. The performance of proposed transferable knowledge component model is also demonstrated in the next chapter.

Chapter 5 Transferable Knowledge Component Model

5.1. Introduction

The hypotheses presented in the previous chapter can be tested through a practical application of transferable knowledge component model. In this chapter, a new knowledge component model for transfer of knowledge is proposed which functions as the framework for extracting and transferring knowledge in a neural network based deep architectures. The efficiency of proposed transferable knowledge component model is also demonstrated through the evaluation of results obtained from the initial experiments.

Firstly, the theoretical aspects of a transferable knowledge component model designed to establish a relationship between input features and DNN weights. This is achieved by proposing a component model constituted using DNN weights.

The second part of this chapter presents an evaluation and discussion of how the hypothesis (H2-3) presented in Chapter 4 is related to the component model. This is followed by a discussion of component extraction approaches and initial experiments using these approaches that evaluate the proposed component model.

5.2. The Component Model

The input data that is fed into a DNN consists of features (input features) that are combination of one or more attributes present in the data. A DNN consists of layers into which these input features are transmitted in the form of weights. The transformation of input features through the neural network layers is significant. It is widely accepted that it is the features that a DNN is learning and is the key success factor of deep learning. The DNN layers consist of numeric weights, but the form of the features that the DNN learns from these weights is not proven. It is the conjecture presented in this thesis that the knowledge is represented in the form of patterns by a set of weights grouped together and DNN is acquiring this knowledge through training. The weights in the DNN layers are optimised for better learning which gives better knowledge to the DNN and makes it more efficient in problem solving. It is noteworthy to observe that there are some weights that may not be necessarily contributing to the problem solving yet remain in the layers. The insight into the principle of extracting significant weights called knowledge components will provide a new direction in the research of transfer learning.

Consider an input feature f_i which transforms through the DNN layer l where $l = 1 \ldots L$ (number of layers). The component C_{li} is the i^{th} component in the l^{th} layer can be defined as the linear combination of features extracted from the correlated weights (weights grouped together with underlying patterns) w with an error ε. The non-correlated weights have no role and are mere numbers with minimal or no influence. However, the determining factor in the transformation is unknown and has not been the subject of research at the time of writing of this thesis.

The basis for DNN learning is how features are translated from the input (raw) and then learnt as high-level representations through which the labels associated with the data can be identified. Since the core components in the DNN layers are weights, the features exist in the layers in the form of weights. The DNN scans through these weights and identifies and learns the underlying features which is knowledge attained by DNN.

From the literature, it is clearly evident that the layers near the input have raw and low-level features that are nearly the same as the discrete form of input. Further, the layers near the output are high-level features that are responsible for the classification or identification of particular aspect of the feature. From the learning mechanism of DNNs (representation learning) the low-

level features are transformed to form high-level features across multiple transformations as they pass through various layers of DNN.

Consider an input data with *n number of* features, *F* defined as,

$$F = f_1, f_2, \ldots\ldots f_n \qquad (7)$$

where each feature is a linear combination of attributes (ref. Chapter 2)

To determine the projection of these feature vectors in DNN layers:
Consider a DNN with *l* layers with W_{il} being weight of the i^{th} node in l^{th} layer with l = 1…. L.

The aspect of projection of weights for separating the features as a group of weights in a DNN layer can be attributed to the concept of Blind Signal Separator model (BSS) [227]. The DNN weights are obtained by a combination of input values and variables involved in training mechanism (bias, error and training algorithm variables).

If *W* is the weight tensor, a component *C* thus may be defined as a combination of these weights in the form of features. To define *C* in terms of *W*, it is necessary to establish a relationship between the input features and components.

The weights of the DNN are then obtained by combining input features and an undetermined number (may be bias, learning rate or other values obtained through training algorithm) as follows:

$$C = F'S + \varepsilon \qquad (8)$$

where
F' is the vector component projection of features *F* in the DNN
ε is the stochastic error
S is underlying function value or factor that affects the transformation.

F' is the component obtained through a combination of weights of DNN that matches a particular set of features. F' consists of an individual feature or combination of features of F that exists as underlying representations in weights W. The introduction of S is due to the fact that not every weight is significant in the learning process by itself. It is noteworthy that the individual weight may be significant in a group as defined in the literature [228]. However, it is important to note that the value of S becomes 0 or no value in case of optimised DNNs. Thus, *S is a generalised hidden function and determination of S is out of the scope of this research.*

The novelty of this research is in attempting to determine the underlying patterns that exists in the weights. Therefore, the challenge is to define F' in terms of W.

A simple approach to start with would be extracting (weight) components. Using PCA, F' can be obtained through the covariance matrix of W for the example feature vectors. This way the eigen values and eigen vectors of W can be found. The m eigen vectors having the largest Eigen values are then used as the columns of F'. Thus, *a component C can be extracted through the weights using PCA.*

So (from PCA),

$$F' = W\ \mu \tag{9}$$

Where
W is the weight tensor
μ is the variance

Thus,

$$C = W\ \mu\ S + \varepsilon \tag{10}$$

To obtain the a^{th} component in the l^{th} layer

$$C_{l,a} = W_a \mu_l S_l + \ \varepsilon_a \tag{11}$$

where $S_1 \cong 1$

The component thus can be extracted using PCA in the case of non-overlapping and highly separable features. For datasets with highly overlapping features with high correlation between features, Factor Analysis (FA) can be used. FA can produce the similar results to PCA by considering the feature commonalties rather than correlation. Thus, for non-overlapping features, FA can be used to define the component *C* as follows:

To obtain the a^{th} component in the l^{th} layer for highly overlapping features, a component can be determined by the correlated weight values based on the variance and the knowledge components can be extracted using FA.

Consider,

$$C = FS + \varepsilon \qquad (12)$$

The FA model is used to determine the knowledge components in the layer l with unknown features as follows:

$$C_{l,1} = F'_1 S_{l,1} + F'_2 S_{l,2} + \cdots .. + \varepsilon_l \qquad (13)$$

In line with FA, it can be generalised as follows:

$$C_{l,a} = \sum_{p} F'_n S_l + \varepsilon_l \qquad (14)$$

where

$C_{j,a}$ is the a^{th} component in l^{th} layer

p is the number of components present in the l^{th} layer

F'_n is the feature factor determined through *W*

S_l is the Sree (unknown) constant that exists in the weights in the l^{th} layer

ε_l is the error in *l*th layer (stochastic)

5.3. Hypothesis vs Component Composition in DNN Weights

In principle, the hypothesis H2 presented in the Chapter 4 Section 4.3 states that the number of components extracted from the individual layers is based on the variance between the weights thus providing a direct relationship between the weights extracted from the components and standardized PCA and FA.

Both PCA and FA are functionally dependent on variance. Number of components or factors are generated based on variance. In case of PCA, the number of correlated variables is transformed into a smaller number of uncorrelated variables. PCA is performed on a symmetric matrix which can be either pure sum of squares or correlation. So, in principle, PCA is based on total variance.

From the equation for PCA component extraction,

$$C_{j,a} = W_a \mu_1 \; S_1 + \; \varepsilon_a \qquad (15)$$

The μ_1 is the variance, that influences the composition of the components and is often considered as important parameter. Further, in PCA the number of components extracted is based on variance of the data matrix.

In case of FA, the commonalities are considered instead of correlations in order to extract underlying knowledge components.

$$C_{j,a} = \sum_{p} F'_n \, S_j + \; \varepsilon_j \qquad (16)$$

F'_n are individual factors based on common variance i.e., variance shared across the variables. This variance has direct influence on number of factors deduced from the data matrix.

Consider a weight matrix W_l for layer *l* (*l* = 1…. L) for an *L* layered DNN and for each layer C_l components.

According to the proposed hypotheses H1,
For each layer l,

C_l is minimum for max (μ_l) and
C_l is maximum for min (μ_l)
in the weights W_l

For a dataset *D*, the number of components extracted from the weights of layer *l* using PCA or FA, i.e., C_l is at its minimum when the variance between the variables is at its highest value and vice versa.

Thus, the influence of variance stated in standard PCA and FA is similar to the influence of variance mentioned in hypothesis H2.

The number of components extracted is determined by variance and can be controlled by introducing variance-based stopping criteria.

5.4. Component Transfer Model

Consider a trained DNN called DNN_t from which the components are extracted and transferred to a new destination DNN_D. The components C_t extracted from l^{th} DNN_t is obtained from equation 11 as

$$C_{tl} = W_{tl}\mu_l S_l + \varepsilon_l \qquad (17)$$

The C_{Sl} is the transferable component(s) that needs to be defined in terms of weights Wt_l. To transfer the component, the destination Weights needs to be replaced with the values obtained through C_{tl}. This transfer of weights is possible through a weight update using error propagation as proposed in BP mechanism. This is a form of learned feature transfer.

The objective now is to define the destination weights W_D in terms of Wt_l. The initial values of weights including that of W_D are initially random numbers to start with followed by weight updates, for instance, through BP.

$$W_{Dl} = w^1, w^2, w^3 \ldots . w^n \quad (18)$$

The transfer of components involve the transfer of feature weights to l^{th} layer of the destination $DNN_{D.}$

To start with, the number of hidden nodes must be equal: hence,

$$W_{Dl} = \left(C_{tl} - \mu_{\mathrm{Dl}}\right) + S \quad (19)$$

where the values S and C_{tl} are transferable weights and μ_{Dl} is the mean variance obtained from the existing weight values in the destination.

To obtain individual weights values

$$W_{Dl}^{i} = C_{tl}^{i} - \mu_{\mathrm{Dl}} + S \quad (20)$$

where i is the i^{th} weight, S and C_{tl}^{i} are obtained from trained weights.
The value of μ_{Dl} is based on number of weights (hidden nodes in the next layer) to be replaced. When the source and destination weights are same i.e., replacing all the weights (no initialisation) μ_{Dl} will be 0. If only a set of weights are to be replaced in the destination, the value of μ_{Dl} depends on variance calculated on the weights of the destination. The next section evaluates equation 20 through transfer learning experiments. The scenario used for the experiments is based on the assumption of having the same number of weights in the source and destination.

5.5. Component Extraction Experiments

A 7-layer deep neural network is used for this experiment with 50 hidden nodes in each layer. The configuration and parameters of this DNN are similar to what was used for the initial experiments detailed in Chapter 3. In order to extract components, weights are extracted from

the middle layers i.e., layer 3, 4, and 5 since these layers are already proven to be significant (Chapter 3).

To help determine the association of weights and relationship with features, firstly, a multivariate wine dataset [229] with 13 attributes and 178 samples is used for the experiments. Proposed component model is used to extract the components according to equation 16 presented in the previous section.

Figure 5-1 illustrates the number of components extracted from layer three, four and five, and it is possible to determine that layer three has one component more than the middle layer (layer four).

Both layer three and four consisted of the condensed representation of the original 13 features. From the middle layer the two components are expanded in the next layer (layer five) to provide 12 components. It is likely that this expansion is what makes the next layer features more problem specific. This change in the number of components over layers 3 and layer 4 seem to represent the Blossom Effect and provides support for the hypothesis H2. This is also in line with proposed effect of weight variance which states that the middle layer has minimum variance that influence the representation of features (knowledge) as presented in the Chapter 3. Furthermore, equation 16 of variance calculation is also in line with the weight variance for layer three, four and five as presented in the Chapter 3 **Figure 3-19.** The number of components is based on variance-based stopping criteria similar to other statistical-based component extraction approaches as mentioned in Chapter 4.

The features present in layer four, the middle layer, are overlapped and/or condensed based on the input representations that exist in the attributes of dataset. From the middle layer (layer four) to layer five, it appears that the components start unfolding and become more problem-specific features. This unfolding exposes the high-level features that are folded-in at the middle layer. This effect of fold-in and fold-out is proposed as ‘The Blossom Effect’. Thus, this experiment provides the evidence to the existence of ‘The Blossom Effect’.

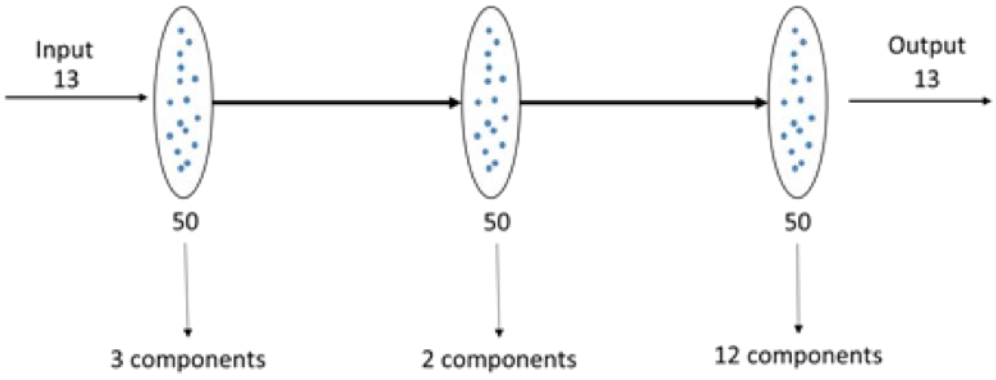


Figure 5-1: Extraction of components from the weights: The number of components is reduced towards middle layer and then increases their after towards the last layer. This is due to the weights being together (condensed representations) as presented in Chapter 3 Section 3.8

The experiments conducted to support conclusions using inductive research approach must be extensive, rigorous and comparable and to be performed on variety of datasets to support the validity of conclusions and observations. Therefore, the hypothesis H2 is further investigated extending this initial experiment using diversified datasets, deep architectures and different technologies (hardware and software) in Chapter 6.

The next section presents the extraction of components using autoencoders and WofW followed by a comparison with the proposed model.

5.6. Evaluation using Autoencoders

Since PCA and FA are threshold based, it is not feasible to force the number of components to be extracted without losing potential knowledge. So, PCA or FA cannot be used for comparing the proposed approach.

The main aim of this research is to extract feature components from the neural network weights so that they can be analysed against the input features-based components extracted using proposed knowledge component model. As mentioned earlier in this thesis, autoencoders are traditionally used for feature reconstruction which makes them idle choice for evaluating proposed component model.

With deep autoencoders, the number of weights (nodes) can be set to a constant value so that the weights in the layer represent the condensed features (components). So, the experiment with autoencoder is conducted to identify the possibility of reducing nodes in the middle layer.

A deep autoencoder network learns the representations in the features by reconstructing the input. Once the input is encoded, the middle layer represents the features in the form of weights that are fine-tuned to reconstruct the input. However, we can restrict the number of nodes in the middle layer to a set number so that all the features are condensed to specific number of weights as shown in **Figure 5-2.**

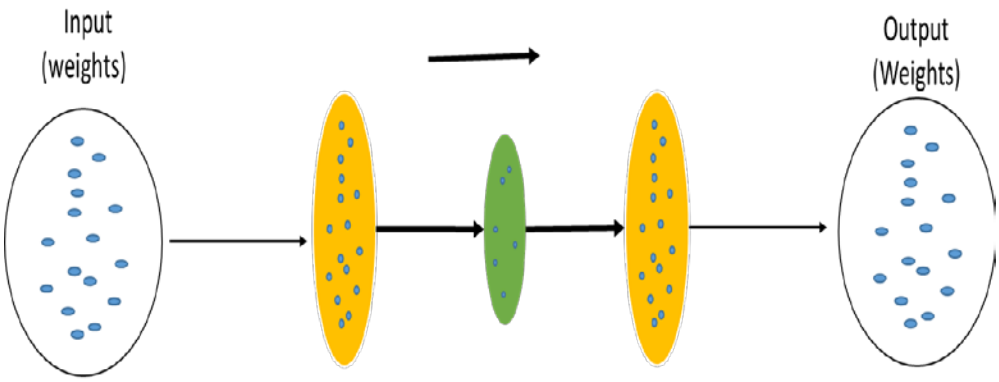


Figure 5-2: Deep Autoencoder with encoding, decoding layers represented in amber and middle layer represented in green.

In this experiment, the weights achieved from three layers are given as input to an autoencoder which encodes them into ten nodes that represent the entire essence of 50 weights. Then, these ten weights are used with a decoding mechanism to reconstruct the input. The decoded weight values are obtained by transposing of the encoded weight values. The error between input and reconstructed input is propagated, and weight values in the hidden layer are adjusted to minimise the error. This experiment is represented in **Figure 5-3**.

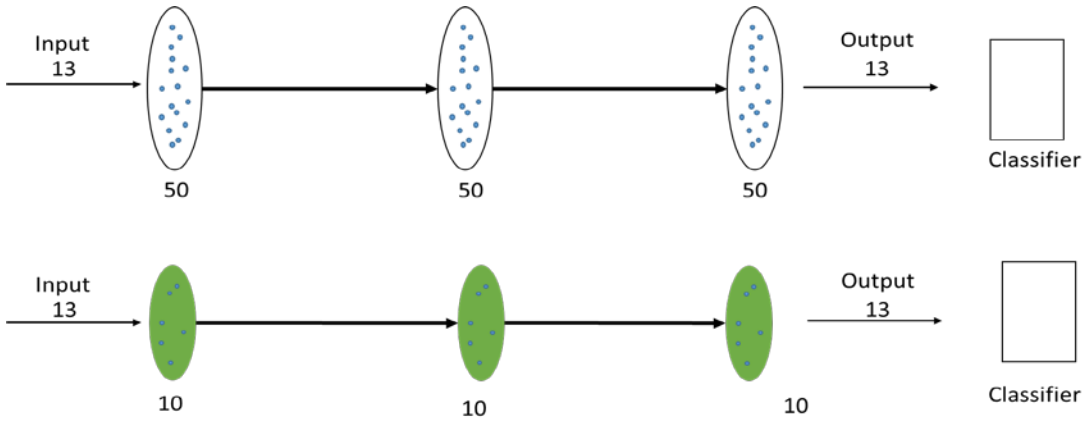


Figure 5-3: The strategy of extracting components from deep autoencoder network

This experiment is repeated for all three layers extracting ten components each layer and the results obtained with various combinations of middle layer weights are presented in **Table 5-1.**

Table 5-1: Experimental Results: classification experiments carried out on a Deep autoencoder network for four different scenarios.

Scenario	Nodes	Classification Accuracy (%)
Random weights	50,50,50	81.77
Random weights	10,10,10	27.55
Extracted Weights of the weights (WofW), all layers	10,10,10	75.65
Extracted WofW for middle layer, random weights for the other two layers	10,10,10	58.25

In the first two scenarios, random weights are used for the experiments, and an accuracy of 81.77% is achieved with 50 hidden nodes but only 27.55% with ten hidden nodes. In the next set of experiments, the weights of the weights (WofW), extracted using autoencoder are used. With WofW replacing the random weights in all the three layers, the classification accuracy is improved by around 50% reaching a value of 75.65% when compared to the experiment using random weights. However, when the WofW are used only for the middle layer, the classification accuracy is reduced to 58.25% which is 17.4% less than the value when WofW weights are used for all the three layers.

When the proposed model is applied to extract knowledge components, the number of weights extracted that are present in the knowledge components are obviously not the same as WofW, since DAE can be dictated to extract required number of components whereas the proposed model cannot. To overcome this, the number of components is ignored, and the layers are updated with the components attained through proposed model and the classification accuracies are presented in **Table 5-2**.

Table 5-2: Classification accuracies: Comparison of deep autoencoder, WofW and proposed Knowledge Component Model

Scenario	Nodes	Accuracy (%)
Random weights	50,50,50	81.77
Weights of the weights	10,10,10	75.65
Knowledge Component Model	10,10,10	77.93
Knowledge Component Model Full components On transfer learning	28,30,30	**81.28**

The classification accuracies achieved for random weights, WofW and knowledge component model are 81.77%, 75.65% and 77.93% respectively. The experiments results show that the proposed model outperformed WofW by 2%. It is interesting to see that the classification accuracy achieved using the proposed knowledge component model is in between the accuracy values of WofW and the best accuracy (with 50 nodes rather than 10 nodes) random weights model. The small difference in the classification accuracy may be due to the fact that the number of nodes is forced to 10. When the knowledge component is transferred without forcing them to be 10 nodes, the classification accuracy is 81.28%, reaching almost the best accuracy when transferred to an untrained DAE.

Apart from experiments stated above, there are two alternative scenarios that can be used for implementation. Firstly, replacing the weights of each layer of the DAE with the components extracted using the knowledge component model presented in Section 5.1. Secondly, WofW in the middle layer should be replaced with the knowledge components.

5.7. Chapter Summary

This chapter presents the mathematical expression for the proposed Transferable Knowledge Component model using standard statistical approaches for feature extraction and dimensionality reduction. The existence of the Blossom Effect is also demonstrated through the experiment results of knowledge component model. The comparative analysis between the various experiment results show that the proposed model is able to retain the efficiency of the DNN for transfer learning experiments.

The next chapter consists of experiment results and reconciliation of the proposed component model on number of diversified datasets of multiple domains. Further, the experiment results are assessed and analysed to provide a conclusive evidence for the existence of the Blossom Effect in DNNs.

Chapter 6 Experiment Results and Evaluations

The previous chapter presented the Transferable Knowledge Component model and the results obtained from the initial experiment conducted to test the proposed component model. This chapter presents the evaluation of the proposed model using a variety of datasets from multiple domains.

This chapter is divided into three parts, and each part has an overview section to provide a brief explanation on the contents of that section. The first part presents the details of various datasets used for the experiments along with the results of the classification experiments performed on them. The second part presents the extensive evaluation of proposed Transferable Knowledge Component model followed by the third section with the analysis and discussion where the proposed hypothesis is validated.

PART I: Datasets & Technical specifications

Overview of the section

This section provides the details of various datasets used for the experiments and the technical specifications pertaining to both software and hardware. This section also provides the technical details of experiment setup and various parameters and their values. Firstly, the details of datasets like type of domain, attributes, number of samples, resources from the literature and other particulars are presented. The results from the classification experiments are also presented in this section. The transferable model is extracted from the deep architectures that have attained the highest possible classification accuracy on their respective datasets.

6.1. Hardware and Software specifications

The hardware and software specifications are very important for mitigating technology specific bias if any, and to evaluate the consistency in the experiment results. For any experimental evaluation, it is necessary to mention the hardware and software specifications to provide clarity on execution time for ML framework / libraries used for the experiments. The experiments reported in this thesis are carried out on a variety of hardware from a simple laptop (Microsoft Surface) to powerful GPU based systems. The details of hardware configurations are presented in Appendix I.

The experiments are carried out using Weka, MATLAB, TensorFlow, Keras libraries for Python, and Microsoft C#. Open source code (Matlab) is used for some feature extraction experiments. Open source Matlab libraries for t-Distributed Stochastic Embedding is used for visualisations. The detailed software specifications are presented in Appendix I.

In principle, there are three types of deep architectures that are used. Other than the number of layers, the parameter values for majority of the ANN and deep architectures are the same. The technical details of various deep architectures used for the experiments are presented in Appendix I.

6.2. Datasets

The experiment setup is one of the most important parts of both inductive and deductive research. Choosing the correct type of evaluation method and experiment design is a crucial and critical aspect of validating the hypothesis. This chapter provides the specifications for the various types of deep neural networks used, their parameters and other technical details.

To enable variety, veracity, volume, correlation, overlapping and dependency, the datasets are chosen from different domains and applications. The details of the datasets used in the experiments are presented in **Table 6-1**. There are several customised datasets derived from these standard datasets and their details are presented in later sections. The datasets are used for classification experiments on various deep architectures that are fine-tuned to achieve the highest possible classification accuracy.

Table 6-1: List of datasets used for the experiments: The properties of various datasets used for the experiments categorised based on the domain of application.

Dataset Category	Dataset Name	Category	No. samples	Attributes	Classes	Comments
Generic benchmark datasets for testing	IRIS	Flowering plant	150	4	3	Modified versions of IRIS (3) datasets are also used
	Wine	Wine classification	178	13	3	
Image	MNIST	Character recognition	60,000	785	10	7 variants of MNIST are used later
	ImageNet	Image recognition	60,000	20	20	A subset of the original dataset is used
	CIFAR-10	Image Recognition	60,000	785	10	
NLP (speaker)	TIMIT	Speaker / Speech	6300	39	630	
	AN4	Speaker	1154	16	84	Total samples available 94816

						MFCC Features
Air Pollution	CASTNET	Air Pollution spatiotemporal	8700	13	2	Total samples 9358
High dimensional	Prostate Cancer	Gene expression	102	12600	2	variant of dataset with random values is also used
Hierarchical	Synthetic	Synthetic Hierarchical	90	16	2	Distributed representations

6.3. IRIS

6.3.1. Datasets

The IRIS dataset is one of the widely known datasets in pattern recognition particularly for classification [219]. The dataset consists of four attributes with 150 instances classified into three classes of iris plant. One class is linearly separable from the other two which are not linearly separable from each other. This creates an associative relationship that influences the classification. **Figure 6-1** presents a pictorial representation of the class clusters for the IRIS dataset. The feature distribution across the classes for each attribute is presented in **Figure 6-2**. For the class iris-setosa, petal length and petal width are clearly separable and can be considered as the most influential and determining factor variants. Further, it can be observed that two of the four features are correlated with the iris class value which makes the results highly predictable.

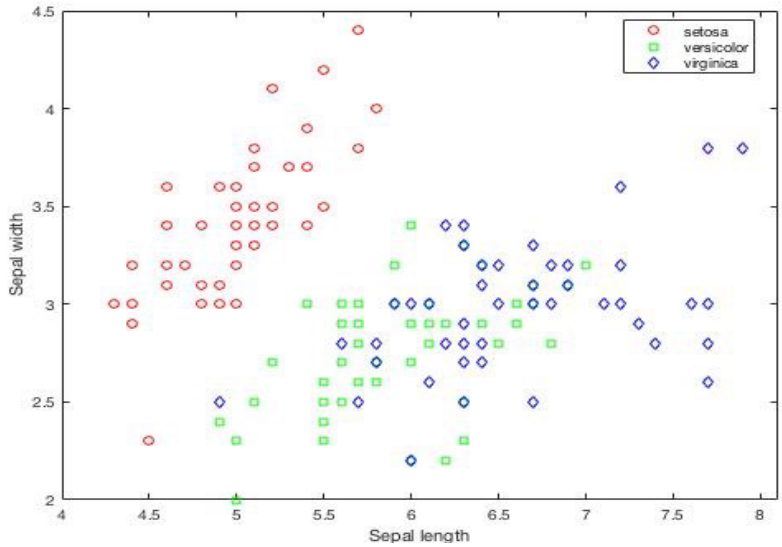


Figure 6-1: Plot of IRIS dataset indicating the classes clustered across the 2D feature space.

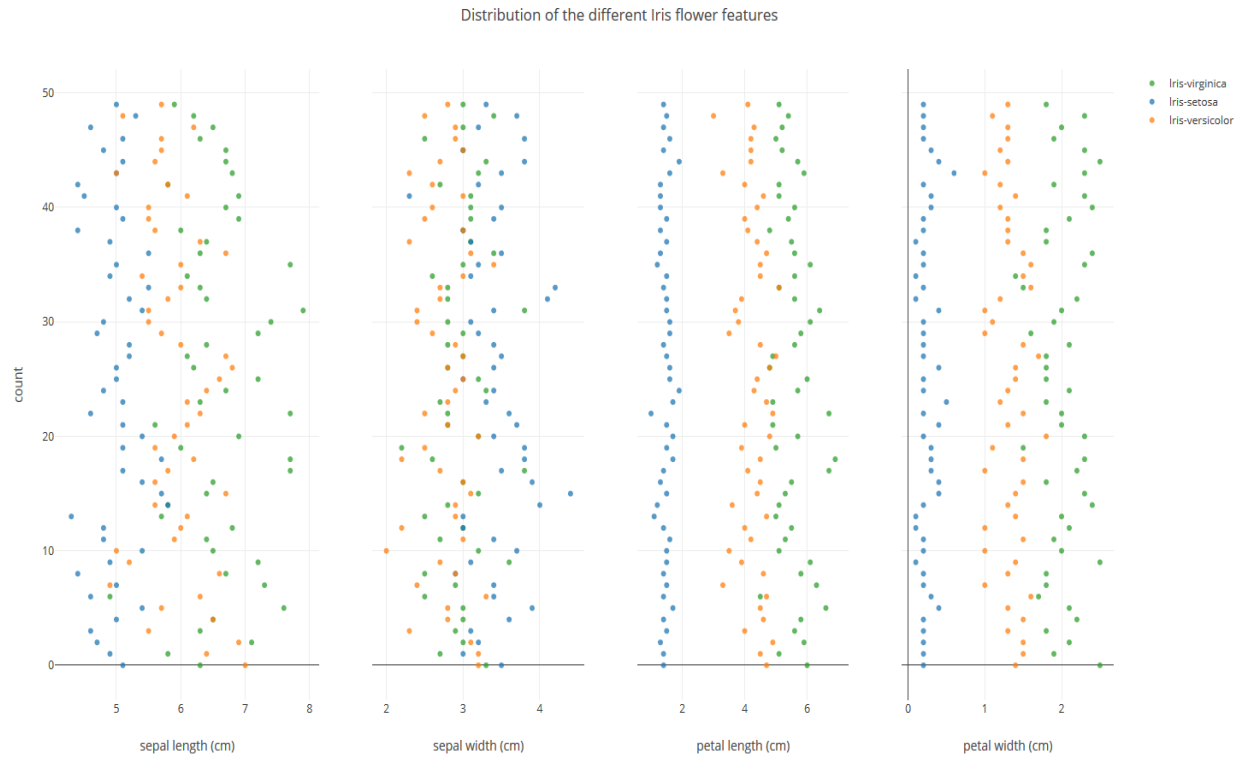


Figure 6-2: Plot for the IRIS dataset representing the class distribution for its four attributes. This figure illustrates that classes are clustered within an attribute. For some attributes such as sepal length and sepal width, all the classes are closely associated whereas for petal length and petal width, one class is clearly separated (Iris-verginica). The count of Y-axis represents the index of the sample.

M-IRIS is a modified version of the IRIS dataset, developed specifically for this research, where attributes are tweaked so that more overlapping is exhibited when compared to regular IRIS dataset as shown in the **Figure 6-3.** The highest possible accuracy that can be achieved for IRIS is 99% since one sample consists of ambiguous values that are not specifically associated to a single class. These values are tweaked to create M-IRIS with which a clear accuracy of 100% can be achieved. Further, there are two more versions of this modified dataset where one of the three attributes are replaced by an independent or associative attribute based on type of experiment.

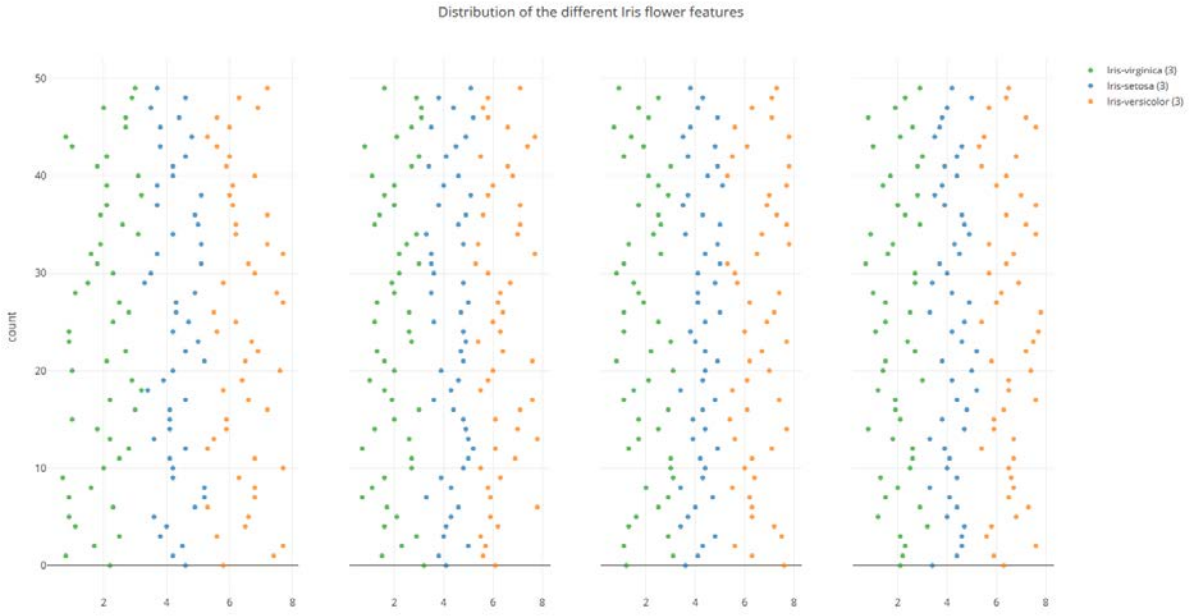


Figure 6-3: Feature/attribute values distribution for Modified IRIS dataset (M-IRIS) dataset. The plot indicates the classes being overlapped for all 4 attributes. The count represents the index of the sample.

The modified IRIS datasets can be used to examine the impact of modifying input values on neural network weights. The tweaking is also performed by adding or deleting attributes in the input dataset that represent features.

M-IRIS1: The values of attributes are tweaked to achieve 100% accuracy.
M-IRIS2: An attribute is added which is correlated to all the original four attributes.
M-IRIS3: An independent attribute is added which has no correlation with original four attributes.

The feature distribution of the M-IRIS1 dataset is presented with a clear association of values in attributes three & four when compared to regular unmodified IRIS dataset. The feature maps of MIRIS2 & MIRIS3 are presented later (Sections 6.14.2) along with the feature extraction.

6.3.2. Classification Experiments

Classification experiments were carried out using ANNs with 3, 5, 9 and 13 hidden layers using gradient descent layer-wise training with BP for overall fine-tuning. The experiments are carried out using MATLAB 2017a and Weka and the technical details of the experiments carried out on IRIS and its variants are presented in Appendix A.1. The experiment results are presented in **Table 6-2** and a graphical representation in **Figure 6-4.**

Table 6-2: Classification Accuracies for IRIS and modified IRIS datasets for four different topologies

No. of Layers	Classification Accuracy			
	IRIS	**M-IRIS1**	**M-IRIS2**	**M-IRIS3**
3	78.3	98.6	58.4	28.9
5	79	99.8	43.7	43.8
9	84.5	88.3	45.5	45.3
13	27.1	41.8	16.4	11.2

The classification results indicate that increasing the number of layers may not always produce efficient results. With more layers, the DNN learning mechanism may indulge in deeper segregation of features at discreate levels which will end up with more similarities at the discrete level resulting in confusion while matching classes. This is the case where the 13-layered DNN produced the least accuracy which is an observation that lead to removing this topology for feature extraction. M-IRIS1 is able to achieve over 99% accuracy. Therefore, can be used to compare representation in other IRIS datasets. Detailed results for these experiments are tabulated in Appendix A.2.

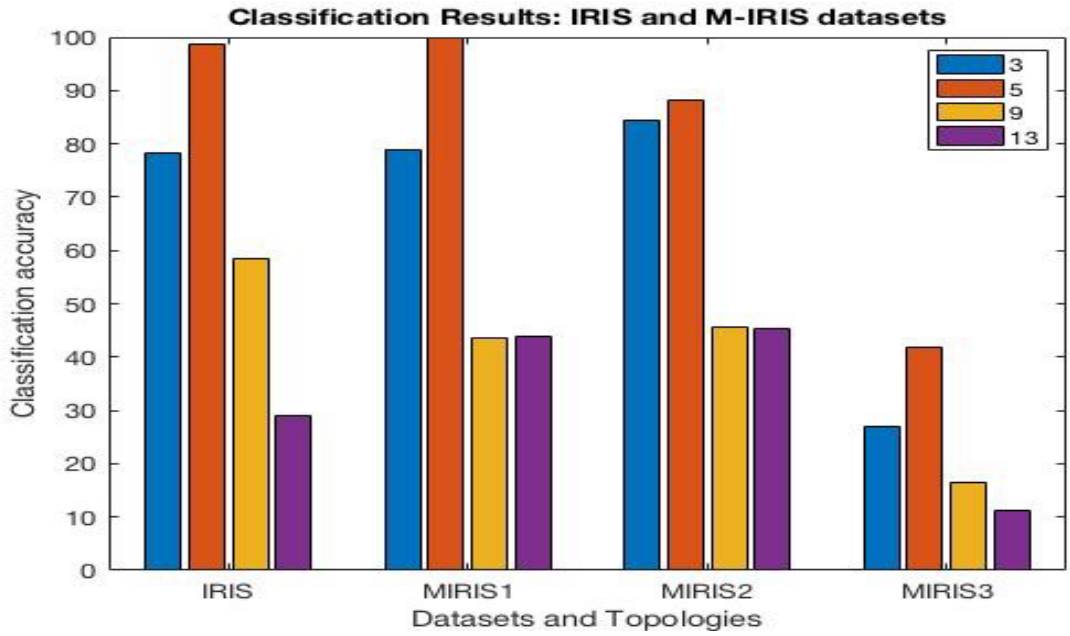


Figure 6-4: Classification results for IRIS and M-IRIS datasets with three, five, nine and 13-layered deep neural networks

6.4. Wine Dataset

The wine dataset is a collection of data obtained from the chemical analysis of wines from different cultivars [230]. The wine dataset consists of 178 instances and 13 attributes with real and integer values and contains three classes as summarised in **Table 6-3**. The attributes contribute to the overall chemical composition. The wine dataset is considered as a well-structured and poised dataset with high consistency. However, this dataset is not challenging because of its controlled behaviour.

Table 6-3: Details of Wine Dataset

Data Set	Multivariate	**Number of Instances:**	178
Attribute	Integer, Real	**Number of Attributes:**	13
Associated Tasks:	Classification	**Missing Values?**	No

The three classes can be segregated using simple clustering as shown in **Figure 6-5**.

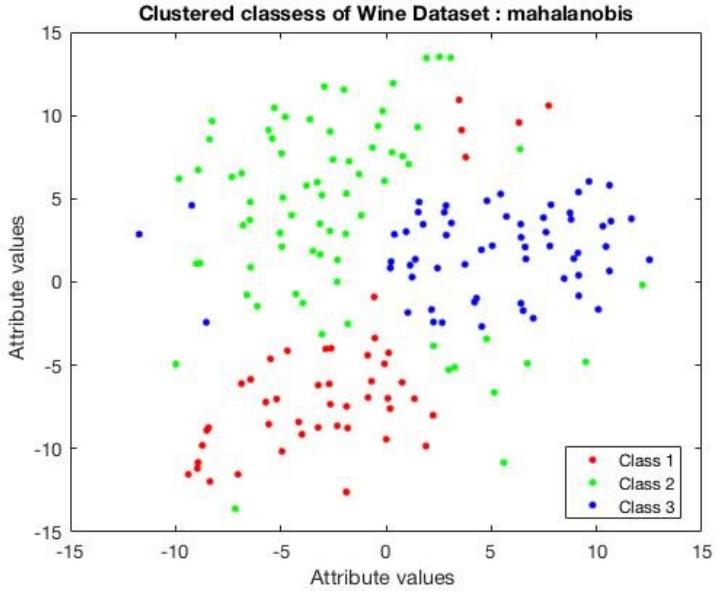


Figure 6-5: The plot for cluster analysis on wine dataset: The 2D scatter shows the distribution of the 3 classes of wine dataset samples.

The clustering is based on Mahalanobis distance. There are discrepancies in some classes, for example, five instances of class 1 (Red wine) values are located quite far from the cluster. However, ANNs are able to learn these minor issues and are able to classify the examples due to the strong relationship between the attributes.

The classification experiments are carried out using Weka and MATLAB MLP (NN) toolbox based custom code. A one-layered ANN is enough to achieve 100% accuracy for the wine dataset due to its simplicity. However, two more topologies are used for the experiments to investigate the influence of size and other topological factors on the internal structure of the features. The one-layered and 3-layered based experiments are carried out using Weka and the 9-layered DNN experiments using MATLAB and the results are presented in **Figure 6-6**. The 9-layered DNN has given the worst classification accuracy, while the one-layered ANN gave the best accuracy.

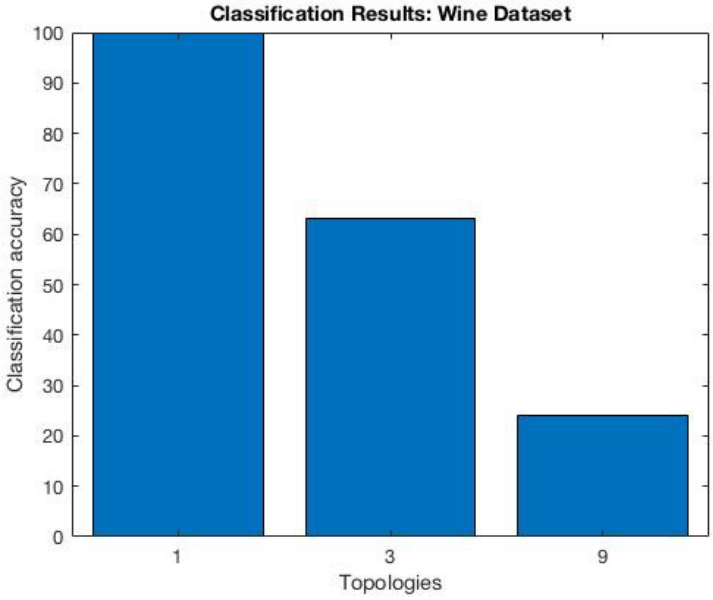


Figure 6-6: Experiment results from the classification experiments for Wine dataset with one, three and 9-layered DNNs.

6.5. MNIST

6.5.1. Dataset

The MNIST dataset is very critical to this research and has been one of the important datasets used at the initial stages of the research. The variety and veracity of the MNIST dataset makes it ideal for testing new hypothesis particularly on feature extraction.

The database is a large image dataset, the most acclaimed dataset among ML datasets for training and testing image processing systems. MNIST consists of images of handwritten digits (0-9) predominantly used for character recognition and widely accepted as benchmark dataset in the ML community (**Figure 6-7**). The dataset is owned by The National Institute of Standards and Technology (NIST), and the total size of the dataset is unknown. However, the training and testing dataset that is used for benchmark tests is 60,000 samples for training and 10,000 samples for testing and is a subset of the NIST dataset. This original black and white dataset is standardized to 28 x 28 pixels and subjected to grey scale levelling to reduce distortion (anti-aliased). The structure and various technical details of the MNIST dataset are presented in Appendix C.1.

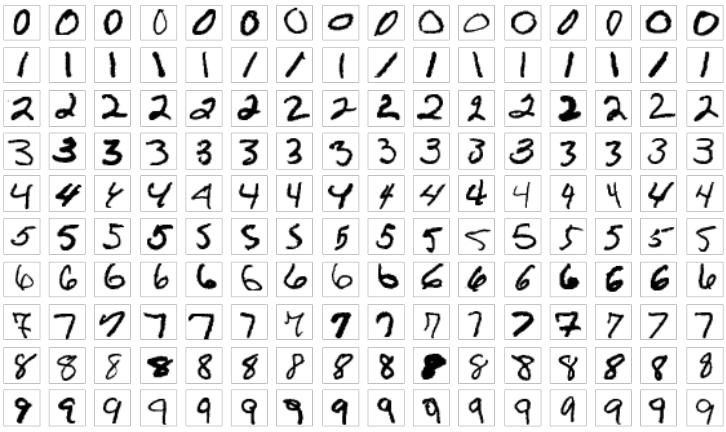


Figure 6-7: Sample data of handwritten character recognition data (MNIST) dataset

MNIST was first used in 1998 by Lecun with SVMs as a classifier that achieved high accuracy with an error of 0.8% [24]. In 2012, an error rate of 0.23% was achieved with CNNs [231] whereas in 2016 an error rate of 0.21% was achieved using an ensemble of five CNNs [232].

An extended version of MNIST called EMINST was published in 2017 which consists of 240,000 training and 40,000 testing images with the same characteristic features as that of MNIST [233]. The visual diagram of the original MNIST digits is presented in **Figure 6-7.**

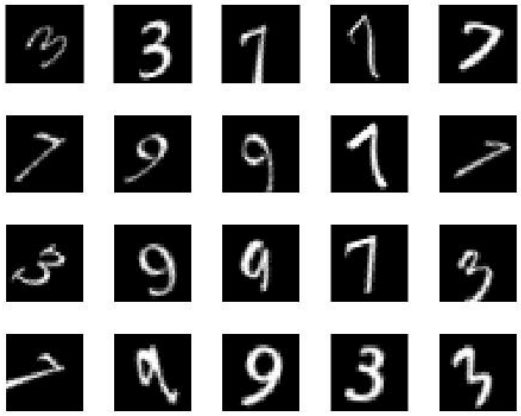


Figure 6-8:MNIST Training samples remonstrated from the weights

From the training samples, a set of digits are extracted to provide a view of what exactly the DNN is trained on. This reconstruction of training samples is presented in **Figure 6-8.** It can be observed that the training data are quite good and are often able to achieve over 95% of accuracy for majority of classifiers. **Figure 6-9** shows the samples extracted from testing data. The majority of the samples are easily recognisable and are identified correctly by the classifier. It can be observed that some of the samples are not complete or stopped abruptly. However, this has no effect on classifiers, particularly the DNNs which can identify the characters even though the data is incomplete or distorted. Furthermore, some of the digits have uneven rotations and angles which might create issues for some classifiers.

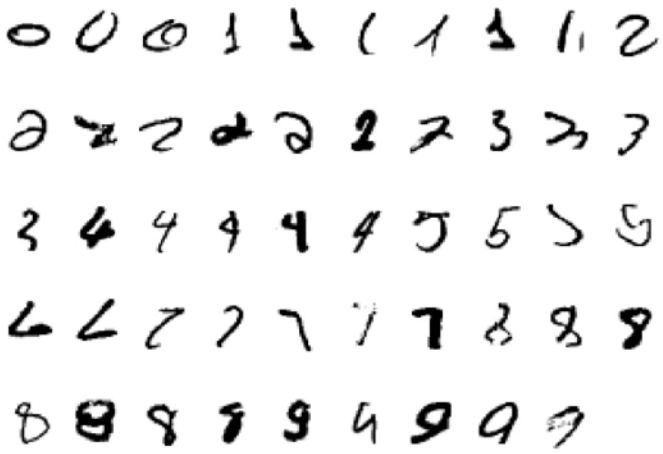


Figure 6-9: MNIST samples reconstructed from the weights (testing samples)

Modified versions of MNIST were created by applying different types of changes to MNIST dataset. For instance, M-MNIST1 is created by removing all digits except '1' and '7'. The modified versions are designed to test the impact of changes in the input features on the pattern of hidden weights. The variants of MNIST and their details are presented below:

- M-MNIST1: Digits 1 and 7
- M-MNIST2: Digits 6 and 9
- M-MNIST3: Digits 0 and 8
- M-MNIST4: Digits 1, 7 modified by greying out the top bar of digit 7
- M-MNIST5: Digits 1 and 9 (for similarities)
- M-MNIST6: Digits 0,2,7,4
- M-MNIST7: Digits 0,8,6,9

6.5.2. Classification Experiments

The experiments were carried out using three types of deep architectures.

DNN: ANN with multiple layers trained using greedy layer-wise training.
DBN: RBM based deep belief network proposed by Hinton [27].
DAE: Multiple autoencoders stacked together.

There are multiple topologies used based on the type of DNN. For DNN and DBN, the initial number of layers for the experiments are set at 7 layers followed by 13, 17, and 33 layers. Since the arrangement of layers is different for DAE networks, initially the experiments are carried out by stacking three autoencoders followed by five and nine.

The topologies and number of neurons are fairly decided based on prior literature [234] in which high accuracy results is achieved on MNIST with DNNs, DBNs and DAEs.

The number of neurons used for various topologies is in the pattern of 10-785-1024-2048...2048-1024-785, followed by the output layers. The intermediate layers are fixed at 2,048 neurons. This is followed across all the experiments. The results from the classification experiments are presented in **Table 6-4.** The detailed results with training and testing errors and other details are presented in Appendix C.

Table 6-4: Classification accuracies for MNIST and modified MNIST datasets for all architectures and topologies

Type of DNN	No. of Layers	Classification Accuracies (%)							
		MNIST	Modified MNIST						
			1	2	3	4	5	6	7
DNN	7	99.27	99.53	99.70	99.40	98.41	98.24	98.29	98.25
	13	97.87	98.83	96.01	96.88	97.27	97.25	98.85	96.27
	17	73.02	72.68	76.14	81.74	74.56	78.99	76.04	74.49
	33	58.46	64.74	57.09	62.17	60.04	60.46	63.04	59.57
DBN	7	99.72	98.52	99.02	98.14	98.06	98.99	99.31	99.51
	13	97.65	97.36	98.72	98.27	98.76	97.79	98.24	98.92
	17	76.70	82.60	73.00	80.33	86.15	84.83	72.38	73.67
	33	58.50	61.72	54.00	57.17	53.44	64.56	66.21	55.31
DAE	3	97.96	97.30	96.80	95.33	97.40	97.72	97.90	95.24
	5	99.43	98.45	98.71	98.21	99.77	99.50	98.00	99.79
	9	79.69	81.91	75.37	76.31	82.19	78.83	84.63	80.52

From the results it can be noted that the best classification accuracy for DNNs and DBNs is achieved for the 7-layered and 13-layered networks. This is in line with benchmark results for DNNs & DBNs obtained from the literature but still fall short of the best accuracies achieved using CNNs.

A graphical representation of classification accuracies for DNN is presented in **Figure 6-10.** The bar graph indicates that the 33-layered DNN performs worst in terms of classification accuracies which reiterates that increasing 'depth' may not relate to an increase in accuracy. The classification accuracy achieved for the 7-layered and 13-layer topologies (Blue & Red bars **Figure 6-10**) are very similar and this is a point of interest.

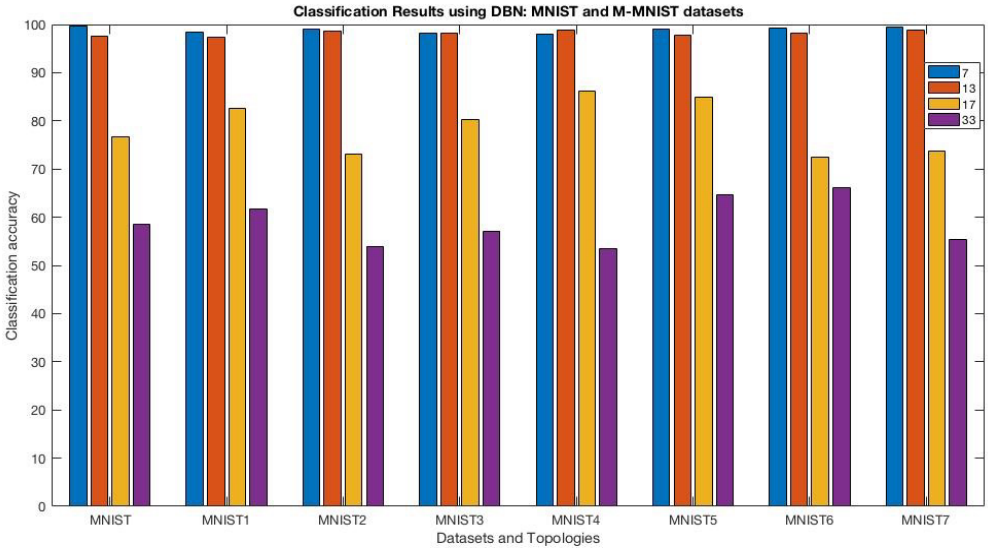


Figure 6-10: Classification Results for MNIST and MMNIST Datasets using DBN

The results from the RBM based DBNs presented in **Figure 6-11** are similar to that of the DNNs with the exception of the 17-layered DNN with MMNIST1, MMNIST3, MMNIST4, MMNIST5 which achieved better accuracies when compared to DNNs. For overall accuracies on MNIST6, DBN achieved better results than DNNs.

Figure 6-12 presents the classification experiment results for three, five and 7-layered DAE networks. DAE based experiments achieved better results for 5-layered topologies with least being with 7-layered topologies.

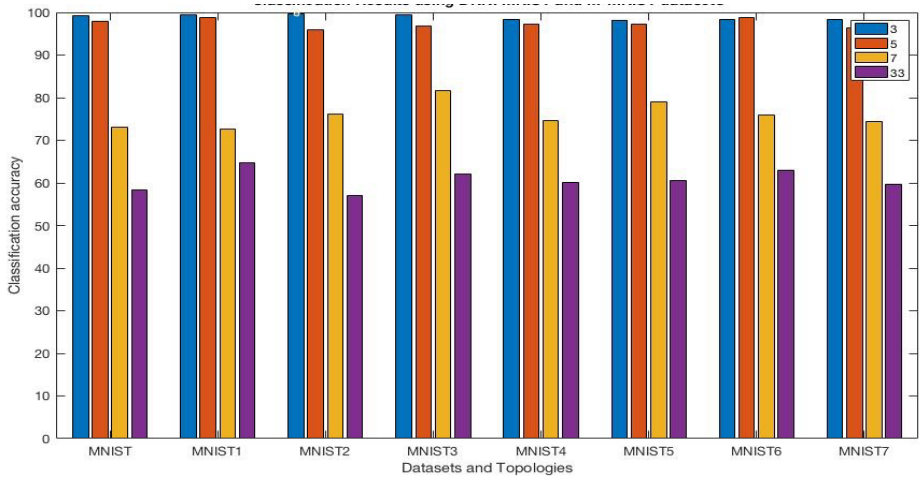


Figure 6-11: Classification Results for MNIST and MMNIST Datasets using DNN

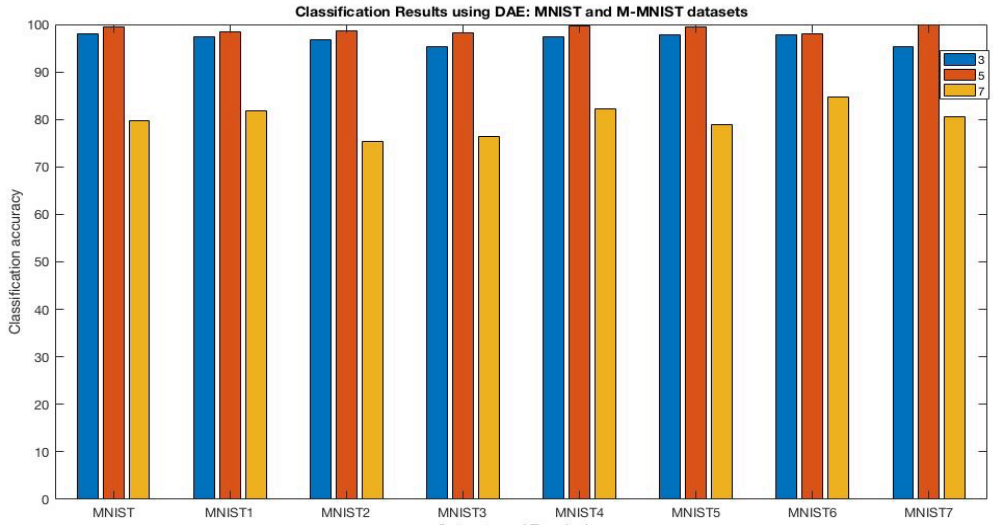


Figure 6-12: Classification Results for MNIST and MMNIST Datasets using DAE

6.6. Image Datasets

6.6.1. ImageNet

The ImageNet database consists of over 14 million images with hand-annotated labels for categorisation. It consists of more than 20,000 categories of which most of them are familiar and common objects [235]. The annotation in ImageNet is framed as whether an object exists or not in the image e.g., 'there is a balloon' or 'there is no balloon.'

ImageNet is considered as one of the challenging datasets for visual recognition. An international competition on ImageNet called the 'ImageNet Challenge' was initiated in 2010 that aims at developing and evaluating efficient ML algorithms. The ImageNet challenge uses a subset of 1000 classes with 90 dog breed classes since the dog breed classes are considered to be the most challenging to classify. The challenge at the initial stages was to be able to achieve an error rate of 16% (in 2012). This target started decreasing with the rise of deep learning. By 2017/2018, the majority of the teams in the competition were able to achieve over 95% classification accuracy. AlexNet [236] achieved the highest accuracy on ImageNet with just a 3.57% error rate in 2016, a significant reduction in value compared to the winner of ImageNet challenge 2015 (6.7%) [237]. Recently a classification accuracy of 97% (top five runs) was achieved using giant (very deep and large) DNNs [238]. However, in an article published in the google blog, Google AML project claimed to achieve highest accuracy on ImageNet2012 dataset using 'AI child bot', outperforming all existing models [239].

For this research, a subset of ImageNET with 200,000 images in 50 categories is used. The dataset and topologies are chosen to achieve over 98% of accuracy after a number of trial and error experiments.

6.6.2. CIFAR-10

The CIFAR-10 is a 10-class image dataset with 60,000 images of size 32 x 32 pixels with typically 50,000 images for training and the remaining 10,000 for testing. The classes are mutually exclusive in spite of overlapping naming conventions. For instance, there are two classes of trucks and automobiles, but automobiles exclude big trucks which are categorised as trucks. When CIFAR-10 was used for the first time, in 2010, a two layer convolutional deep belief network could achieve a classification accuracy of only 79% [240]. Recently in 2018, the Google brain project was able to achieve 98.5% on CIFAR-10 [241]. CIFAR-100 is an extended dataset based on CIFAR-10 and extended to include a larger number of classes, 100 instead of 10.

6.6.3. Classification Experiments

Classification experiments are carried out using 7, 12, and 18-layered DNNs and DBNs on a subset of ImageNet and CIFAR-10 datasets. A set of randomly selected images are modified by changing perceptual image characteristics like colour. The experiments with DNNs are carried out using Google's TensorFlow library [242] (python library). The other experiments are carried out using MATLAB 2018a open source script customised for dataset and experiments. The classification accuracies for DNNs are presented in **Figure 6-13** followed by the results for DBN experiments in **Figure 6-14.**

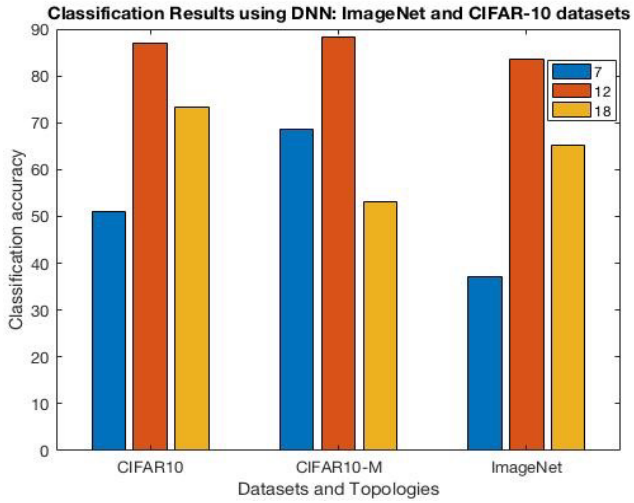


Figure 6-13: Classification results for CIFAR-10 and ImageNet Datasets using DNNs

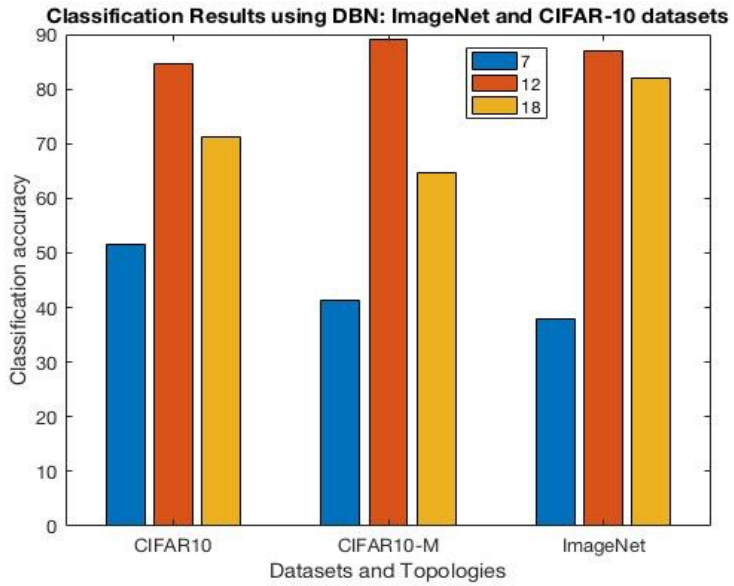


Figure 6-14: Classification results for CIFAR-10 and ImageNet Datasets using DBNs

6.7. Speech and Speaker Datasets

6.7.1. AN4 Speech Data

The speakers' data are extracted from Census (AN4) speech database provided by Carnegie Mellon University [243]. The database comprises of 1158 speech samples collected form 84 subjects of both genders sampled at a bit rate of 16KHz using 16-bit linear sampling. However, this research uses the publicly available dataset with only 948 samples for training (from 53 males and 21 female subjects) and 130 samples for testing (from seven male and three female subjects). All the samples are combined (both training and testing) and 600 random samples are selected for the experiments.

6.7.2. TIMIT

The speech recognition data, used to verify the proposed approach, was extracted from TIMIT Acoustic-Phonetic Continuous Speech Corpus that includes 630 speakers of eight different dialects of American English [244]. The database provides ten phonetically rich utterances of different statements for each speaker, making a total of 6300 samples. The data are recorded in a closed environment at a bit rate of 16 kHz with 16-bit linear sampling. The training and testing datasets are divided in the ratio of 70% to 30%. TIMIT is one of the speech databases that has been widely used as a benchmark to validate speech recognition algorithms and methods [245]. High levels of accuracy have been achieved by normalizing the speaker level mean and variance using strong voice active detection or VAD [246]. The research reported in [247] extracted 39 MFCCs and fed them into a feed forward neural network and a less than 6%

error rate was reported. However, technical details such as topology and parameters are missing in the work which makes it difficult to analyse or compare the results. Further, in [247], the complete TIMIT dataset was not used whereas in this research the full dataset is employed. The highest classification and verification accuracies reported for the TIMIT dataset was achieved using an ANN implementation [247].

6.7.3. Classification Experiments

For NLP, particularly speaker identification, DAEs are predominately successful and produced efficient results [248]. For the AN4 dataset, speech features are presented as MFCCs using MATLAB libraries [222]. 16 MFCCs are extracted from the inputs for each source from the dataset. Once the features are extracted, different type of classifiers are defined using three DAEs with one, three, and five layers. Usage of multiple and divergent DAEs is necessary to test the accuracy rates and to determine the importance of the number of hidden layers in DNNs as discussed in the literature [52, 182, 248].

Though, three different topologies with one, three and five autoencoders are used to create a DAE for both AN4 and TIMIT datasets, for TIMIT dataset an extra 7-layered DAE is also used. The classification accuracies are presented in **Figure 6-15**. The detailed results are presented in Appendix D for the AN4 dataset and Appendix E for the TIMIT dataset. The details of number of nodes, training and testing errors are also presented in Appendix D and E.

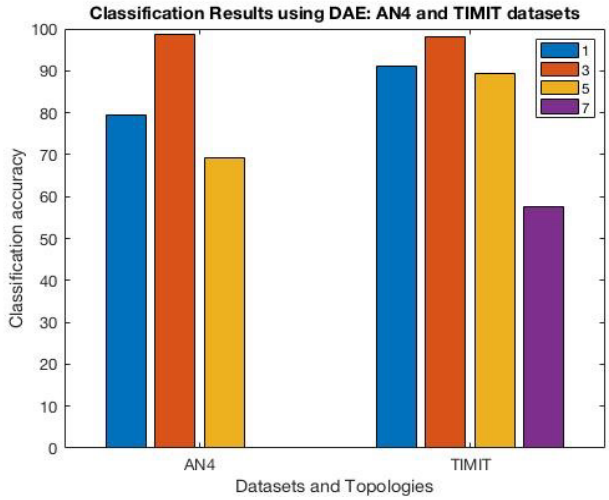


Figure 6-15: Classification results for AN4 and TIMIT datasets

6.8. Air Pollution (CASTNET)

Air pollution data is spatio-temporal in nature which makes it difficult to analyse due to the complexity of features. The dataset CASTNET is the air pollution dataset for the year 2010

obtained from the Environment Protection Agency of USA [224]. The dataset consists of 8700 samples, 13 attributes with two classes based on air quality as being ‘good’ and ‘moderate.’ There is about 6% missing values in the dataset. This missing data has minimal influence in classification.

The classification experiments are carried out using three topologies 7, 13 and 18-layered DNNs and the results obtained indicate that a 7-layered DNN produces highest classification accuracy among all three topologies. The classification results are presented in **Figure 6-16** and the details results are presented in Appendix G.

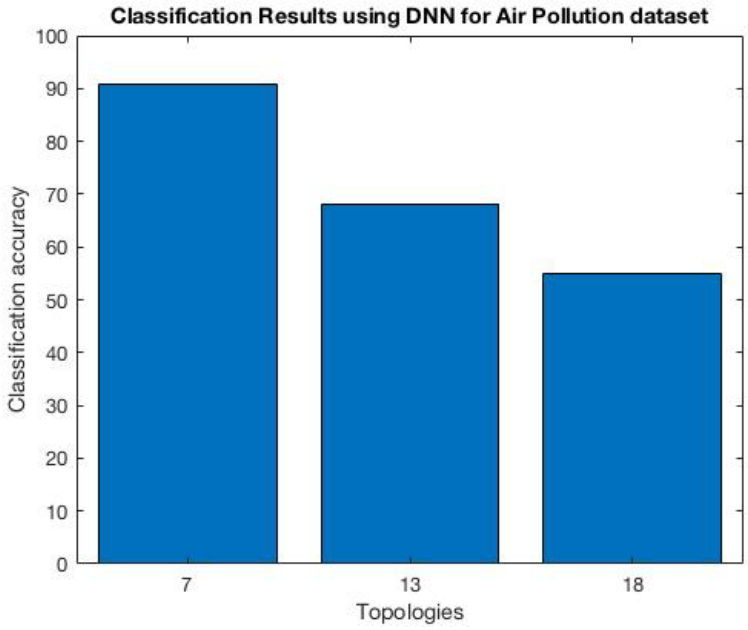


Figure 6-16: Classification results for Air Pollution dataset

6.9. Gene Expression Dataset

The gene expression data is multivariant and often considered as difficult to classify due to the complex associations among attributes. The datasets that have been considered up until this point in this research have more samples than attributes which is a regular practise in designing datasets. However, it is important to test the new hypotheses on datasets, such as gene expression where number of attributes are higher than number of samples. Among all cancer datasets, the prostate cancer gene expression dataset is considered to be the most challenging with 12533 attributes and only 102 samples with 52 normal and 50 tumour cases [249]. The feature map of all 12533 attribute values is presented in **Figure 6-17**.

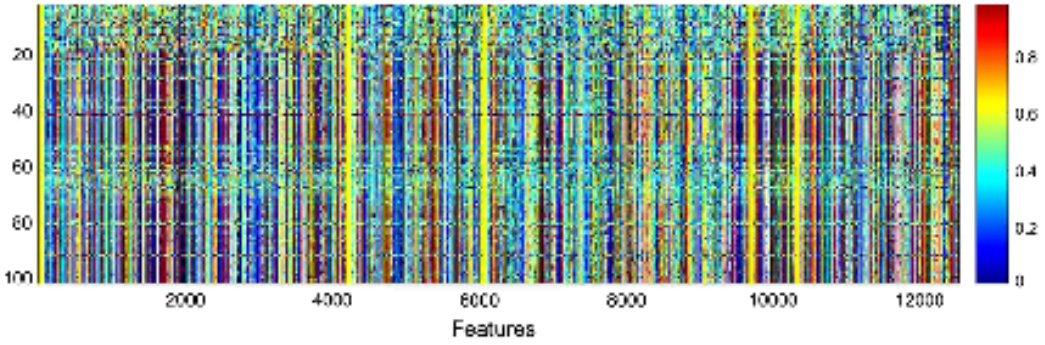


Figure 6-17: Feature Map of the prostate cancer dataset

The classification of such datasets is typically accomplished through attribute selection and evaluation using ML and statistical methods. One such approach is presented in [73] where ANN was used for classification and diagnostic prediction of prostate cancer which achieved a classification accuracy of 100% for classification and 95.1% for predicting whether a given sample is a tumour or not.

The experiments for this research are carried out using 1 to 5-layered topologies starting with one layer. The ANN is not trained using layer-wise training, but instead conjugate gradient-based scale algorithm is used with a learning rate of 0.1 and a momentum of 0.3. The layer-wise training is not used since the experiment topology starts from one layer and it is impractical to implement this on one or two layered topologies. Moreover, the experiments are carried out to optimise the neural network for classification rather than testing training algorithm. A traditional sigmoid function was used as the activation function with BP as the training algorithm. Apart from using the original dataset, this thesis incorporates the testing with a variant of the gene expression dataset GeneExpM. This GeneExpM is created by replacing the original values of significant attributes identified in [73] with random numbers.

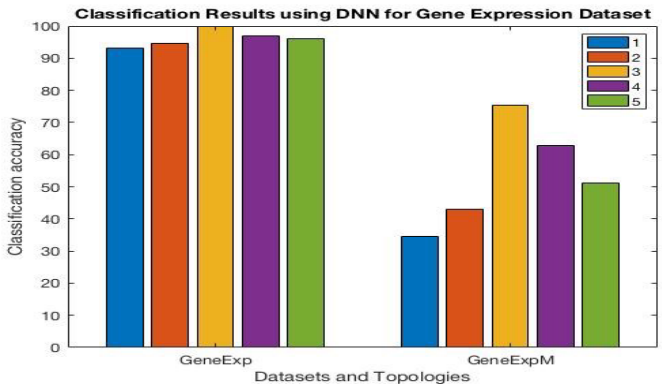


Figure 6-18: Classification results for the Prostate cancer gene expression dataset

The classification results are shown in **Figure 6-18** as bar graphs and detailed numeric values along with the technical details are presented in Appendix H. The results from the experiments performed indicate that best accuracy was achieved for a three-layered ANN for both GeneExp and GeneExpM datasets. This result reiterates that increasing the number of layers may not necessarily improve accuracy.

6.10. Synthetic Hierarchical Dataset

A Synthetic dataset consisting of 90 samples with known feature hierarchies was created. The details of the dataset and the experiment results are presented in Chapter 3.

6.11. Random Values Dataset

In some cases, it is important to investigate how a DNN reacts to a dataset of random values. There are two experimental scenarios followed for the classification of random data. In the first scenario, the attribute values in the existing datasets are replaced with random values. In the second scenario, a synthetic dataset is created and populated with random values. The experiments are carried out on both datasets. In either of the cases, the data in the dataset is purely random with no correlation. This dataset is used to demonstrate the ability of proposed knowledge component model to extract components from the random values with unknown features. It is important to see whether the proposed model could extract meaningful information by which the underlying patterns in a random dataset could be analysed. The classification of these random valued datasets yields no results; hence these were not mentioned. The dataset is used in the experiments is presented in the Section 6.17 and Appendix C.4.

6.12. Summary

This section presented the details of the datasets used in this research to facilitate an insight into the data's characteristics. The results of classification experiments using different ANN/DNN topologies and datasets are also presented. The accuracy of the classification results will assure that the extracted Knowledge Components are efficient since the weights are optimised to achieve best possible accuracies.

PART II: Experimental Evaluation of Proposed Transferable Knowledge Component Model

Overview of the section

Part II of this chapter presents the experimental appraisal of the proposed component-based model to establish a relationship between input features and DNN weights. The experiments are carried out for extracting components, and the results are evaluated for validating the hypotheses.

Transfer of knowledge (transfer learning) experiments are carried out to endorse the importance of the middle layer that contain significant knowledge as proposed in the hypothesis. Initial experiments presented in earlier chapters (Chapter 3 and Chapter 4) are confined to one dataset and typically to one type of topology. The experiments in this chapter explore the importance of the middle layer in more depth and breadth.

Part II experiments fall into one of two categories. The first is a set of experiments for evaluating the hypothesis H2 on proposed knowledge component model. The second set is comprised of transfer learning experiments designed to reiterate the importance of the middle layer (hypotheses H1 & H2) which supports the proposed Blossom Effect.

6.13. t-distributed Stochastic Embedding: Visualisation

Data Visualisation is widely accepted as the one of the best methods to analyse data [250, 251]. Traditional visualisation techniques use dimensional projection to present data and are quite consistent for normal and undistorted data [250, 252, 253]. However, for complex data like high dimensional data, several new tools and approaches are adopted.

The fundamental aim of this research is to show how features are represented in the weights and how these representations are changed according to changes in the input. To demonstrate the influence as clearly as possible, it is necessary to adopt efficient visualisation technique. The volume of data, complexity and dimensionality used in this research poses a challenge for any visualisation technique. Dimensionality reduction is also necessary for high volume data like DNN weights.

t-distributed stochastic neighbour embedding or t-SNE is an ML algorithm for visualisation using nonlinear dimensionality reduction [23]. t-SNE is suitable for visualising high dimensional data in two or three dimensions without much information loss. The objects are projected using probabilistic distance points. For similar objects the points are nearby and closely associated whereas for dissimilar objects they are distinct and far. t-SNE shows the representations based on data values (similarity or distinctiveness). An efficient visualisation will show separated clusters. The values of the parameters for t-SNE are left as default values for all the experiments in this thesis.

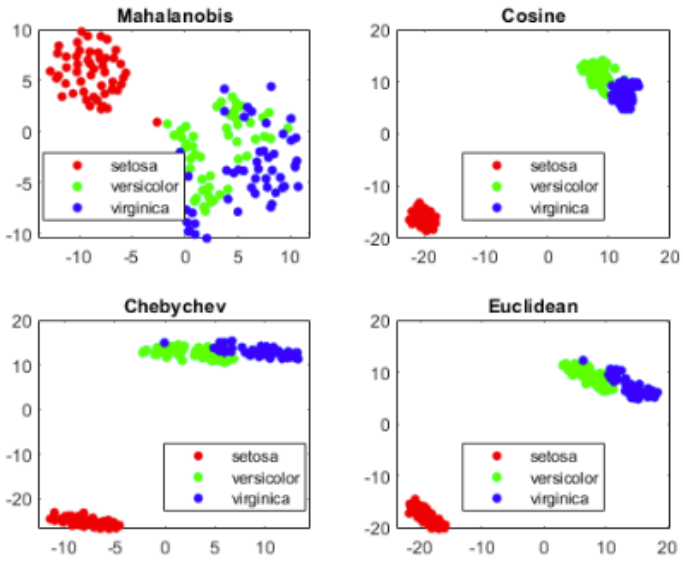


Figure 6-19: Visualization of IRIS dataset using t-SNE for four different types of distance measurements.

The importance of distance between the clusters or object points is very significant in t-SNE visualisations. t-SNE supports four types of distance measurements namely Mahalanobis, Cosine, Chebychev and Euclidean. **Figure 6-19** represents the visualisation of IRIS data using t-SNE all four types of distance measurements presented earlier.

6.14. Experiment Results for Hypothesis 1

Scenario 1 of the hypothesis (Section 4.2.1) is based on the assumption that the input attributes that constitute feature are clearly independent and do not possess any relationship. With such dataset with independent features, 100% classification accuracy can be achieved. There are no existing benchmark datasets with these features. Therefore, the existing benchmark datasets are modified to make them suitable for the experiments. There are four datasets that are used

for this experiment from different domains and categories. These datasets are described in the Sections from 6.11 to 6.14.

6.14.1.Modified IRIS (M-IRIS)

The experiments were carried out on M-IRIS dataset with four independent features. Any attribute correlation that exists in IRIS is removed by replacing correlated values with random values as explained in the Section 6.11.

Four types of topologies: three, five, nine and 13-layered DNNs are used for the experiments and the weights of the three middle layers, the hypothetical knowledge components, are extracted. These weight values (aka knowledge components) are fed as input, one layer at a time, to produce a visualisation.

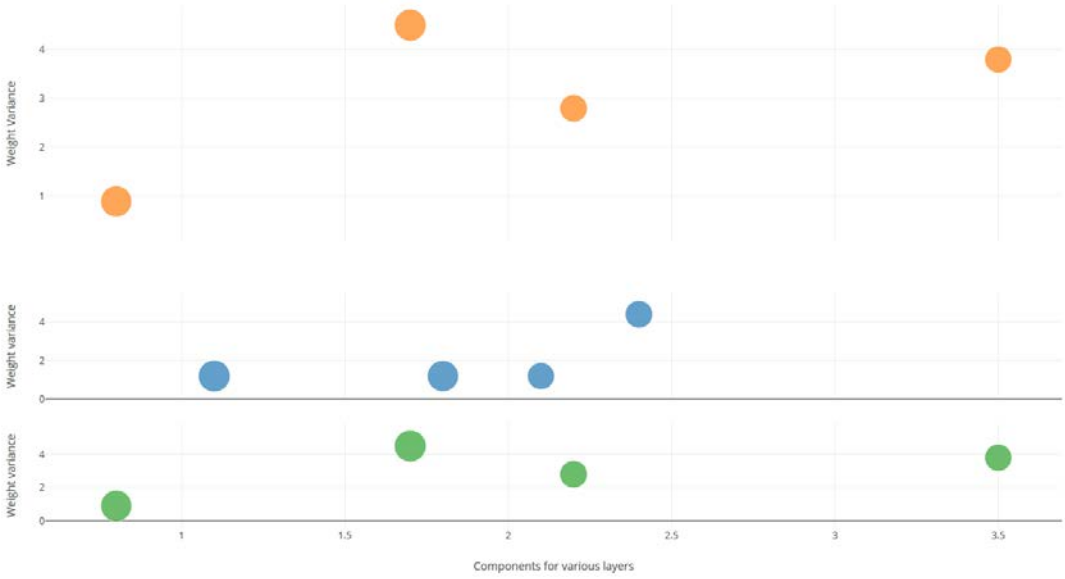


Figure 6-20: Plot of knowledge components extracted from Weights for the M-IRIS dataset.

According to Hypothesis (H1), in case of independent components, number of attributes (n) must be equal to the number of components C extracted for the layers. In this case, therefore, $n = C = 4$. **Figure 6-20** shows the components that are extracted from the layers, layer by layer, with variance in the x-axis and components in the y-axis. Each layer has four components. It can be noted that the components in the middle layer (denoted by the blue dots) show the least variance between them when compared to those of layer 4 (green) and 6

(orange). This is the reduced variance of components in the middle layer that can be related to the learning mechanism of the DNN. In the middle layer, components are purer (cleaner) with least noise thus, significantly contribute to the overall accuracy of the DNN.

The original IRIS dataset consists of three classes and can attain a maximum accuracy of 99.8% since one sample has inconsistent data that disturbs the correlation. The 2D and 3D visualisation of the IRIS classes is shown in **Figure 6-21** and **Figure 6-22**.

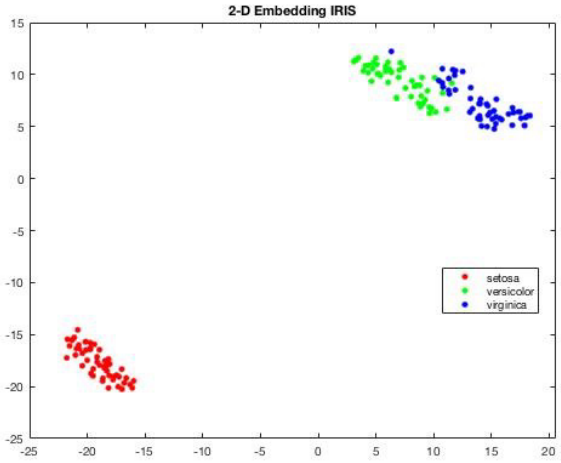


Figure 6-21: 2D visualisation of IRIS dataset (all samples) showing the distance between the classes

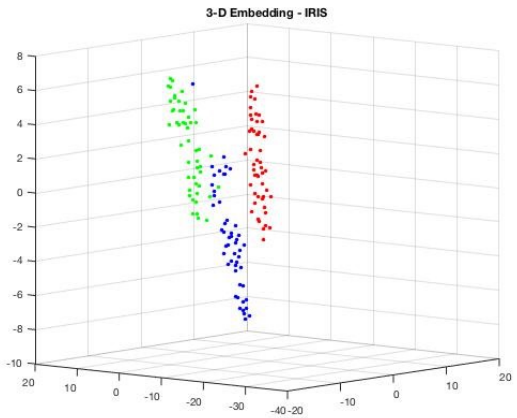


Figure 6-22: 3D visualisation of IRIS dataset (all samples) showing the distance between the classes. The classes appear closer when compared to 2D visualisation (**Figure 6-21**).

However, when looking into the component visualisation for IRIS through the weights of the middle layer of ANN dataset, as presented in **Figure 6-23**, it can be observed that two

components are nearly overlapping, similar to the raw attributes of the IRIS Dataset. However, when the dataset is tweaked to make the variables independent (random values), the visualisation of the extracted components is different as shown in **Figure 6-23**.

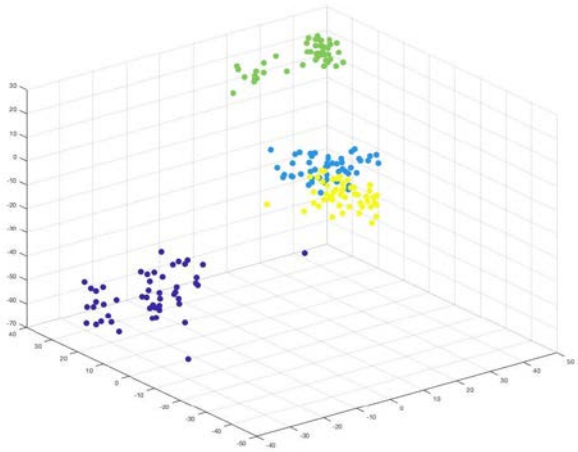


Figure 6-23: 3D Visualisation of attributes (Features) values of IRIS dataset for all samples

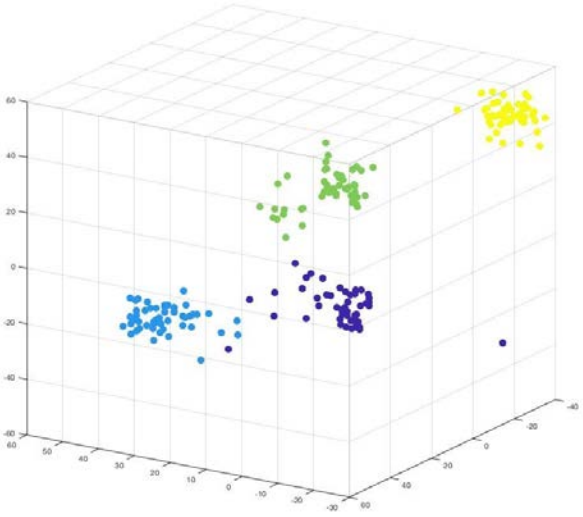


Figure 6-24: 3D Visualisation of the attributes of the M-IRIS dataset. The fourth attribute values (yellow cluster) are adjusted in such a way that it becomes completely isolated which is reflected in the plot.

The representation of components when the dataset is tweaked to create complete independence between the attributes is shown in **Figure 6-24**. To present the notable indication of distance between the clusters and to confirm the independence more clearly, a 2D flat representation as shown in **Figure 6-25**.

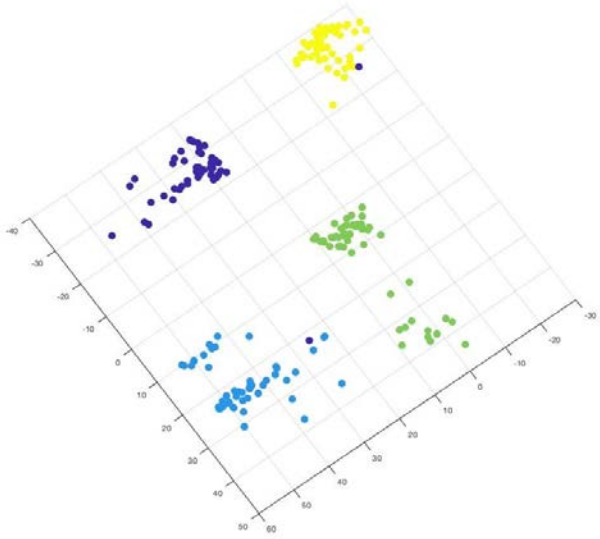


Figure 6-25: Visualisation (flat) of attributes of M-IRIS dataset. It can be noticed that the isolated attribute (with modified values) (yellow) is clearly separated from the rest of the attributes. However, a purple dot can be noticed in the yellow cluster which looks quite close compared to 3D visualisation (**Figure 6-24**).

These visualisations show that when the input (attribute) values are tweaked to make a set of attributes separated, the distinction will be reflected in the projection of weights. This further confirms the presence of underlying representations in the weights that are directly associated with input. Further, to obtain a clear interpretation on the separation between the features, it is necessary to observe the projections in both 2D and 3D spaces with a proper distance measuring function.

6.14.2.MNIST Character Recognition Dataset (M-MNIST)

In case of M-MNIST, the experimental results show a component distribution pattern that is similar to M-IRIS results. Since there are ten independent classes, it is important to observe the behaviour for all three types of topologies that are used in the experiment. Each class is directly attributed to 28 x 28 pixel images (a 784-valued representation per digit).

The topologies used for both DNN and DBN consists of the same structure whereas for DAE it is lesser number and comprised of stacked autoencoders in the form of layers. The number of weights in this case are the same but the number of layers is different. The middle layer for DAE was chosen to be the same as for other two topologies.

The selection of topologies for component extraction is based on the classification accuracy achieved for that topology as reported in Chapter 3 and in this case topology with highest

classification accuracy are selected. Among all the topologies used for DNN, DBN and DAE, the 13-layered topology is selected for DNN and DBN and 5-layered DAE is selected for extracting components. The MNIST dataset has overlapping features since they are handwritten digits with some visible similarities between digits, for example 1 and 7, or 6 and 8. These similarities are considered as underlying commonalities in features and therefore, provide a challenging testing ground for The Blossom Effect.

Table 6-5: MNIST Dataset – Number of components extracted from the weights of various layers for the three types of deep architectures. The number of components is based on the input features which is determined by the component extraction model.

Layer No.	**No. of Components**		
	DNN	**DBN**	**DAE**
1	19	23	29
2	16	19	15
3	14	25	11
4	10	22	14
5	9	17	28
6	9	14	XXX
7	7	9	
8	11	14	
9	12	15	
10	14	18	
11	19	23	
12	21	22	
13	26	25	

The results from the experiments carried out for extracting knowledge components is presented in **Table 6-5**. From these results, it is evident that the middle layer has a minimum number of components in accordance with the Hypothesis H2 proposed in the Chapter 4 Section 4.3. This is in line with the first part of proposed Blossom Effect which is features folding into the middle layer. The folding is similar to the results of encoding the input into the middle layers of autoencoders. The results of the component extraction are presented in **Figure 6-26**. This is

similar to the weight variance graph (reverse bell) which shows that the middle layer has a minimum variance which is responsible for the number of components extracted being minimum. For all three types of architectures, the results are the same. However, the pattern for the number of components is different for DBN, where its value is reduced in the second layer compared to the other two architectures. Further, the number of components extracted from the input data are 8. This reiterates the fact that the middle layer is able to produce a same number of knowledge components.

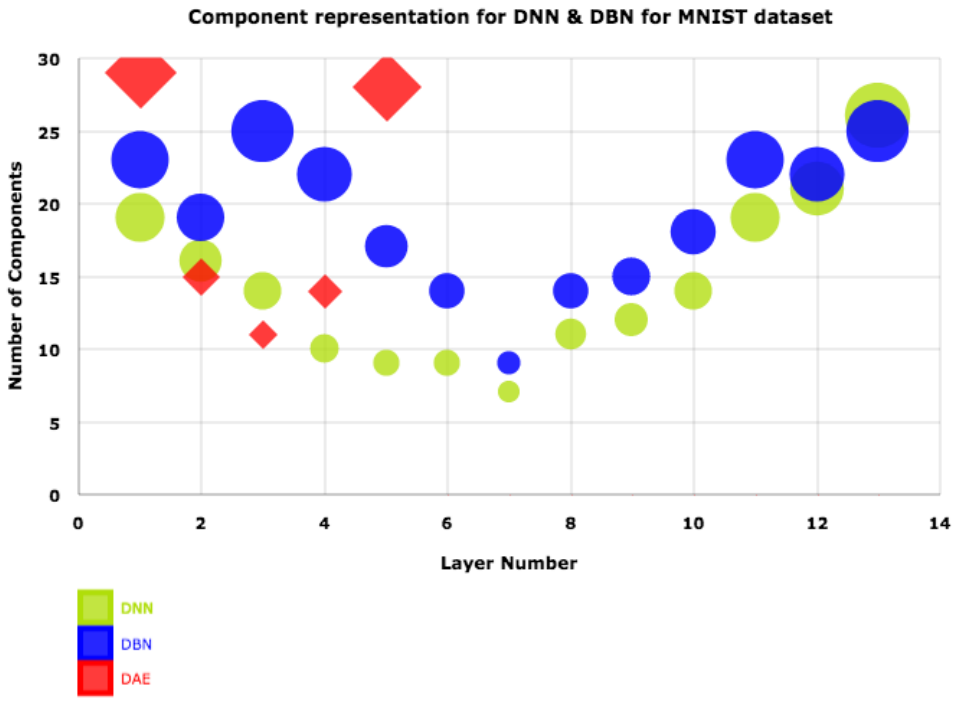


Figure 6-26: MNIST: Component representation for various layers for DNN, DBN and DAE. This figure indicates the components being less in number as they approach middle layer (as proposed and experimentally evaluated in Chapter 3).

The next visualisations are presented in order to provide an insight on the feature patterns for the various modified MNIST datasets. To start with, the visualisation of classification results for the original MNIST dataset is shown in **Figure 6-27**. It can be noted that the input digits in the form of images are clearly separated.

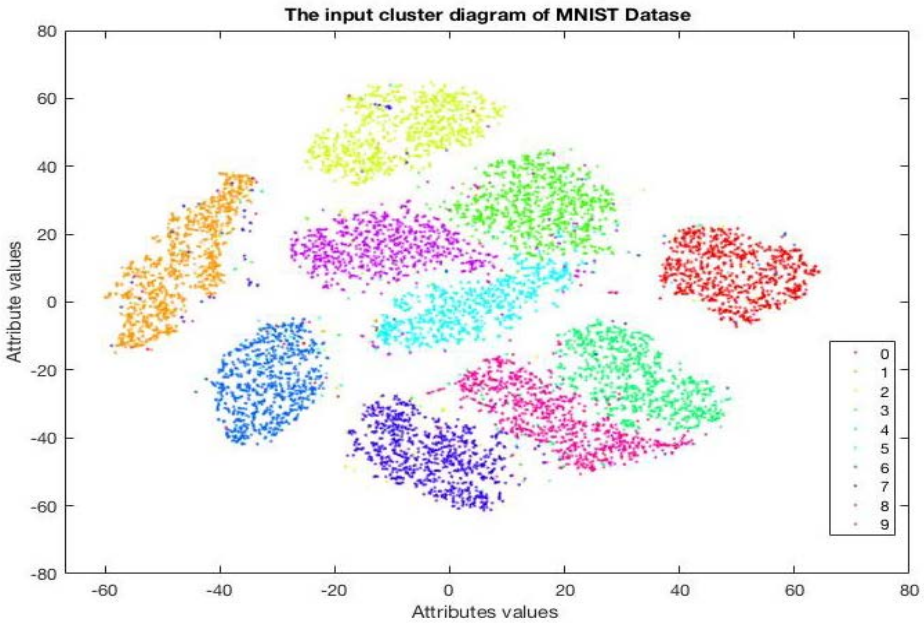


Figure 6-27: Input cluster visualisation of MNIST dataset for all ten digits (0-9) with different colour for each digit. Each component represents the individual digits. This demonstrates that the weights associated with individual class been separable.

From the weights of the middle layer, seven components are obtained which indicates that the input features are represented in seven knowledge components. When the input data are used to extract features, 12 feature components are extracted to represent the features. Since MNIST consists of overlapping features, the number of knowledge components extracted can be trusted. The extracted input components closely match with the number of components in the middle layer. This difference is due to the fact that the components extracted from the middle layer do not resemble the input features completely (100%) due to the nature of the DNN's capability to learn features efficiently [15, 16][1]. Many variants of MNIST datasets are also used in the experiments (see Section 6.12 for details of these modified datasets). These datasets help to demonstrate how weights are represented in the middle layer. There are seven components extracted from the middle layer of the DNN weights and the weight projection is presented in the **Figure 6-28**. The component extraction for M-MNIST4 where the only digits 1 and modified 7 are used is shown in **Figure 6-28**.

[1] A similar set of experiments was conducted with the Wine dataset. For this dataset, the number of input features extracted matched the number of components extracted from the middle layer This match is not unexpected since the classification accuracy achieved for that dataset is nearly 100%. The details of these experiments are presented, for completeness, in Appendix C.

The digit ‘7’ is modified in such a way that the top line is lightened to make it nearly resemble ‘1’. Since the digits ‘1’ and modified ‘7’ are very similar, the overlapping of the features is clearly visible.

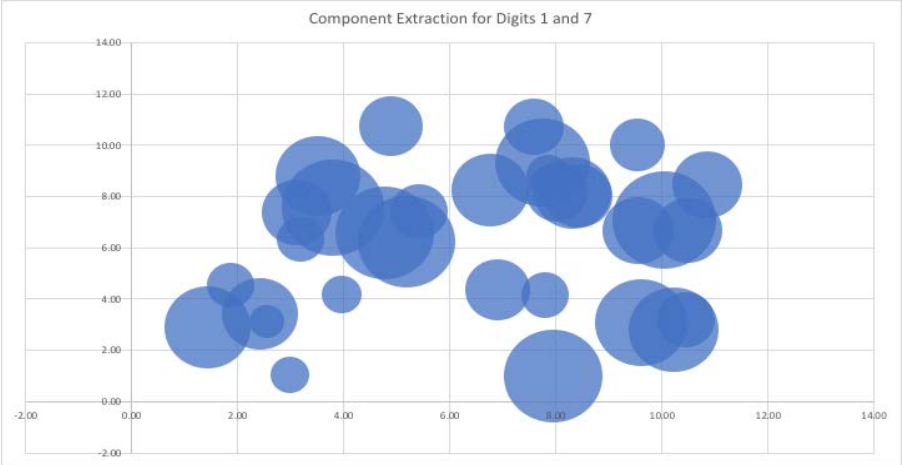


Figure 6-28 : The visualisation of component extraction from the weights for the M-MNIST1(digits ‘1’ & ‘7’) depicting a clear overlapping due to the similarities in the features of digit ‘1’ and digit ‘7’.

Another important aspect is checking experiment results using dataset with two digits with almost no visible resemblance. The digits selected for this experiment are ‘1’ and ‘9’, and the M-MNIST1 dataset. From the weights of the middle layer, two components are extracted. The experiment is carried out multiple times and the results for selected experiments are presented in **Figure 6-29**.

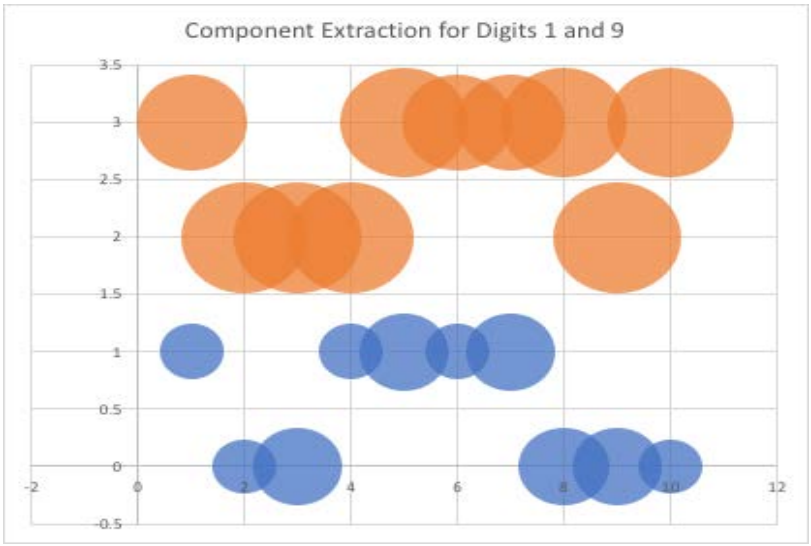


Figure 6-29: The visualisation of component extraction from the weights for M-MNIST5. The components are clearly separated indicating very few overlapping features exist in digit

‘1’ and digit ‘9’. The size of the component shows the strength of the features in terms of number of weights strongly associated.

The datasets M-MNIST5 (digits 1 & 9) and M-MNIST6 (0, 2, 7, & 4) are then tested to identify the component structure for data with less similarities in features when compared to other digits[2]. These experiments are carried out with reference to the hypothesis H2 (Chapter 5, Section 4.4). The component diagram for these datasets is shown in **Figure 6-29**.

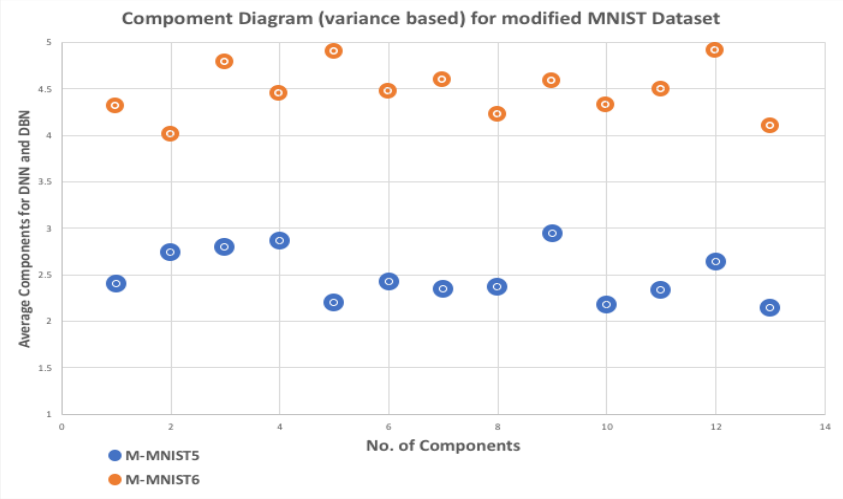


Figure 6-30: The plot indicating the average number of components extracted from each layer of DNN and DBN for modified MNIST Datasets (M-MNIST5 and M-MNIST6)

The M-MNIST2 dataset consists of the digits 6 and 9, and it is possible to reconstruct one from the other. Thus, a similar component model to the M-MNIST1 dataset can be expected which has digits 1 and 7 and 0 and 8. The component diagrams for these datasets are provided in Appendix C.

Hypothesis H1 is further evaluated on image datasets and the results are presented in Appendix F. The experiments result with modified datasets shows that the accuracy varies based on the input feature values and the noise present in the dataset as stated in the hypothesis H1.

[2] Similar results are attained for the random number dataset, see Appendix C.

6.15. Experiments for Hypothesis 2:

The experiments described in this section are carried out to evaluate the importance of the various layers of a DNN and to assess the prominence of the middle layer. The experiments are grouped into two sections. In the first section a set of experiments are undertaken where the classification accuracy of a trained DNN is compared to with and without the middle layer(s), as per the experiments carried out to identify the importance of a particular layer in Chapter 3 (Preliminary Investigation). The experiment results for IRIS, MNIST and the Synthetic hierarchical dataset are presented in Chapter 3 along with the execution times. The key observation is that the execution time was less when the middle layer(s) from the untrained network is replaced with a trained network.

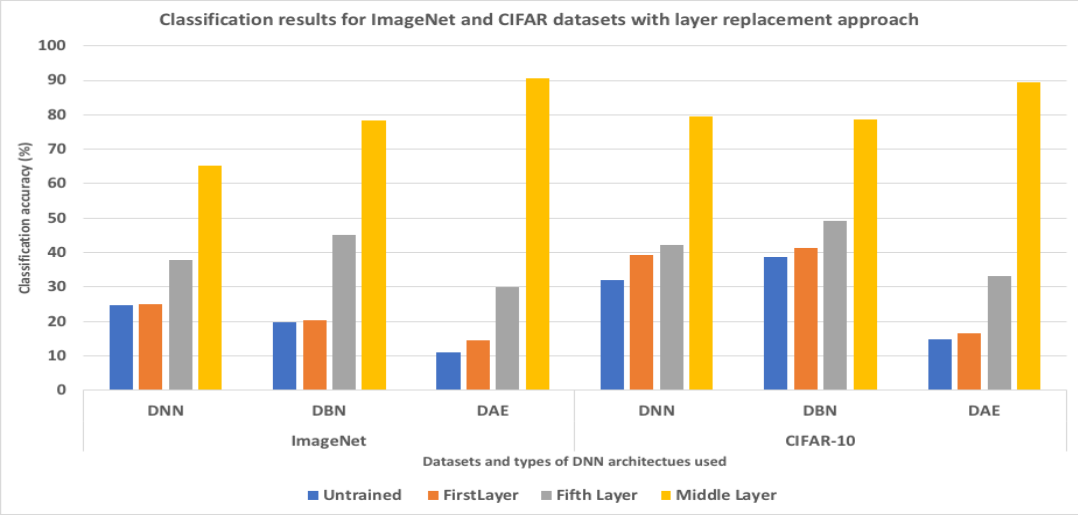


Figure 6-31: Classification results for ImageNet and CIFAR-10 datasets when various layers in the untrained 13-layer network are replaced with components extracted from the layers of a trained network

The classification results for these experiments are considered from the perspective of the highest achieving topologies for DNN and DBN, the 13-layered topology. For ImageNet and CIFAR-10[3], the comparison of classification accuracies of an untrained network with the accuracies when various layers are replaced with the components from a trained network is presented in **Figure 6-31.** It can be observed that there is a considerable improvement in classification accuracies, when the untrained weights are replaced by the component weights extracted from the trained network. Before converging, the highest accuracy is achieved for the network with the middle layer being replaced. These results are similar to the classification

[3] The experiment results using the TIMIT dataset produced similar results to ImageNet and CIFAR. The results for the TIMIT dataset are presented in Appendix F.

results achieved for the initial experiments on IRIS, Synthetic dataset and MNIST datasets (see Chapter 3).

This shows that the transfer of components in the middle layer has given a warm start for the experiment and produced better accuracy and reduced execution time, which is presented in **Figure 6-32**. The full details of the experiment results are presented in Appendix F.

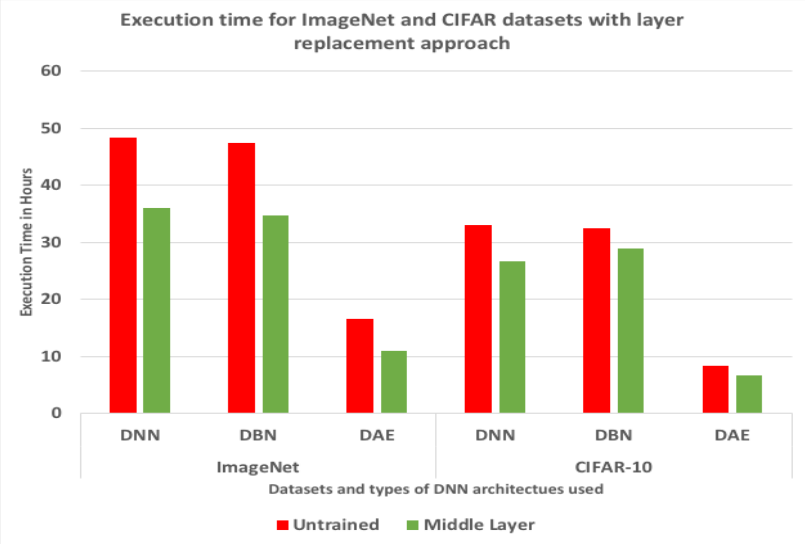


Figure 6-32: Execution time for classification experiments on ImageNet and CIFAR-10 datasets with and without replacing the middle layer.

The second set of experiments are carried out to compare the input feature components with the number of components extracted at each layer. Further, when selected features are removed from the input, the number of weight components extracted from the middle layer must show a clear indication or impact. **Figure 6-33** presents the number of weight components extracted from the middle layer when the input dataset has 100% of features, 70% of features and 50% of features respectively.

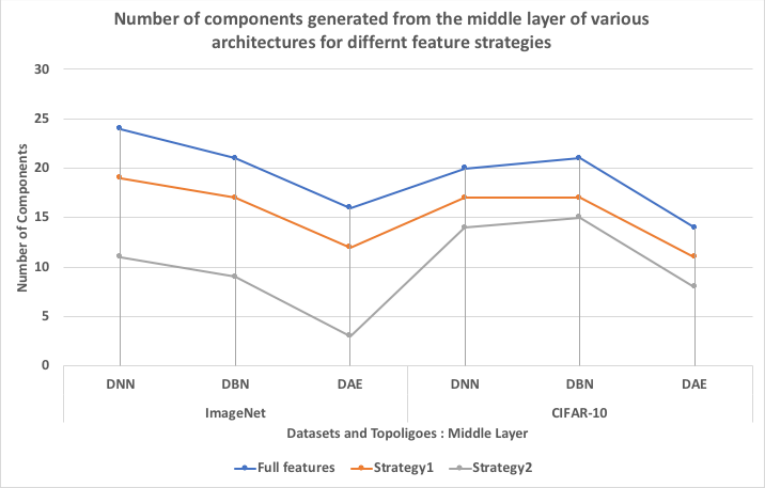


Figure 6-33: Number of components extracted from the middle layer for various topologies and datasets. Three strategies: Full features, Strategy 1: removing 30% of features, strategy 2: removing 50% of features (randomly)

The feature removal is purely at random using open source MATLAB code for removing $n/3$, $n/2$ attributes from the total number of attributes n. To maintain the consistency with the other experiments carried out for this research, three types of architectures (DNN, DBN and DAE) are used for the experiments. The experiment results show that transfer of the middle layer produces significant impact on accuracy of the deep architectures (DNN, DBN and DAE) and the component extraction follows same pattern i.e. middle layer has minimum number of components as shown in Appendices E, F and G.

6.16. Application of the Proposed Knowledge Component Model

The prospect of transferring knowledge from one system(trained) to a new system(untrained) will provide a warm start to the new system and will help in reducing training time. Since DNNs are recently categorised as connectionist systems [254], the possibility of having two different systems for training and testing cannot be ruled out. Since the execution time for DNN based AI systems is high, it is practical to have a faster testing system which utilises the knowledge from the training systems. For instance, a DNN system with 13 layers is optimised, the same sized 13-layered network may not be necessary for the testing system. Therefore, a reconstruction of a DNN with a smaller number of layers based on knowledge components / middle layers could be used for testing. This is a type of knowledge transfer approach method being practically implemented.

A second approach that might be worked out in the future is using trained deep architectures in portable form as deployable models. Since the knowledge is concentrated in the middle

layers, the layers near the input may not be necessary. So, these insignificant layers (if fully identified) can be either removed or combined with the previous or next layers. The layers may have been useful to build-up the knowledge at the middle layer, but their importance will be diminished once the middle layer has enough knowledge. The approaches proposed above are implemented in three applications which are presented below.

6.16.1.Deep Autoencoder Model for Digital Watermarking Analysis

Digital watermarking has been used to authenticate documents, images, videos and other e-resources. The case of identifying whether or not a watermark exists is an identification/pattern recognition problem. This identification of the watermark gets more complicated when the source is a scanned documents/images, and particularly for distorted sources where only a portion of the watermark may exist. The NWND dataset of 444 images is watermarked with text, image (logo) and shape, making the total sample size 1776. A three-layered DAE network is used for the identification experiments against traditional feed-forward ANNs. For both classification and identification, the DAEs outperformed the ANNs with 77.9%, 82.1% and 64.2% for classification of different watermarks and 86% of accuracy for identification. This work, which was published in 2018 [255] was extended in this doctoral research to provide the possibility of portable DAE for identification. After training the three-layered DAE for classification, a new DAE_{test} is created with one layer extracted from the original three-layered DAE. A simple fine tuning is performed on the new DAE_{test} which has taken about one-third of original training time. When the DAE_{test} is used in the identification experiment, 86 out of 100 images are identified to be correctly producing exactly the same accuracy rate of the original 3-layered DAE. The results from this experiment further affirms the transfer of knowledge through aspect proposed in this thesis. However, there is a marginal difference in error values (rmse) for 28 images (average rmse = 0.134) where in transfer modal (DAE_{test}) had less error ratio compared to original DAE.

6.16.2.A CNN based Model for Image Analysis

The experimental evaluation on CNNs is excluded since this thesis is based on only feed forward networks. This section is a preliminary investigation for implementing an approach similar to transfer learning for image analysis. Though this work is published, this is not in the scope of the thesis but could be considered for future work (as presented in future work section of Chapter 7).

The second set of experiments was performed using CNNs on the CIFAR-10 dataset. Initially, a CNN with six convolution layers is trained with a subset of 5000 images from the CIFAR-10 dataset. The experiments are performed using TensorFlow with the keras library (cifar10_trained_model.h5) [256] which achieved 89% accuracy.
Max pooling is then adopted for pooling layers and the experiment performed for 100 epochs. The pooling layers consist of 32 x 32 pixel for de-dimensioning with a batch size of 32 and the final Softmax layer with ten nodes (number of classes). After the CNN was trained, the testing experiments are performed and attained an accuracy of 79%.

Further experiments are performed for modelling a portable-CNN by removing one layer at a time, starting from the first layer and without any new training. For the 5, 4 and 3-layered CNNs, the model is able to produce a prediction accuracy of 79%, 76.5% and 78.6% respectively. These results clearly show the influence of the different layers. With a simple fine-tuning, the 3-layer network is able to achieve 79% accuracy with only 12% of additional time for training. The execution time was about 39% of the time required original 6-layer CNN since the number of layers are reduced, which makes this portable model a workable model.

6.16.3.A Transfer of Knowledge Applications

Transfer of knowledge, or in some cases referred as transfer learning, has attained success for various applications. As a part of this research, experiments are carried out to extract a transferable model instead of merrily transferring the entire layer. The application of such a model provides a clear view of whether or not what is being transferred is significant knowledge (that is ‘sufficient’ to provide required results). The transfer of knowledge components approach has high potential in industrial applications due to reduction in the size and amount of parameters/values to be transferred. This transferable component method is applied on three different types of deep architectures with different datasets and domains.

DAE approach for classification of corrupt datasets

In line with the experiment results mentioned in the previous chapters and the publication referring to the knowledge transfer approach using DAE [56], the proposed transfer learning component model is applied to this work. A synthetic hierarchical dataset is used for this experiment which is the same dataset that is used for the initial experiments reported in Chapter 3. With the initial model of transferring the layers from a trained DAE to an untrained DAE, the classification accuracy improved to 78.9% from 56.7%. The current experiment is

performed by extracting transferable components from the original DAE and transferring them to a new trained DAE. The component model achieved a classification accuracy of 72.1% without fine-tuning. When fine-tuned, it was able to achieve the original accuracy of 78.9%. It is noteworthy to observe that the training time has reduced considerably since the entire network is not re-trained. The testing data is independent and is not included in the training dataset.

CNN model for the approach for identification of digits classification in MNIST Dataset

The experiments are carried using keras modelled CNN with 6 layers on the MNIST dataset using TensorFlow. Firstly, a CNN called CNN_{Train} is trained on all 50,000 training samples and when tested, it achieved an accuracy of 94.5%. The test data is provided to a three-layered CNN called CNN_{Test} which is not previously trained and the classification accuracy at the first run is recorded as 19.8%. This is followed by transferring all the parameters and values of the last three layers from the trained CNN_{Train} to CNN_{Test} and running the testing experiment. This approach achieved an improved accuracy of 78.3% with fine-tuning using an additional Softmax layer.

A further experiment with the replacement of weights of the three-layered CNN_{Test} by the component values extracted from CNN_{Train}, achieved an accuracy of 74.4% after fine-tuning for 13 times. This decrease in accuracy (~4%) is attributed to the complexity of the CNN structure and its learning procedure. The pooling layers are the controlling factor in the case of CNN. Therefore, the proposed component model requires special type of extraction method for these layers since they are not just regular weights. However, the significant reduction in the training time in spite of fine-tuning the CNN multiple times indicates that this approach cannot be ruled out. CNNs are always considered as a special case of deep architectures and their limitations, particularly their restriction to image analysis and weakness in learning mechanism is a point of debate. Well known deep learning expert Hinton has expressed the same opinion in his recent work and proposed a new approach called matrix capsules to overcome the limitations of CNNs [257].

Transfer of knowledge for evolving DBN

The research on evolving DNNs using evolutionary strategies while working on this thesis [52] has inspired to work on evolving DBNs. However, instead of starting from scratch which was

the earlier approach adopted in this research for DNNs, the experiments detailed in this chapter are carried out to evolve problem specific DBN test networks which can be deployed and can run with minimal hardware requirements. The approach has been tested on small datasets with significant success but achieved less, but still notable success due to longer training times rather than lower accuracy on larger datasets.

6.17. Summary

Part II of this chapter provided the experimental evaluation of the proposed Transferable Knowledge Component model. The experiment results provide decisive evidence in support of the Blossom Effect proposed in the hypothesis. The next section provides assessment and reconciliation of the proposed approach which is utilised in these experiments.

PART III: Assessments and Reconciliation

Overview of the section

This section presents the reconciliation on how the proposed hypothesis is tested and validated through the experimental evaluation and analysis of the results. The principle findings based on the relationship between the input features and the weights in the hidden layers are presented in this chapter. Finally, the Blossom Effect proposed in this thesis is justified by the findings obtained through the experiment results. This concept instigated a re-thinking of the research directions on the internals of neural network learning.

6.18. Validity of Research Hypotheses

The validity of the hypotheses is carried out by analysing the experimental results detailed in this thesis. The hypotheses presented in Chapter 4 are revisited here.

Hypothesis 1

H_1: For a given input with n features with x% of least relevant information (noise or distortion) that significantly effects the accuracy, x is distributed among L/2 layers in which there exists c components in the middle layer (L_m) such that when x is minimised, $c \leq n$.

There are three scenarios were detailed for H1 and these are considered here.
The first scenario of H1 states that "*the number of components extracted from the layers must equal the number of input variables when the attributes/features in the input are not overlapping.*"

Scenario 1 is validated through the experiments carried out on the modified MNIST datasets. The results of the experiments on M-MNIST5 (digits 0, 9) and M-MNIST6 (digits 0, 2, 7, 4) supported the hypothesis. This is further validated on datasets with independent variables such as the random values dataset and the experiment results are presented in Appendix C.4.

For the modified IRIS dataset, when the dependency/correlation of attributes is removed (removing the correlation for the class), the number of components extracted are same as the number of attributes as proposed in Hypothesis 1. The independence of these features is reflected in a clear separation of feature components, as shown in **Figure 6-23** and **Figure 6-24**. The number of components for different input feature-based strategies is presented in **Figure 6-33**. The number of components extracted is based on structure of the input features. Similarly, when the relationships are removed from the Synthetic hierarchical dataset, the feature association becomes distracted and the overlapping of features is reduced as presented in the previous section.

When the number of components is the same across all layers, a 'pipe' like structure is produced. Thus, Hypothesis 1 is validated across four different types of datasets and after analysis of results, H1 has tested to be true.

Considering the results from the MNIST experiments, the second scenario of Hypothesis 1 can be verified. The hypothesis states that *the middle layer has significant features and that number of middle layer features matches closes to the number of input features*. The initial experiments carried out in Chapter 3 produced affirming results. Further, the experiments carried out on other datasets have reasserted the 'reverse bell graph' principle, presented in Chapter 3, **Figure 3-2** (Chapter 3 Section 3.7), which is the results obtained for the MNIST dataset. The experiment results for rest of the datasets including Wine, AS4 are presented in Appendices C and D.

The third scenario is presented for Hypothesis 1 to cover all possible combinations of input features that influence the component arrangement in the layers of a DNN. The first two scenarios test the possibility of features being either fully independent or partially dependent. The third scenario checks the case where the features are 100% overlapping. i.e. every input attribute is associative. This scenario helps to reject the possibility of the null hypothesis. The most suitable datasets for these types of experiments are ImageNet, CIFAR1-0, TIMIT and the Synthetic (hierarchical) dataset with known FHs. The experiment results presented in Chapter 6 part II confirm the truth of the statement presented in scenario three. Thus, all the scenarios are assessed and proven to be correct which combined to validate Hypothesis 1.

6.19. Principle Findings on the Relationship between Input Features and Neural Network Weights

The evaluation of Hypothesis 2, the Blossom Effect, through various experiments paved the way for the observation of the weight patterns in a neural network's hidden layers. This research provides an insight into the neural network weight patterns.

The experiment results presented in Part II of this chapter indicate that the number of components extracted from the weights is directly dependent on input features. **Figure 6-34** presents the visualisation of weights from the middle layer for the experiments carried out using the M-MNIST2 dataset with two digits, '6' and '9'.

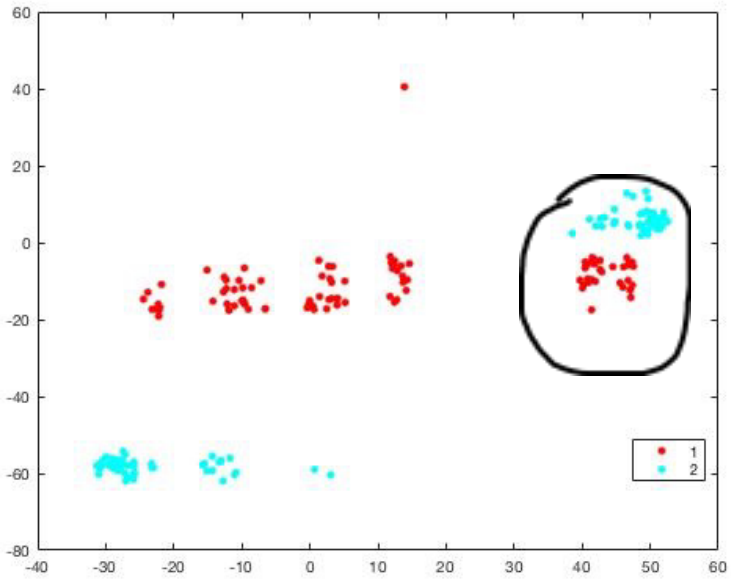


Figure 6-34: Visualisation of weights for digits 6 & 9: M-MNIST2 dataset. The commonalities represented by weights that are closely banded.

The reduced dimensional weights of digit ‘6’ are indicated in red with component 1, whereas the digit ‘9’ is indicated in cyan with component 2. The weight pattern indicates a separation between the components of ‘6’ & ‘9’, with one set of weights of digit ‘6’ being close to that of digit ‘9.’ This indicates the commonality of features for ‘6’ & ‘9’, that may be of shape or area.

When the contrasting case of digits ‘1’ and ‘7’ (M-MNIST1) is considered, the visual representation of weights indicates a different picture as shown in **Figure 6-35** where the red colour indicates digit ‘1’ and the cyan colour indicates digit ‘7’. The features are clearly overlapping with a minimum distinction which indicates the overlapping of the attribute values/input features.

To pursue the validation of Hypothesis 2 further, it is important to test the results by repeating the experiments and modifying the same dataset. This modification of M-MNIST1 carried out by greying out the top bar of images of the digit ‘7’, this makes it appear more similar to the digit ‘1.’ M-MNIST4 is the dataset with this modification, and the weight component visualisation in a reduced dimensionality is presented in **Figure 6-36**. The figure, where instances of digit ‘1’ are represented by red dots and instances of the grey-out digit ‘7’ are shown in cyan, clearly shows the overlapping of features when the top bar of the digit seven is greyed out.

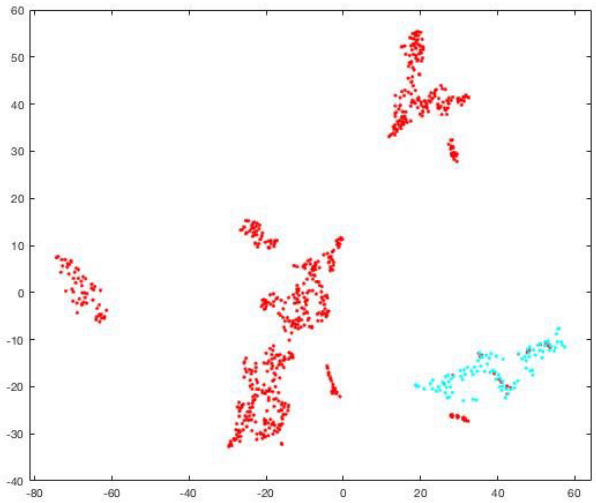


Figure 6-35: Visualisation of weights for digits 1 & 7: M-MNIST1 dataset. The is considerable overlapping between the two digits and the isolated cyan indicates the difference in the features probably the upper part of digit '7' and the lack of angle in digit '1.'

The experiment results are a strong indication of how input features affect the weight patterns. The visualisation indicates the reason for the number of feature components that are extracted from weights.

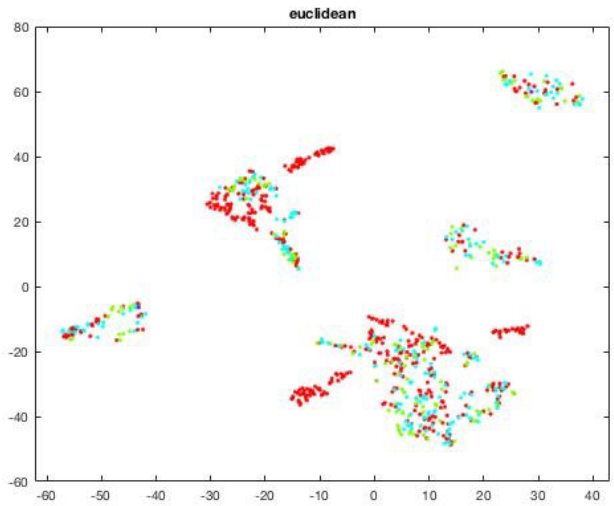


Figure 6-36: Visualisation of the weights for digits '1' (Red) & '7' greyed out (Cyan): M-MNIST4 dataset. The overlapping in the majority of the parts indicates that when the upper part of digit '7' is greyed out, the majority of the representations in the weights are common and overlapping in digits '1' and '7.'

In the case of distinctive features, a clear separation can be observed. 60 samples were extracted from the ImageNet creating two categories of images with distinctive characteristic features. Some of the images were handcrafted so that there are no known similarities. One such example

is a triangle and a circle where the triangle was created with dashed lines. Category one has coloured images excluding the colours black and white, whereas the images in the category two are black and white. Category one has images of animals and living organisms whereas category two has transportation vehicles. When trained on a DBN, the initial projection of the weights with four components in the middle layer is presented in **Figure 6-36**. There are some commonalities at the discrete level which are exposed in red and cyan colours, whereas the two other components are almost invisible.

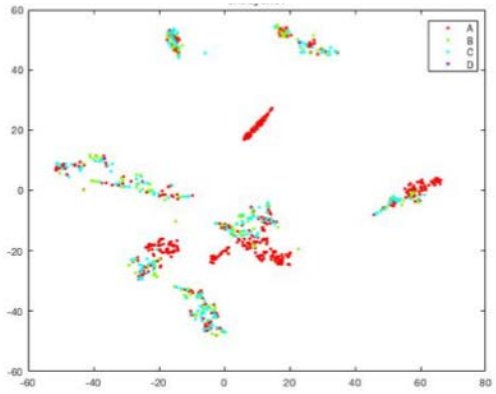


Figure 6-37: Component projection for weights from the middle layer of DBN for modified ImageNet Dataset with 60 handpicked samples

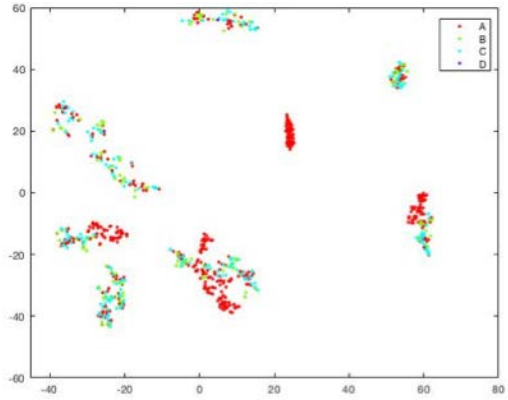


Figure 6-38: Component projection for weights from the middle layer of DBN for modified ImageNet Dataset with 30% of samples modified

When the 30% of the random pictures from category two are replaced with coloured images, there is a change in the manifestation of weights with an increasing presence in component A. With a few more experiments, the projection of the middle layer components could provide low level information on which characteristic feature influences which component, as shown in **Figure 6-38**.

Similarly, fine-tuning the DNN has an immense effect on placement of weights in the component model which is presented below. Two speaker corpuses are selected from the AN4 dataset and trained on a DNN without a Softmax layer. The features that are extracted from the middle layer after layer-wise training are shown in **Figure 6-39**. When the DNN is fine-tuned with a Softmax layer utilising the traditional BP algorithm, one may observe the change in the patterns of the feature components. A change in the scale can also be noticed. The key

difference in the representations is highlighted in the **Figure 6-39**. It can be noticed that some of the weight groups have been disbanded as shown in **Figure 6-40**.

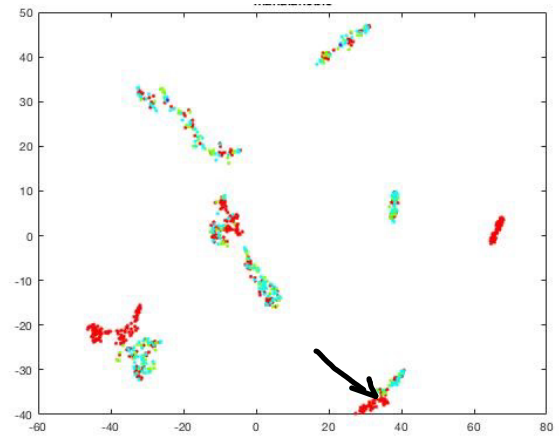


Figure 6-39: Before fine-tuning : component projection of weights from the middle layer of one speaker data from AN4 dataset

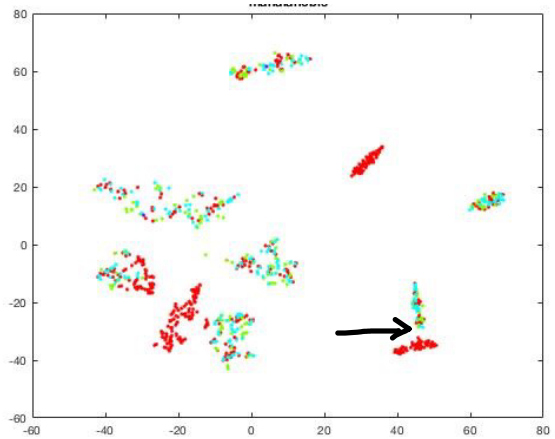


Figure 6-40: After fine-tuning : component projection of weights from the middle layer of one speaker data from AN4 dataset

Sometimes, the variety in the characteristics of the input attribute creates a confusion of classification. However, the middle layer in the DNN can differentiate and segregate these characteristics. The components extracted from the middle layer might have only one pattern, but the representations vary based on the input features. For instance, consider a randomly generated dataset of three attributes with a minimum variance of 0.001 between the values. The dataset consists of two classes with overlapping attribute values. For the first experiment, when the DNN is trained using this random data, the DNN was able to achieve an accuracy of 69%. It is possible to achieve a 100% accuracy when the DNN is fine-tuned and some values in the dataset are adjusted. When the components are extracted from the middle layer of this 7-layered DNN, there is only one component isolated with multiple features (groups) as presented in **Figure 6-41**. Only a single component is derived due to the fact that the classes are not separated due to the presence of overlapping (attribute values) features. Despite this overlapping, the DNN is able to differentiate the classes which indicates the unique capability of deep learning.

When the dataset is adjusted to ensure that there is no overlapping of features, the middle layer now consists of clearly differentiable features and a single component. The components features are shown in **Figure 6-42**. Thus, the extracted component is a set of the same features

in both of these experiments. The patterns of the weights in the hidden nodes are not just a reflection of input features but possess an underlying representation in the form of features which the DNN is learning and passing over to the next layer as knowledge. It is noteworthy to observe that this process of understanding the representations in the weights gets more and more complex with the increasing number of samples, classes as well as hidden nodes.

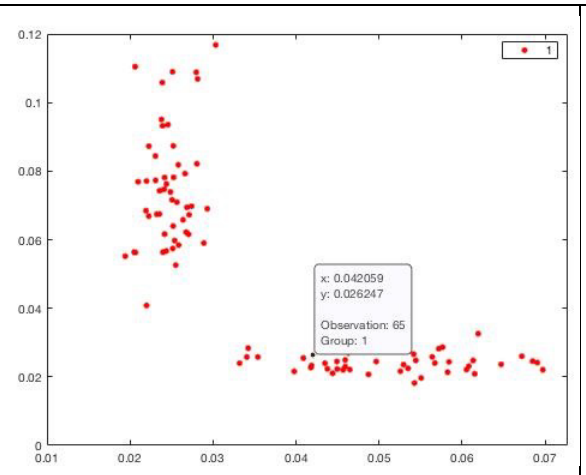


Figure 6-41: Component extraction from the middle layer of a 7-layered DNN. Random value dataset with fully overlapping variables

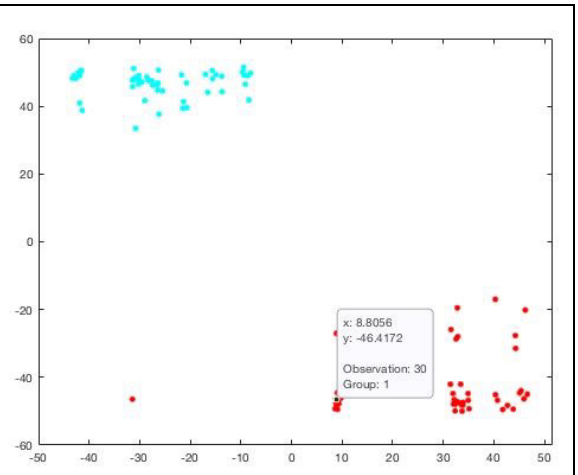


Figure 6-42: Component extraction from the middle layer of a 7-layered DNN: Random value dataset with two classes

The impact can be seen with the help of visualisations, but it is hard to reason out what exactly is responsible for this functionality and how the DNN is gaining this expertise. However, the layer-wise training has a high impact and may be responsible in providing 'clarity' for the DNN.

To throw more light on this aspect, the visualisation of the weights extracted from the middle layer of a DNN trained on a random value dataset is used. The random value dataset is a modified version of gene expression dataset with 102 samples and about 12500 attributes with two classes. To construct random values dataset, the original values in the attributes are replaced by random numbers and the number of classes is increased to ten. For the first experiment, the dataset is modified such that there is 90% of attribute values belonging to all ten classes (90% overlapping). This is followed by reducing the overlapping to 75%, 60% and finally 50%. The DNN is trained to achieve highest possible accuracy of 90%.

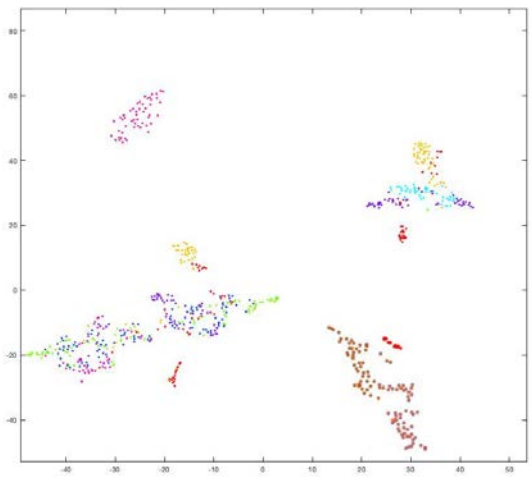


Figure 6-43: Attribute overlapping 90%: visualisation of middle layer for random dataset

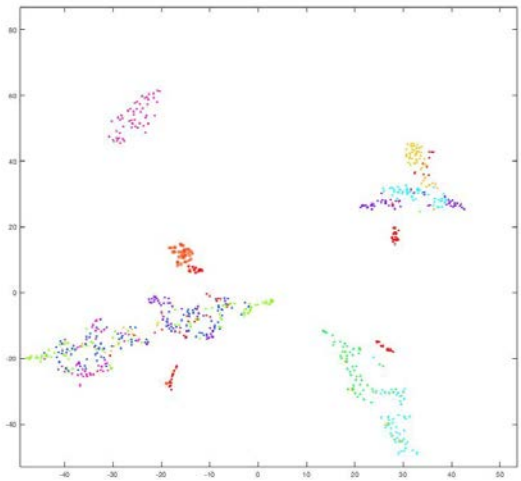


Figure 6-44: Attribute overlapping 75%: visualisation of middle layer for random dataset

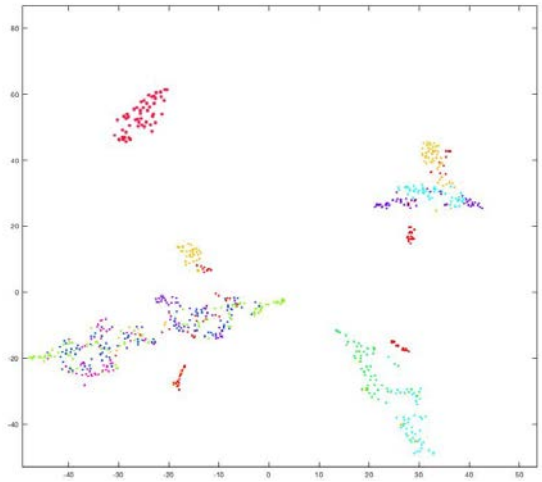


Figure 6-45: Attribute overlapping 65%: visualisation of middle layer for random dataset

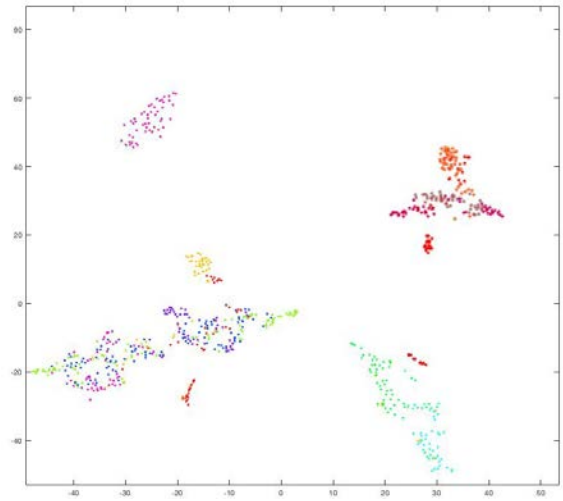


Figure 6-46: Attribute overlapping 50%: visualisation of middle layer for random dataset

The projection of weights is presented in **Figure 6-43**, **Figure 6-44**, **Figure 6-45** and **Figure 6-46**. The transformation of weights based on the degree of attribute overlapping which in turn is feature overlapping can be observed. The segregation or grouping of features is based on variance values and represents the underlying components.

6.20. Conclusive Assessments: The Blossom Effect

The process of identifying how underlying features are represented has been the most challenging part of this research and has a direct relationship with the research problem as well as exploring the behaviour of neural networks.

The initial experiments are able to prove the importance of the middle layer(s) which has been evaluated through further theoretical (literature) and experimental (as part of this thesis) evidence.

The experimental results presented in this thesis have proven the existence of a funnel effect i.e., number of components exacted keeps reducing towards the middle layers and then increases after the middle layers, creating a funnel.

It is evident that in traditional ANNs with only one layer, the features are condensed in the hidden units of that (single) hidden layer. However, with DNN using layer-wise training across multiple layers the features are spread across the layers. The decrease in the number of components at the middle layer indicates that the features are condensed in the middle layer and then are expanded towards the output layer (classifier). This phenomenon encourages the researcher community to rethink the functionality of autoencoders. An autoencoder encodes the features into the low dimension and then decodes them as the original features for reconstructing the input patterns. Genuine dimensionality reduction which will remove insignificant/low profile features, will make it impossible to recover and reconstruct after the middle layer. So, the dimensionality reduction can be applicable for the funnel in, but it cannot be applicable for the funnel out because once lost, the features cannot be recovered. This is a simple principle similar to removing unnecessary or less significant pieces from a carving which cannot be recovered once they are removed.

However, with the functionality of autoencoders and other deep architecture models, it is evident that the features are actually getting reconstructed. In case of deep architectures, the learned features are forming more high-level ‘super features’ that resemble high-level representations which assist in classification. The clearer (more precise and accurate without noise) these high-level features are the more accurate the classifier.

6.20.1.The Blossom Effect:

The Blossom Effect can be verified through exploring the weights in the layers of autoencoders. To explore the phenomenon of The Blossom Effect, it is significant to know how weights and features are represented in autoencoder. This is followed by the necessity to observe the difference between weights extracted from the middle layer of DAE and the feature values provided by the DAE for the same layer. This comparison will procure a solid evidence on the underlying representation in the condensed weights. This experiment is performed using some randomly selected samples from the CIFAR-10 dataset and the visualisation is presented below.

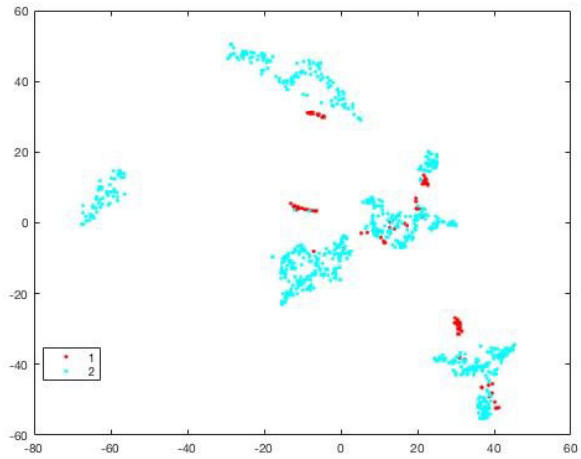


Figure 6-47: Cluster analysis of weights for projection of weights: DAE trained on CIFAR-10 dataset: middle layer components based on variance

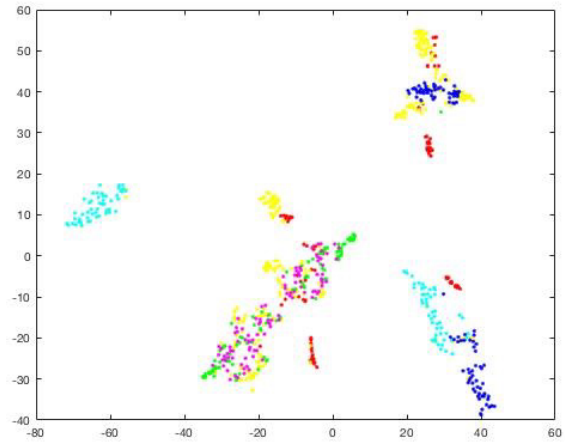


Figure 6-48: Cluster analysis of feature values with colour coding: DAE trained on CIFAR-10 dataset: middle layer components based on features in autoencoder

Figure 6-47 represents the clusters with two components from the middle layer of a 3-layered DAE. The architecture of DAE provides the values of the features attached to that layer which is presented in **Figure 6-48**. CIFAR-10 consists of a variety of images with complex overlapping features. In the **Figure 6-47,** there are two components representing the section of features extracted from the middle layer. The high-level component based representation based on variance has some inbuilt hidden representations (features) which can be observed when these weights are used to plot a feature based projections as presented in **Figure 6-48**.

In case of a simple dataset such as IRIS, similar representations can be observed. A one layer autoencoder is used for the experiment on IRIS dataset and the projection of weights and

features are presented in **Figure 6-49** and **Figure 6-50**. The representations in the **Figure 6-49** presence the variance component and the **Figure 6-50** presents the feature representation of weights.

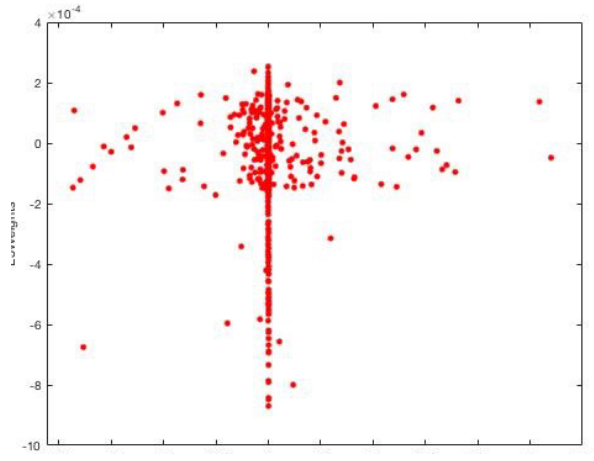


Figure 6-49: Cluster analysis of weights for projection of weights: DAE trained on IRIS dataset: middle layer components based on variance

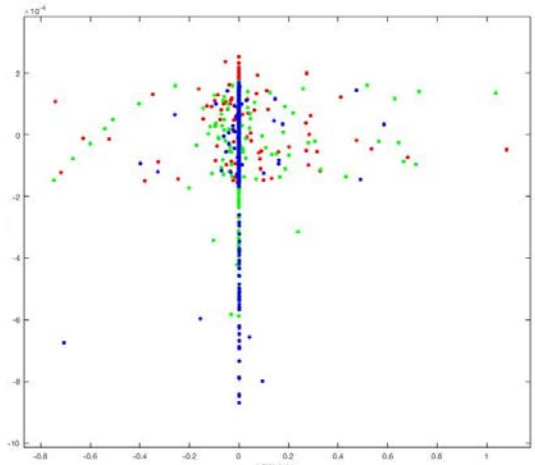


Figure 6-50: Cluster analysis of feature values with colour coding: DAE trained on IRIS dataset: middle layer components based on features in autoencoder

The conclusions on importance of the middle layer and the process of features getting folded into one component are not based on assumptions but on the evaluation of results from various experiments presented in this thesis. The proposed Blossom Effect may challenge the traditional definition of the autoencoder to some extent, however, the autoencoders can be considered as a special case of the proposed Blossom Effect. This aspect definitely needs and warrants further investigation with a view of exploring weight patterns in the hidden layers.

6.21. Chapter Summary

This chapter presents the implementation and evaluation of the transferable knowledge component model proposed in Chapter 5. The key outcomes of this chapter can be summarised as follows:

- The number and the diversity of the datasets used for the experiments provide assurance on the adoptability of the model and validation of the hypothesis.

- The potential for technological bias was tested by implementing the model on a variety of deep architectures using various hardware and software setups.
- The results obtained from the classification experiment presented in this chapter provides evidence for the extracted knowledge component model.
- The experimental results show that the number of components extracted from the layers of a DNN follows The Blossom Effect in principle. At the layers closet to the input, the number of components is greater, and these components become reduced towards the middle layer as the features are conceptually folded into the middle layer. As the features pass through the middle layer, the number of components increases as the features are transformed into high-level and problem specific features.

The hypotheses proposed in this thesis which drove the design of the experiments and selection of datasets are thus validated (and definitively proven, at least in the context of the experiments presented herein). The next chapter presents the final conclusion and suggested directions for future work.

Chapter 7 Conclusions & Prospects

The opaque nature of neural networks inspired this research quest to find answers to some of the fundamental questions about neural network learning and the knowledge that neural networks possess. While exploring the basic concepts of ANNs and DNNs in order to develop a complete understanding of the technicalities of neural networks, my intuition directed me towards the idea of the Blossom Effect. This thesis validated the proposed Blossom Effect hypothesis through systematic research and provided evidence that allowed comparison of the process of neural network learning with the postulated Blossom Effect.

Initial experiment results provided enough confidence to continue the research. The preliminary investigation (Chapter 3) provided the answers to my initial research questions on the importance of neural network layers and the existence of transferable knowledge. The proposed Transferable Knowledge Component model is shown to be efficient through a stringent experimental evaluation on different types of dataset and deep architectures.

For any future research, it is important to start with the dataset with scalable and independent features which will provide a clear insight on representations in the neural network weights.

7.1. Key Contributions

- The Blossom Effect provides an insight into the working principle of widely used neural networks such as autoencoders.
- This research provides a new direction for the future researchers guiding them towards creating generalised neural network models that are transferable.
- This thesis provides an insight into the operation of deep learning and how neural networks are able to learn efficiently.
- The relationship established in this thesis between input features and neural network weights provides a new approach in neural network optimisation through transferring a knowledge model rather than traditional transfer learning of copying or moving the layers altogether. This will help to implement a portable deep learning model and transferable models for applications.
- A transferable knowledge component model which provides a wide range of implementations for various industrial applications.

- DNN optimisation through reducing layers and providing a separate deployable DNN model will be of research interest.
- The Blossom Effect may be a challenging concept for orthodox neural network researchers but might encourage other less traditional neural network researchers to explore and possibly define the nature of this effect further.
- It is anticipated that this research will have some impact on how transfer learning is implemented in future research and development.

7.2. Research Limitations

There are a number of limitations related to the research presented in this thesis. These are:

- There is no known work on generalising neural network models for multiple problems. While this thesis may not be generalised to all datasets, in spite of using different types of datasets and deep architecture, it is a step towards demonstrating the presence of a generic core knowledge that exists in the middle layer. This would help to work towards a generalised neural network model based on underlying common knowledge that exists in the form of representations.
- Due to their unique nature and peculiar implementation, Convolutional Neural Networks (CNNs) are tested with only one dataset and thus the results are limited to image analysis presented in this thesis and a further exploration is required for proposing a transferable model for CNNs.
- Each deep architecture is primarily used for a particular type of problem. For instance, CNNs are used for image recognition and autoencoders are used for speech and speaker classification and identification. Comprehensive research is required to test all cross domain across all deep architectures (CNNs, DBF and other models) to draw fully generalised conclusions. Some experiments are performed to address this issue but due to limitations of PhD study, all deep architectures are not covered.
- All the datasets used for the experiments are static and the impact for dynamic and temporal or streaming datasets is out of scope of this thesis.
- Hybrid deep architectures (unconventional) are not tested hence the implementation implications are unknown.

7.3. Future Work

There are a number of opportunities for future work:

- Convolutional Neural Network based implementations on a wide range of datasets.
- Assessing the capability of the Transferable Knowledge Component model for temporal data.
- Industrial level implementation of a separate training and testing model for deep neural networks.
- Evaluation and implementation of the knowledge component transfer model for offline and mobile neural network based on Transferable Knowledge Component Model.

BIBILIOGRAPHY

[1] C. Percy, A. S. d. A. Garcez, S. Dragičević, M. V. França, G. Slabaugh, and T. Weyde, "The need for knowledge extraction: understanding harmful gambling behavior with neural networks." pp. 974-981.

[2] G. E. Hinton, and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *science*, vol. 313, no. 5786, pp. 504-507, 2006.

[3] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436, 2015.

[4] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929-1958, 2014.

[5] S. S. Tirumala, and A. Narayanan, "Transpositional neurocryptography using deep learning." pp. 330-334.

[6] M. van Gerven, and S. Bohte, *Artificial neural networks as models of neural information processing*: Frontiers Media SA, 2018.

[7] W. S. McCulloch, and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The bulletin of mathematical biophysics*, vol. 5, no. 4, pp. 115-133, 1943.

[8] P. J. Werbos, *The roots of backpropagation: from ordered derivatives to neural networks and political forecasting*: John Wiley & Sons, 1994.

[9] P. Werbos, "Beyond Regression:" New Tools for Prediction and Analysis in the Behavioral Sciences," *Ph. D. dissertation, Harvard University*, 1974.

[10] D. E. Rumelhart, and D. A. Norman, "Active Semantic Networks as a Model of Human Memory." pp. 450-457.

[11] D. E. Rumelhart, J. L. McClelland, and P. R. Group, *Parallel distributed processing*: MIT press Cambridge, MA, 1987.

[12] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, *Learning internal representations by error propagation*, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[13] D. Sorvisto, "THREATS AND CHALLENGES IN MACHINE LEARNING," 2018.

[14] G. Marcus, "Deep learning: A critical appraisal," *arXiv preprint arXiv:1801.00631*, 2018.

[15] Y. Bengio, "Learning deep architectures for AI," *Foundations and trends® in Machine Learning*, vol. 2, no. 1, pp. 1-127, 2009.

[16] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: A review and new perspectives," *IEEE transactions on pattern analysis and machine intelligence*, vol. 35, no. 8, pp. 1798-1828, 2013.

[17] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural networks*, vol. 61, pp. 85-117, 2015.

[18] M. Moreira, and E. Fiesler, *Neural networks with adaptive learning rate and momentum terms*, Idiap, 1995.

[19] L. N. Smith, "A disciplined approach to neural network hyper-parameters: Part 1--learning rate, batch size, momentum, and weight decay," *arXiv preprint arXiv:1803.09820*, 2018.

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

[21] K. Fukushima, "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position," *Biological cybernetics*, vol. 36, no. 4, pp. 193-202, 1980.

[22] Y. LeCun, B. E. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. E. Hubbard, and L. D. Jackel, "Handwritten digit recognition with a back-propagation network." pp. 396-404.
[23] L. v. d. Maaten, and G. Hinton, "Visualizing data using t-SNE," *Journal of machine learning research*, vol. 9, no. Nov, pp. 2579-2605, 2008.
[24] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278-2324, 1998.
[25] K. Fukushima, and S. Miyake, "Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition," *Competition and cooperation in neural nets*, pp. 267-285: Springer, 1982.
[26] S. Hochreiter, and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735-1780, 1997.
[27] G. E. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets," *Neural computation*, vol. 18, no. 7, pp. 1527-1554, 2006.
[28] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle, "Greedy layer-wise training of deep networks." pp. 153-160.
[29] S. S. Tirumala, and A. Narayanan, "Hierarchical data classification using deep neural networks." pp. 492-500.
[30] J. Ma, M. K. Yu, S. Fong, K. Ono, E. Sage, B. Demchak, R. Sharan, and T. Ideker, "Using deep learning to model the hierarchical structure and function of a cell," *Nature methods*, vol. 15, no. 4, pp. 290, 2018.
[31] S.-A. Rebuffi, H. Bilen, and A. Vedaldi, "Efficient parametrization of multi-domain deep neural networks." pp. 8119-8127.
[32] J. M. Torres, C. I. Comesaña, and P. J. García-Nieto, "machine learning techniques applied to cybersecurity," *International Journal of Machine Learning and Cybernetics*, pp. 1-14, 2019.
[33] R. Pizarro, H.-E. Assemlal, D. De Nigris, C. Elliott, S. Antel, D. Arnold, and A. Shmuel, "Using deep learning algorithms to automatically identify the brain MRI contrast: implications for managing large databases," *Neuroinformatics*, vol. 17, no. 1, pp. 115-130, 2019.
[34] R. Anderson, H. Li, Y. Ji, P. Liu, and M. L. Giger, "Evaluating deep learning techniques for dynamic contrast-enhanced MRI in the diagnosis of breast cancer." p. 1095006.
[35] Z. Tayeb, J. Fedjaev, N. Ghaboosi, C. Richter, L. Everding, X. Qu, Y. Wu, G. Cheng, and J. Conradt, "Validating deep neural networks for online decoding of motor imagery movements from EEG signals," *Sensors*, vol. 19, no. 1, pp. 210, 2019.
[36] Z.-Q. Wang, X. Zhang, and D. Wang, "Robust Speaker Localization Guided by Deep Learning-Based Time-Frequency Masking," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 27, no. 1, pp. 178-188, 2019.
[37] A. Chatterjee, U. Gupta, M. K. Chinnakotla, R. Srikanth, M. Galley, and P. Agrawal, "Understanding Emotions in Text Using Deep Learning and Big Data," *Computers in Human Behavior*, vol. 93, pp. 309-317, 2019.
[38] S. Ahmed, M. Islam, J. Hassan, M. U. Ahmed, B. J. Ferdosi, S. Saha, and M. Shopon, "Hand Sign to Bangla Speech: A Deep Learning in Vision based system for Recognizing Hand Sign Digits and Generating Bangla Speech," *arXiv preprint arXiv:1901.05613*, 2019.
[39] A. Thangthai, B. Milner, and S. Taylor, "Synthesising visual speech using dynamic visemes and deep learning architectures," *Computer Speech & Language*, vol. 55, pp. 101-119, 2019.

[40] H. Yao, S. Zhang, R. Hong, Y. Zhang, C. Xu, and Q. Tian, “Deep representation learning with part loss for person re-identification,” *IEEE Transactions on Image Processing*, 2019.
[41] R. Zhao, R. Yan, Z. Chen, K. Mao, P. Wang, and R. X. Gao, “Deep learning and its applications to machine health monitoring,” *Mechanical Systems and Signal Processing*, vol. 115, pp. 213-237, 2019.
[42] X. Yuan, P. He, Q. Zhu, and X. Li, “Adversarial examples: Attacks and defenses for deep learning,” *IEEE transactions on neural networks and learning systems*, 2019.
[43] T. Falk, D. Mai, R. Bensch, Ö. Çiçek, A. Abdulkadir, Y. Marrakchi, A. Böhm, J. Deubner, Z. Jäckel, and K. Seiwald, “U-Net: deep learning for cell counting, detection, and morphometry,” *Nature methods*, vol. 16, no. 1, pp. 67, 2019.
[44] S. Zhang, L. Yao, A. Sun, and Y. Tay, “Deep learning based recommender system: A survey and new perspectives,” *ACM Computing Surveys (CSUR)*, vol. 52, no. 1, pp. 5, 2019.
[45] S. S. Tirumala, “DEEP LEARNING USING UNCONVENTIONAL PARADIGMS,” *International Journal of Computer Research*, vol. 23, no. 3, pp. 295, 2016.
[46] A. Abdullah, R. C. Veltkamp, and M. A. Wiering, "An ensemble of deep support vector machines for image categorization." pp. 301-306.
[47] Y. Tang, “Deep learning using linear support vector machines,” *arXiv preprint arXiv:1306.0239*, 2013.
[48] Y. Cho, and L. K. Saul, "Kernel methods for deep learning." pp. 342-350.
[49] J. H. Lee, T. Delbruck, and M. Pfeiffer, “Training deep spiking neural networks using backpropagation,” *Frontiers in neuroscience*, vol. 10, pp. 508, 2016.
[50] J. Lamos-Sweeney, “Deep learning using genetic algorithms,” 2012.
[51] R. Miikkulainen, J. Liang, E. Meyerson, A. Rawal, D. Fink, O. Francon, B. Raju, H. Shahrzad, A. Navruzyan, and N. Duffy, "Evolving deep neural networks," *Artificial Intelligence in the Age of Neural Networks and Brain Computing*, pp. 293-312: Elsevier, 2019.
[52] S. S. Tirumala, S. Ali, and C. P. Ramesh, "Evolving deep neural networks: A new prospect." pp. 69-74.
[53] S. J. Pan, and Q. Yang, “A survey on transfer learning,” *IEEE Transactions on knowledge and data engineering*, vol. 22, no. 10, pp. 1345-1359, 2009.
[54] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, "How transferable are features in deep neural networks?." pp. 3320-3328.
[55] L. Torrey, and J. Shavlik, "Transfer learning," *Handbook of research on machine learning applications and trends: algorithms, methods, and techniques*, pp. 242-264: IGI Global, 2010.
[56] S. S. Tirumala, "A Deep Autoencoder-Based Knowledge Transfer Approach." pp. 277-284.
[57] M. Long, Y. Cao, J. Wang, and M. I. Jordan, “Learning transferable features with deep adaptation networks,” *arXiv preprint arXiv:1502.02791*, 2015.
[58] S. Gutstein, O. Fuentes, and E. Freudenthal, “Knowledge transfer in deep convolutional neural nets,” *International Journal on Artificial Intelligence Tools*, vol. 17, no. 03, pp. 555-567, 2008.
[59] S. S. Tirumala, “Evolving deep neural networks using coevolutionary algorithms with multi-population strategy,” *Neural Computing and Applications*, pp. 1-14, 2020.
[60] X. Glorot, and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks." pp. 249-256.
[61] S. Wang, W. Liu, J. Wu, L. Cao, Q. Meng, and P. J. Kennedy, "Training deep neural networks on imbalanced data sets." pp. 4368-4374.

[62] S. Latif, R. Rana, S. Younis, J. Qadir, and J. Epps, “Transfer learning for improving speech emotion classification accuracy,” *arXiv preprint arXiv:1801.06353*, 2018.
[63] D. Han, Q. Liu, and W. Fan, “A new image classification method using CNN transfer learning and web data augmentation,” *Expert Systems with Applications,* vol. 95, pp. 43-56, 2018.
[64] D. C. Cireşan, U. Meier, and J. Schmidhuber, "Transfer learning for Latin and Chinese characters with deep neural networks." pp. 1-6.
[65] D. K. Milligan, and M. J. D. Wilson, "Fundamental Structure/Behaviour Relationships in Synchronous Boolean Neural Networks." pp. 997-1000.
[66] L. Chen, and J. Gasteiger, “Knowledge discovery in reaction databases: Landscaping organic reactions by a self-organizing neural network,” *Journal of the American Chemical Society,* vol. 119, no. 17, pp. 4033-4042, 1997.
[67] K. J. Cios, W. Pedrycz, and R. W. Swiniarski, "Data mining and knowledge discovery," *Data mining methods for knowledge discovery*, pp. 1-26: Springer, 1998.
[68] R. Brause, T. Langsdorf, and M. Hepp, "Neural data mining for credit card fraud detection." pp. 103-106.
[69] L. Fu, “Knowledge discovery based on neural networks,” *Communications of the ACM,* vol. 42, no. 11, pp. 47-47, 1999.
[70] K. J. McGarry, S. Wermter, and J. MacIntyre, "Knowledge extraction from radial basis function networks and multilayer perceptrons." pp. 2494-2497.
[71] Z. Waszczyszyn, *Neural networks in the analysis and design of structures*: Springer, 1999.
[72] B. Arguello, “A survey of feature selection methods: algorithms and software,” 2015.
[73] S. Tirumala, and A. Narayanan, “Classification and diagnostic prediction of prostate cancer using gene expression and artificial neural networks,” *Neural Computing and Applications*, pp. 1-10, 2018.
[74] G. Chandrashekar, and F. Sahin, “A survey on feature selection methods,” *Computers & Electrical Engineering*, vol. 40, no. 1, pp. 16-28, 2014.
[75] S. Kamruzzaman, and M. M. Islam, “Extraction of Symbolic Rules from Artificial Neural Networks.”
[76] K. Hornik, M. Stinchcombe, and H. White, “Multilayer feedforward networks are universal approximators,” *Neural networks,* vol. 2, no. 5, pp. 359-366, 1989.
[77] R. Geirhos, C. R. Temme, J. Rauber, H. H. Schütt, M. Bethge, and F. A. Wichmann, "Generalisation in humans and deep neural networks." pp. 7538-7550.
[78] S. S. Tirumala, S. R. Shahamiri, A. S. Garhwal, and R. Wang, “Speaker identification features extraction methods: A systematic review,” *Expert Systems with Applications,* vol. 90, pp. 250-271, 2017.
[79] I. Guyon, S. Gunn, M. Nikravesh, and L. A. Zadeh, *Feature extraction: foundations and applications*: Springer, 2008.
[80] C. M. Bishop, *Pattern recognition and machine learning*: springer, 2006.
[81] P. Trinidad, D. Benavides, and A. R. Cortés, "Isolated Features Detection in Feature Models."
[82] H. Motoda, and H. Liu, “Feature selection, extraction and construction,” *Communication of IICM (Institute of Information and Computing Machinery, Taiwan) Vol*, vol. 5, pp. 67-72, 2002.
[83] A. Senawi, H.-L. Wei, and S. A. Billings, “A new maximum relevance-minimum multicollinearity (MRmMC) method for feature selection and ranking,” *Pattern Recognition,* vol. 67, pp. 47-61, 2017.

[84] L. Yu, and H. Liu, "Efficient feature selection via analysis of relevance and redundancy," *Journal of machine learning research,* vol. 5, no. Oct, pp. 1205-1224, 2004.
[85] I. Guyon, and A. Elisseeff, "An introduction to variable and feature selection," *Journal of machine learning research,* vol. 3, no. Mar, pp. 1157-1182, 2003.
[86] S. Jayanthi, and C. R. Robin, "A survey on different classification methods for microarray data analysis," *Advances in Environmental Biology,* vol. 11, no. 5, pp. 13-16, 2017.
[87] S. Adams, and P. A. Beling, "A survey of feature selection methods for Gaussian mixture models and hidden Markov models," *Artificial Intelligence Review,* pp. 1-41, 2017.
[88] Y. Yoo, L. Y. Tang, T. Brosch, D. K. Li, S. Kolind, I. Vavasour, A. Rauscher, A. L. MacKay, A. Traboulsee, and R. C. Tam, "Deep learning of joint myelin and T1w MRI features in normal-appearing brain tissue to distinguish between multiple sclerosis patients and healthy controls," *NeuroImage: Clinical,* vol. 17, pp. 169-178, 2018.
[89] K. Lillywhite, D.-J. Lee, B. Tippetts, and J. Archibald, "A feature construction method for general object recognition," *Pattern Recognition,* vol. 46, no. 12, pp. 3300-3314, 2013.
[90] B. Tran, S. Picek, and B. Xue, "Automatic Feature Construction for Network Intrusion Detection." pp. 569-580.
[91] K. Li, C. Zou, S. Bu, Y. Liang, J. Zhang, and M. Gong, "Multi-modal feature fusion for geographic image annotation," *Pattern Recognition,* vol. 73, pp. 1-14, 2018.
[92] J. Seidlitz, F. Váša, M. Shinn, R. Romero-Garcia, K. J. Whitaker, P. E. Vértes, K. Wagstyl, P. K. Reardon, L. Clasen, and S. Liu, "Morphometric Similarity Networks Detect Microscale Cortical Organization and Predict Inter-Individual Cognitive Variation," *Neuron,* vol. 97, no. 1, pp. 231-247. e7, 2018.
[93] X. Yu, J.-y. Lin, F. Jiang, J.-w. Du, and J.-z. Han, "A Cross-Domain Collaborative Filtering Algorithm Based on Feature Construction and Locally Weighted Linear Regression," *Computational Intelligence and Neuroscience,* vol. 2018, 2018.
[94] D. Koller, and M. Sahami, *Toward optimal feature selection*, Stanford InfoLab, 1996.
[95] X. Zhu, X. Li, S. Zhang, C. Ju, and X. Wu, "Robust joint graph sparse coding for unsupervised spectral feature selection," *IEEE transactions on neural networks and learning systems,* vol. 28, no. 6, pp. 1263-1275, 2017.
[96] J. Li, K. Cheng, S. Wang, F. Morstatter, R. P. Trevino, J. Tang, and H. Liu, "Feature selection: A data perspective," *ACM Computing Surveys (CSUR),* vol. 50, no. 6, pp. 94, 2017.
[97] B. Xue, M. Zhang, and W. N. Browne, "A comprehensive comparison on evolutionary feature selection approaches to classification," *International Journal of Computational Intelligence and Applications,* vol. 14, no. 02, pp. 1550008, 2015.
[98] B. Xue, M. Zhang, W. N. Browne, and X. Yao, "A survey on evolutionary computation approaches to feature selection," *IEEE Transactions on Evolutionary Computation,* vol. 20, no. 4, pp. 606-626, 2016.
[99] L. Wang, Y. Wang, and Q. Chang, "Feature selection methods for big data bioinformatics: A survey from the search perspective," *Methods,* vol. 111, pp. 21-31, 2016.
[100] J. Miao, and L. Niu, "A Survey on Feature Selection," *Procedia Computer Science,* vol. 91, pp. 919-926, 2016.
[101] O. Osanaiye, H. Cai, K.-K. R. Choo, A. Dehghantanha, Z. Xu, and M. Dlodlo, "Ensemble-based multi-filter feature selection method for DDoS detection in cloud

computing," *EURASIP Journal on Wireless Communications and Networking*, vol. 2016, no. 1, pp. 130, 2016.
[102] F. Bellal, H. Elghazel, and A. Aussem, "A semi-supervised feature ranking method with ensemble learning," *Pattern Recognition Letters*, vol. 33, no. 10, pp. 1426-1433, 2012.
[103] L. Talavera, "An evaluation of filter and wrapper methods for feature selection in categorical clustering." pp. 440-451.
[104] D. W. Aha, and R. L. Bankert, "A comparative evaluation of sequential feature selection algorithms," *Learning from data*, pp. 199-206: Springer, 1996.
[105] H. Peng, F. Long, and C. Ding, "Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 27, no. 8, pp. 1226-1238, 2005.
[106] V. Rodriguez-Galiano, J. Luque-Espinar, M. Chica-Olmo, and M. Mendes, "Feature selection approaches for predictive modelling of groundwater nitrate pollution: An evaluation of filters, embedded and wrapper methods," *Science of The Total Environment*, vol. 624, pp. 661-672, 2018.
[107] C. Lazar, J. Taminau, S. Meganck, D. Steenhoff, A. Coletta, C. Molter, V. de Schaetzen, R. Duque, H. Bersini, and A. Nowe, "A survey on filter techniques for feature selection in gene expression microarray analysis," *IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB)*, vol. 9, no. 4, pp. 1106-1119, 2012.
[108] T. Luo, C. Hou, F. Nie, H. Tao, and D. Yi, "Semi-supervised Feature Selection via Insensitive Sparse Regression with Application to Video Semantic Recognition," *IEEE Transactions on Knowledge and Data Engineering*, 2018.
[109] P. Boonthong, P. Kulkasem, S. Rasmequan, A. Rodtook, and K. Chinnasarn, "Fisher feature selection for emotion recognition." pp. 1-6.
[110] W. Malina, "Some multiclass Fisher feature selection algorithms and their comparison with Karhunen-Loeve algorithms," *Pattern Recognition Letters*, vol. 6, no. 5, pp. 279-285, 1987.
[111] G. Doquire, and M. Verleysen, "A graph Laplacian based approach to semi-supervised feature selection for regression problems," *Neurocomputing*, vol. 121, pp. 5-13, 2013.
[112] J. Ren, Z. Qiu, W. Fan, H. Cheng, and S. Y. Philip, "Forward semi-supervised feature selection." pp. 970-976.
[113] Y. Liu, F. Nie, J. Wu, and L. Chen, "Semi-supervised feature selection based on label propagation and subset selection." pp. 293-296.
[114] Y. Han, Y. Yang, Y. Yan, Z. Ma, N. Sebe, and X. Zhou, "Semisupervised feature selection via spline regression for video semantic recognition," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 26, no. 2, pp. 252-264, 2015.
[115] Z. Ma, F. Nie, Y. Yang, J. R. Uijlings, N. Sebe, and A. G. Hauptmann, "Discriminating joint feature analysis for multimedia data understanding," *IEEE Transactions on Multimedia*, vol. 14, no. 6, pp. 1662-1672, 2012.
[116] C. Shi, Q. Ruan, and G. An, "Sparse feature selection based on graph Laplacian for web image annotation," *Image and Vision Computing*, vol. 32, no. 3, pp. 189-201, 2014.
[117] K. Dai, H.-Y. Yu, and Q. Li, "A semisupervised feature selection with support vector machine," *Journal of Applied Mathematics*, vol. 2013, 2013.
[118] S. Khalid, T. Khalil, and S. Nasreen, "A survey of feature selection and feature extraction techniques in machine learning." pp. 372-378.
[119] H. A. Le Thi, H. M. Le, and T. P. Dinh, "Feature selection in machine learning: an exact penalty approach using a difference of convex function algorithm," *Machine Learning*, vol. 101, no. 1-3, pp. 163-186, 2015.

[120] M. Long, Y. Cao, J. Wang, and M. Jordan, "Learning transferable features with deep adaptation networks." pp. 97-105.
[121] H. Zhao, X. Guo, M. Wang, T. Li, C. Pang, and D. Georgakopoulos, “Analyze EEG signals with extreme learning machine based on PMIS feature selection,” *International Journal of Machine Learning and Cybernetics*, vol. 9, no. 2, pp. 243-249, 2018.
[122] A. L. Blum, and P. Langley, “Selection of relevant features and examples in machine learning,” *Artificial intelligence*, vol. 97, no. 1-2, pp. 245-271, 1997.
[123] I. T. Jolliffe, "Principal component analysis and factor analysis," *Principal component analysis*, pp. 115-128: Springer, 1986.
[124] S. Wold, K. Esbensen, and P. Geladi, “Principal component analysis,” *Chemometrics and intelligent laboratory systems*, vol. 2, no. 1-3, pp. 37-52, 1987.
[125] M. Ringnér, “What is principal component analysis?,” *Nature biotechnology*, vol. 26, no. 3, pp. 303, 2008.
[126] B. Moore, “Principal component analysis in linear systems: Controllability, observability, and model reduction,” *IEEE transactions on automatic control*, vol. 26, no. 1, pp. 17-32, 1981.
[127] I. T. Jolliffe, and J. Cadima, “Principal component analysis: a review and recent developments,” *Phil. Trans. R. Soc. A*, vol. 374, no. 2065, pp. 20150202, 2016.
[128] C. Happ, and S. Greven, “Multivariate functional principal component analysis for data observed on different (dimensional) domains,” *Journal of the American Statistical Association*, pp. 1-11, 2018.
[129] T. Niedoba, “Multi-parameter data visualization by means of principal component analysis (PCA) in qualitative evaluation of various coal types,” *physicochemical problems of Mineral processing*, vol. 50, 2014.
[130] T. Sander, J. Freyss, M. von Korff, and C. Rufener, “DataWarrior: an open-source program for chemistry aware data visualization and analysis,” *Journal of chemical information and modeling*, vol. 55, no. 2, pp. 460-473, 2015.
[131] T. Metsalu, and J. Vilo, “ClustVis: a web tool for visualizing clustering of multivariate data using Principal Component Analysis and heatmap,” *Nucleic acids research*, vol. 43, no. W1, pp. W566-W570, 2015.
[132] E. Mooi, M. Sarstedt, and I. Mooi-Reci, "Principal Component and Factor Analysis," *Market Research*, pp. 265-311: Springer, 2018.
[133] D. Chawla, and M. C. Trivedi, "A Comparative Study on Face Detection Techniques for Security Surveillance," *Advances in Computer and Computational Sciences*, pp. 531-541: Springer, 2018.
[134] C. Callegari, L. Donatini, S. Giordano, and M. Pagano, “Improving stability of PCA-based network anomaly detection by means of kernel-PCA,” *International Journal of Computational Science and Engineering*, vol. 16, no. 1, pp. 9-16, 2018.
[135] A. Khamparia, and B. Pandey, “SVM and PCA Based Learning Feature Classification Approaches for E-Learning System,” *International Journal of Web-Based Learning and Teaching Technologies (IJWLTT)*, vol. 13, no. 2, pp. 32-45, 2018.
[136] G. Kong, L. Jiang, X. Yin, T. Wang, D.-L. Xu, J.-B. Yang, and Y. Hu, “Combining principal component analysis and the evidential reasoning approach for healthcare quality assessment,” *Annals of Operations Research*, pp. 1-21, 2018.
[137] M. Zimmermann, M. M. Ghazi, H. K. Ekenel, and J.-P. Thiran, "Visual speech recognition using PCA networks and LSTMs in a tandem GMM-HMM system." pp. 264-276.
[138] C. Beckett, L. Eriksson, E. Johansson, and C. Wikström, “Multivariate Data Analysis (MVDA),” *Pharmaceutical Quality by Design: A Practical Approach*, pp. 201, 2018.

[139] P. J. Cumpson, N. Sano, I. W. Fletcher, J. F. Portoles, M. Bravo-Sanchez, and A. J. Barlow, "Multivariate analysis of extremely large ToFSIMS imaging datasets by a rapid PCA method," *Surface and Interface Analysis,* vol. 47, no. 10, pp. 986-993, 2015.
[140] Z. Quan, J. Ning, K. Englehart, and B. Hudgins, "Improved Phoneme-Based Myoelectric Speech Recognition," *CMBES Proceedings,* vol. 31, no. 1, 2017.
[141] I. Eide, and F. Westad, "Automated multivariate analysis of multi-sensor data submitted online: Real-time environmental monitoring," *PloS one,* vol. 13, no. 1, pp. e0189443, 2018.
[142] K. Wang, N. Li, L. Bagas, S. Li, X. Song, and Y. Cong, "GIS-based prospectivity-mapping based on geochemical multivariate analysis technology: A case study of MVT Pb–Zn deposits in the Huanyuan-Fenghuang district, northwestern Hunan Province, China," *Ore Geology Reviews,* vol. 91, pp. 1130-1146, 2017.
[143] V. Sharma, K. Yousefi, Z. Haddad, C. Buerki, R. B. Jenkins, E. Davicioni, and R. J. Karnes, "Gene Expression Correlates of Site-specific Metastasis Among Men With Lymph Node Positive Prostate Cancer Treated With Radical Prostatectomy: A Case Series," *Urology,* vol. 112, pp. 29-32, 2018.
[144] G. Morrison, N. Jojo, A. Cunha, Y. Xu, P. S. Robinson, T. B. Dorff, D. I. Quinn, and A. Goldkorn, "Novel method for rapid enrichment of high purity circulating tumor cells (CTCs) for prostate cancer (PCa) gene expression profiling," American Society of Clinical Oncology, 2018.
[145] F. A. P. Peres, and F. S. Fogliatto, "Variable selection methods in multivariate statistical process control: A systematic literature review," *Computers & Industrial Engineering,* vol. 115, pp. 603-619, 2018.
[146] I. T. Jolliffe, "Discarding variables in a principal component analysis. I: Artificial data," *Applied statistics,* pp. 160-173, 1972.
[147] D. J. Bartholomew, F. Steele, J. Galbraith, and I. Moustaki, *Analysis of multivariate social science data*: Chapman and Hall/CRC, 2008.
[148] D. George, and P. Mallery, *IBM SPSS statistics 23 step by step: A simple guide and reference*: Routledge, 2016.
[149] J. J. Hox, M. Moerbeek, and R. Van de Schoot, *Multilevel analysis: Techniques and applications*: Routledge, 2017.
[150] R. P. McDonald, *Factor analysis and related methods*: Psychology Press, 2014.
[151] D. Brown, "Statistics Corner, Questions and answers about language testing statistics: Can we use the Spearman-Brown prophecy formula to defend low reliability," *Shiken: JALT Testing & Evaluation SIG Newsletter,* vol. 4, no. 3, pp. 7-9, 2009.
[152] B. G. Tabachnick, and L. S. Fidell, *Using multivariate statistics*: Allyn & Bacon/Pearson Education, 2007.
[153] S. Corner, "Choosing the right type of rotation in PCA and EFA," *JALT testing & evaluation SIG newsletter,* vol. 13, no. 3, pp. 20-25, 2009.
[154] J.-O. Kim, and C. W. Mueller, *Introduction to factor analysis: What it is and how to do it*: Sage, 1978.
[155] J. Mao, and A. K. Jain, "Artificial neural networks for feature extraction and multivariate data projection," *IEEE transactions on neural networks,* vol. 6, no. 2, pp. 296-317, 1995.
[156] R. P. Gorman, and T. J. Sejnowski, "Analysis of hidden units in a layered network trained to classify sonar targets," *Neural networks,* vol. 1, no. 1, pp. 75-89, 1988.
[157] J. Rubner, and P. Tavan, "A self-organizing network for principal-component analysis," *EPL (Europhysics Letters),* vol. 10, no. 7, pp. 693, 1989.
[158] J. Mao, and A. Jain, "Discriminant analysis neural networks." pp. 300-305.

[159] S. Kung, and K. Diamantaras, "A neural network learning algorithm for adaptive principal component extraction (APEX)." pp. 861-864.
[160] K. Hornik, and C.-M. Kuan, “Convergence analysis of local feature extraction algorithms,” *Neural Networks,* vol. 5, no. 2, pp. 229-240, 1992.
[161] P. Földiák, and P. Fdilr, “Adaptive network for optimal linear feature extraction,” 1989.
[162] P. Baldi, and K. Hornik, “Neural networks and principal component analysis: Learning from examples without local minima,” *Neural networks,* vol. 2, no. 1, pp. 53-58, 1989.
[163] H. M. Abbas, and M. M. Fahmy, "A neural model for adaptive Karhunen Loeve transformation (KLT)." pp. 975-980.
[164] N. Intrator, "A neural network for feature extraction." pp. 719-726.
[165] R. Setiono, and H. Liu, "Feature extraction via neural networks," *Feature Extraction, Construction and Selection*, pp. 191-204: Springer, 1998.
[166] E. Oja, “Simplified neuron model as a principal component analyzer,” *Journal of mathematical biology,* vol. 15, no. 3, pp. 267-273, 1982.
[167] P. Gallinari, S. Thiria, F. Badran, and F. Fogelman-Soulie, “On the relations between discriminant analysis and multilayer perceptrons,” *neural networks,* vol. 4, no. 3, pp. 349-360, 1991.
[168] G. W. Cottrell, "Extracting features from faces using compression networks: Face, identity, emotion, and gender recognition using holons," *Connectionist Models*, pp. 328-337: Elsevier, 1991.
[169] T. L. Kohonen, P., Oja, E., Kortekangas, A., & , and K. Makisara, “Demons!ration of pattern processing
properties of the optimal associative mappings,” in International Conf. on Cybernetics and Society, Washington D C, 1977.
[170] S. Becker, and M. Plumbley, “Unsupervised neural network learning procedures for feature extraction and classification,” *Applied Intelligence,* vol. 6, no. 3, pp. 185-203, 1996.
[171] S. S. Tirumala, and S. R. Shahamiri, "A Deep autoencoder approach for Speaker Identification." pp. 175-179.
[172] M. A. Kramer, “Nonlinear principal component analysis using autoassociative neural networks,” *AIChE journal,* vol. 37, no. 2, pp. 233-243, 1991.
[173] M. Scholz, and R. Vigário, "Nonlinear PCA: a new hierarchical approach." pp. 439-444.
[174] J. M. Ali, M. A. Hussain, M. O. Tade, and J. Zhang, “Artificial Intelligence techniques applied as estimator in chemical process systems–A literature survey,” *Expert Systems with Applications,* vol. 42, no. 14, pp. 5915-5931, 2015.
[175] S. B. Kotsiantis, I. Zaharakis, and P. Pintelas, “Supervised machine learning: A review of classification techniques,” *Emerging artificial intelligence applications in computer engineering,* vol. 160, pp. 3-24, 2007.
[176] Y. Bengio, A. C. Courville, and P. Vincent, “Unsupervised feature learning and deep learning: A review and new perspectives,” *CoRR, abs/1206.5538,* vol. 1, pp. 2012, 2012.
[177] C. Cortes, and V. Vapnik, “Support-vector networks,” *Machine learning,* vol. 20, no. 3, pp. 273-297, 1995.
[178] N. Wagarachchi, and A. Karunananda, "Optimization of multi-layer artificial neural networks using delta values of hidden layers." pp. 80-86.
[179] P. Langley, "Selection of relevant features in machine learning." pp. 245-271.
[180] Y. Bengio, and Y. LeCun, “Scaling learning algorithms towards AI,” *Large-scale kernel machines,* vol. 34, no. 5, pp. 1-41, 2007.

[181] R. Salakhutdinov, and H. Larochelle, "Efficient learning of deep Boltzmann machines." pp. 693-700.
[182] S. S. Tirumala, "Implementation of evolutionary algorithms for deep architectures."
[183] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, and T. N. Sainath, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 82-97, 2012.
[184] H. Lee, R. Grosse, R. Ranganath, and A. Y. Ng, "Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations." pp. 609-616.
[185] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks." pp. 1097-1105.
[186] S. Mehrkanoon, "Deep neural-kernel blocks," *Neural Networks*, vol. 116, pp. 46-55, 2019.
[187] L. Le, and Y. Xie, "Deep embedding kernel," *Neurocomputing*, vol. 339, pp. 292-302, 2019.
[188] Y. Lin, T. Zhang, S. Zhu, and K. Yu, "Deep coding network." pp. 1405-1413.
[189] B. Hutchinson, L. Deng, and D. Yu, "Tensor deep stacking networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 8, pp. 1944-1957, 2012.
[190] E. Fenil, G. Manogaran, G. Vivekananda, T. Thanjaivadivel, S. Jeeva, and A. Ahilan, "Real time violence detection framework for football stadium comprising of big data analysis and deep learning through bidirectional LSTM," *Computer Networks*, vol. 151, pp. 191-200, 2019.
[191] C.-Y. Low, J. Park, and A. B.-J. Teoh, "Stacking-Based Deep Neural Network: Deep Analytic Network for Pattern Classification," *IEEE transactions on cybernetics*, 2019.
[192] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, and G. Ostrovski, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529, 2015.
[193] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.
[194] R. E. Andersen, S. Madsen, A. B. K. Barlo, S. B. Johansen, M. Nør, R. S. Andersen, and S. Bøgh, "Self-learning Processes in Smart Factories: Deep Reinforcement Learning for Process Control of Robot Brine Injection."
[195] G. Jeong, and H. Y. Kim, "Improving financial trading decisions using deep q-learning: Predicting the number of shares, action strategies, and transfer learning," *Expert Systems with Applications*, vol. 117, pp. 125-138, 2019.
[196] X. Qi, Y. Luo, G. Wu, K. Boriboonsomsin, and M. Barth, "Deep reinforcement learning enabled self-learning control for energy efficient driving," *Transportation Research Part C: Emerging Technologies*, vol. 99, pp. 67-81, 2019.
[197] M. Wiering, M. Schutten, A. Millea, A. Meijster, and L. Schomaker, "Deep support vector machines for regression problems." pp. 53-54.
[198] J. Wang, K. Feng, and J. Wu, "SVM-based Deep Stacking Networks," *arXiv preprint arXiv:1902.05731*, 2019.
[199] Y. Tang, "Deep learning using support vector machines," *CoRR, abs/1306.0239*, vol. 2, 2013.
[200] M. Hasan, and T. A. Aleef, "Automatic Mass Detection in Breast Using Deep Convolutional Neural Network and SVM Classifier," *arXiv preprint arXiv:1907.04424*, 2019.

[201] S. R. Young, D. C. Rose, T. P. Karnowski, S.-H. Lim, and R. M. Patton, "Optimizing deep learning hyper-parameters through an evolutionary algorithm." p. 4.
[202] Z. Qu, S. Yuan, R. Chi, L. Chang, and L. Zhao, "Genetic Optimization Method of Pantograph and Catenary Comprehensive Monitor Status Prediction Model Based on Adadelta Deep Neural Network," *IEEE Access*, vol. 7, pp. 23210-23221, 2019.
[203] S. Shahane, N. Aluru, P. Ferreira, S. G. Kapoor, and S. P. Vanka, "Genetic Algorithm based Multi-Objective Optimization of Solidification in Die Casting using Deep Neural Network as Surrogate Model," *arXiv preprint arXiv:1901.02364*, 2019.
[204] S. Fujino, T. Hatanaka, N. Mori, and K. Matsumoto, "Evolutionary deep learning based on deep convolutional neural network for anime storyboard recognition," *Neurocomputing*, vol. 338, pp. 393-398, 2019.
[205] Y. Sun, B. Xue, M. Zhang, and G. G. Yen, "Evolving deep convolutional neural networks for image classification," *IEEE Transactions on Evolutionary Computation*, 2019.
[206] A. V. Terekhov, G. Montone, and J. K. O'Regan, "Knowledge transfer in deep block-modular neural networks." pp. 268-279.
[207] E. Y. Li, "Artificial neural networks and their business applications," *Information & Management*, vol. 27, no. 5, pp. 303-313, 1994.
[208] C. Kandaswamy, L. M. Silva, L. A. Alexandre, J. M. Santos, and J. M. de Sá, "Improving deep neural network performance by reusing features trained with transductive transference." pp. 265-272.
[209] A. Khatami, M. Babaie, H. R. Tizhoosh, A. Khosravi, T. Nguyen, and S. Nahavandi, "A sequential search-space shrinking using CNN transfer learning and a Radon projection pool for medical image retrieval," *Expert Systems with Applications*, vol. 100, pp. 224-233, 2018.
[210] S. Khan, N. Islam, Z. Jan, I. U. Din, and J. J. C. Rodrigues, "A Novel Deep Learning based Framework for the Detection and Classification of Breast Cancer Using Transfer Learning," *Pattern Recognition Letters*, 2019.
[211] A. X. Wang, C. Tran, N. Desai, D. Lobell, and S. Ermon, "Deep transfer learning for crop yield prediction with remote sensing data." p. 50.
[212] G. Wang, J. Qiao, J. Bi, W. Li, and M. Zhou, "TL-GDBN: Growing Deep Belief Network With Transfer Learning," *IEEE Transactions on Automation Science and Engineering*, vol. 16, no. 2, pp. 874-885, 2019.
[213] S. Latif, R. Rana, S. Younis, J. Qadir, and J. Epps, "Cross corpus speech emotion classification-an effective transfer learning technique," *arXiv preprint arXiv:1801.06353*, 2018.
[214] H. Okamoto, M. Suzuki, I. Higuchi, S. Ohsawa, and Y. Matsuo, "DUAL SPACE LEARNING WITH VARIATIONAL AUTOENCODERS," 2019.
[215] T. Pailla, K. J. Miller, and V. Gilja, "Autoencoders for learning template spectrograms in electrocorticographic signals," *Journal of neural engineering*, 2018.
[216] D. E. Rumelhart, and J. L. Mcclelland, "Parallel distributed processing: Explorations in the microstructure of cognition: Foundations (Parallel distributed processing)," MIT Press, August, 1986.
[217] R. R. Sokal, and F. J. Rohlf, "The comparison of dendrograms by objective methods," *Taxon*, vol. 11, no. 2, pp. 33-40, 1962.
[218] F. J. Rohlf, and D. R. Fisher, "Tests for hierarchical structure in random data sets," *Systematic Biology*, vol. 17, no. 4, pp. 407-412, 1968.
[219] R. A. Fisher, "The use of multiple measurements in taxonomic problems," *Annals of eugenics*, vol. 7, no. 2, pp. 179-188, 1936.

[220] Y. LeCun, C. Cortes, and C. Burges, "MNIST handwritten digit database, 1998," *URL http://www. research. att. com/~ yann/ocr/mnist*, 1998.
[221] A. Acero, *Acoustical and environmental robustness in automatic speech recognition*: Springer Science & Business Media, 2012.
[222] S. Young, G. Evermann, M. Gales, T. Hain, D. Kershaw, X. Liu, G. Moore, J. Odell, D. Ollason, and D. Povey, "The HTK book," *Cambridge university engineering department*, vol. 3, pp. 175, 2002.
[223] Y. LeCun, and Y. Bengio, "Convolutional networks for images, speech, and time series," *The handbook of brain theory and neural networks*, vol. 3361, no. 10, pp. 1995, 1995.
[224] E. P. Agency. "Environmental Protection Agency (EPA)," 2016; https://www.epa.gov/air-emissions-inventories/2014-national-emissions-inventory-nei-data.
[225] J. M. Morse, "Principles of mixed methods and multimethod research design," *Handbook of mixed methods in social and behavioral research*, vol. 1, pp. 189-208, 2003.
[226] M. J. Prince, and R. M. Felder, "Inductive teaching and learning methods: Definitions, comparisons, and research bases," *Journal of engineering education*, vol. 95, no. 2, pp. 123-138, 2006.
[227] G. Clifford, "Chapter 15-Blind Source Separation: Principal & Independent Component Analysis," *Course materials for hst. 582j/6.555 j/16.456 j, biomedical signal and image processing, Massachusetts Institute of Technology, Massachusetts, USA*, 2007.
[228] J. D. Olden, and D. A. Jackson, "Illuminating the "black box": a randomization approach for understanding variable contributions in artificial neural networks," *Ecological modelling*, vol. 154, no. 1-2, pp. 135-150, 2002.
[229] P. Cortez, A. Cerdeira, F. Almeida, T. Matos, and J. Reis, "Modeling wine preferences by data mining from physicochemical properties," *Decision Support Systems*, vol. 47, no. 4, pp. 547-553, 2009.
[230] M. Forina, S. Lanteri, and C. Armanino, "PARVUS-An Extendible Package for Data Exploration, Classification and Correlation, Institute of Pharmaceutical and Food Analysis and Technologies, Via Brigata Salerno, 16147 Genoa, Italy (1988)," *Av. Loss Av. O set Av. Hit-Rate*, 1991.
[231] D. Cireșan, U. Meier, and J. Schmidhuber, "Multi-column deep neural networks for image classification," *arXiv preprint arXiv:1202.2745*, 2012.
[232] V. V. Romanuke, "Training data expansion and boosting of convolutional neural networks for reducing the MNIST dataset error rate," 2016.
[233] G. Cohen, S. Afshar, J. Tapson, and A. van Schaik, "EMNIST: an extension of MNIST to handwritten letters," *arXiv preprint arXiv:1702.05373*, 2017.
[234] G. E. Hinton, "Deep belief networks," *Scholarpedia*, vol. 4, no. 5, pp. 5947, 2009.
[235] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database." pp. 248-255.
[236] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition." pp. 770-778.
[237] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, and M. Bernstein, "Imagenet large scale visual recognition challenge," *International journal of computer vision*, vol. 115, no. 3, pp. 211-252, 2015.
[238] Y. Huang, Y. Cheng, D. Chen, H. Lee, J. Ngiam, Q. V. Le, and Z. Chen, "Gpipe: Efficient training of giant neural networks using pipeline parallelism," *arXiv preprint arXiv:1811.06965*, 2018.

[239] J. Deng, A. Berg, S. Satheesh, H. Su, A. Khosla, and L. Fei-Fei, “Imagenet large scale visual recognition competition 2012 (ILSVRC2012),” *Google Scholar*, 2012.
[240] A. Krizhevsky, and G. Hinton, “Convolutional deep belief networks on cifar-10,” *Unpublished manuscript*, vol. 40, no. 7, 2010.
[241] E. D. Cubuk, B. Zoph, D. Mane, V. Vasudevan, and Q. V. Le, “Autoaugment: Learning augmentation policies from data,” *arXiv preprint arXiv:1805.09501*, 2018.
[242] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, and M. Devin, “TensorFlow: Large-scale machine learning on heterogeneous systems, 2015,” *Software available from tensorflow. org*, vol. 1, no. 2, 2015.
[243] A. Acero, “Acoustical and environmental robustness in automatic speech recognition,” Carnegie Mellon University Pittsburgh, 1990.
[244] J. S. Garofolo, L. F. Lamel, W. M. Fisher, J. G. Fiscus, and D. S. Pallett, “DARPA TIMIT acoustic-phonetic continous speech corpus CD-ROM. NIST speech disc 1-1.1,” *NASA STI/Recon technical report n*, vol. 93, 1993.
[245] J. S. Garofolo, L. F. Lamel, W. M. Fisher, J. G. Fiscus, D. S. Pallett, N. L. Dahlgren, and V. Zue, “TIMIT acoustic-phonetic continuous speech corpus,” *Linguistic data consortium*, vol. 10, no. 5, pp. 0, 1993.
[246] A. Benyassine, E. Shlomot, H.-Y. Su, D. Massaloux, C. Lamblin, and J.-P. Petit, “ITU-T Recommendation G. 729 Annex B: a silence compression scheme for use with G. 729 optimized for V. 70 digital simultaneous voice and data applications,” *IEEE Communications Magazine*, vol. 35, no. 9, pp. 64-73, 1997.
[247] Z. Ge, A. N. Iyer, S. Cheluvaraja, R. Sundaram, and A. Ganapathiraju, “Neural Network Based Speaker Classification and Verification Systems with Enhanced Features,” *arXiv*, 2017.
[248] S. S. Tirumala, and S. R. Shahamiri, "A review on Deep Learning approaches in Speaker Identification." pp. 142-147.
[249] S. S. Tirumala, and A. Narayanan, "Attribute Selection and Classification of Prostate Cancer Gene Expression Data Using Artificial Neural Networks." pp. 26-34.
[250] M. F. De Oliveira, and H. Levkowitz, “From visual data exploration to visual data mining: a survey,” *IEEE Transactions on Visualization and Computer Graphics*, vol. 9, no. 3, pp. 378-394, 2003.
[251] C. H. Yu, “Exploratory data analysis,” *Methods*, vol. 2, pp. 131-160, 1977.
[252] B. Franke, J. F. Plante, R. Roscher, E. s. A. Lee, C. Smyth, A. Hatefi, F. Chen, E. Gil, A. Schwing, and A. Selvitella, “Statistical inference, learning and models in big data,” *International Statistical Review*, vol. 84, no. 3, pp. 371-389, 2016.
[253] A. N. Gorban, B. Kégl, D. C. Wunsch, and A. Y. Zinovyev, *Principal manifolds for data visualization and dimension reduction*: Springer, 2008.
[254] C. Potts, “A case for deep learning in semantics,” *arXiv preprint arXiv:1809.03068*, 2018.
[255] S. Tirumala, N. Jamil, and M. A. Malik, "A Deep Neural Network Approach for Classification of Watermarked and Non-watermarked Images." pp. 779-784.
[256] F. Chollet, "Keras," 2015.
[257] G. E. Hinton, S. Sabour, and N. Frosst, “Matrix capsules with EM routing,” 2018.

APPENDICES

A. IRIS Dataset

A.1. Technical details

Table A-1: Technical details of various parameters used for the experiments using IRIS Dataset

Parameter	Value
Training algorithm	Stochastic gradient descent (SGD)
Number of epochs	500
Learning rate	0.32
Momentum	0.48
Softmax (training)	Back Propagation
Training and testing ratio	70:30
Resampling	Cross validation (3-fold)
Software	MATLAB 2014a, Weka 3.9.3 (custom library), .NET Library
Average number of runs	25 (weka) 40 (MATLAB & .NET)
Topology	30. Number of hidden nodes are same across all layers

A.2 Classification Results

Table A-2: Classification results for IRIS and modified IRIS datasets

Experiment No	Dataset	No. of Hidden Layers	Root Mean Squared Error	Accuracy (%)	T-Test
1	IRIS	3	2.64	78.3	0.023
2		5	1.9	79.0	0.019
3		9	0.41	84.5	0.013
4		13	5.53	27.1	0.24
5	M-IRIS1	3	0.024	98.6	0.02
6		5	0.0092	99.8	0.011
7		9	0.48	88.3	0.02
8		13	3.21	41.8	0.89
9	M-IRIS2	3	5.5	58.4	0.12
10		5	4.24	43.7	0.18
11		9	4.09	45.5	0.06
12		13	11.9	16.4	0.08
13	M-IRIS3	3	7.43	28.9	0.069
14		5	4.44	43.8	0.072
15		9	4.682	45.3	0.012
16		13	13.81	11.2	0.63

B. Wine Dataset

B.1 Technical details

Table B-1: Technical details of various parameters used for the experiments using WINE Dataset

Parameter	Value
Training algorithm	Stochastic gradient descent (SGD)
Number of epochs	500
Learning rate	0.32
Momentum	0.48
Softmax (training)	Back Propagation
Training and testing ratio	70:30
Resampling	Cross validation (3-fold)
Software	MATLAB 2014a
Average number of runs	50
Topology	30. Number of hidden nodes are same across all layers

B.2 Classification Results

Table B-2: Classification results for WINE Dataset

Experiment No	No. of Hidden Layers	Root Mean Squared Error	Accuracy (%)	T-Test
1	1	0.0012	100.0	0.029
2	3	1.29	63.2	0.037
3	9	7.94	24.0	0.022

C. MNIST Dataset

C.1 Properties of MNIST dataset

TRAINING SET LABEL FILE (train-labels-idx1-ubyte):

[offset]	[type]	[value]	[description]
0000	32 bit integer	0x00000801(2049)	magic number (MSB first)
0004	32 bit integer	60000	number of items
0008	unsigned byte	??	label
0009	unsigned byte	??	label
....			
xxxx	unsigned byte	??	label

The labels values are 0 to 9.

TRAINING SET IMAGE FILE (train-images-idx3-ubyte):

[offset]	[type]	[value]	[description]
0000	32 bit integer	0x00000803(2051)	magic number
0004	32 bit integer	60000	number of images
0008	32 bit integer	28	number of rows
0012	32 bit integer	28	number of columns
0016	unsigned byte	??	pixel
0017	unsigned byte	??	pixel
....			
xxxx	unsigned byte	??	pixel

Pixels are organized row-wise. Pixel values are 0 to 255. 0 means background (white), 255 means foreground (black).

TEST SET LABEL FILE (t10k-labels-idx1-ubyte):

[offset]	[type]	[value]	[description]
0000	32 bit integer	0x00000801(2049)	magic number (MSB first)
0004	32 bit integer	10000	number of items
0008	unsigned byte	??	label
0009	unsigned byte	??	label
....			
xxxx	unsigned byte	??	label

The labels values are 0 to 9.

TEST SET IMAGE FILE (t10k-images-idx3-ubyte):

[offset]	[type]	[value]	[description]
0000	32 bit integer	0x00000803(2051)	magic number
0004	32 bit integer	10000	number of images
0008	32 bit integer	28	number of rows
0012	32 bit integer	28	number of columns
0016	unsigned byte	??	pixel
0017	unsigned byte	??	pixel
....			
xxxx	unsigned byte	??	pixel

Pixels are organized row-wise. Pixel values are 0 to 255. 0 means background (white), 255 means foreground (black).

Figure C-1: Technical details of MNIST Dataset

C.2 DNN

C.2.1 Technical details

Table C-1: Technical details of various parameters used for the experiments using DNN for experiments using MNIST dataset

Parameter	Value
Training algorithm	Stochastic gradient descent (SGD)
Number of epochs	500
Learning rate	0.31
Momentum	0.38
Softmax (training)	Back Propagation
Training and testing ratio	70:30
Resampling	Cross validation (3-fold)
Software	MATLAB 2014a, MATLAB 2016b, Python
Average number of runs	50
Topology	785-1024-2048.2048-1024-785

C.2.2 Classification Results

Table C-2: Classification results for MNIST and modified MNIST datasets using DNNs

Experiment No	Dataset	No. of Hidden Layers	Root Mean Squared Error	Classification Accuracy (%)	T-Test
1	MNIST	7	0.0419	99.27	0.885
2		13	0.0544	97.87	0.057
3		17	3.050	73.02	0.805
4		33	5.531	58.46	0.678
5	M-MNIST1	7	0.140	99.53	0.056
6		13	0.091	98.83	0.176
7		17	2.179	72.68	0.854

8		33	3.015	64.74	0.202
9	M-MNIST2	7	0.0695	99.7	0.397
10		13	0.88	96.01	0.595
11		17	2.24	76.14	0.896
12		33	1.737	57.09	0.113
13	M-MNIST3	7	0.886	99.7	0.490
14		13	1.2901	96.01	0.473
15		17	3.267	76.14	0.125
16		33	4.351	57.09	0.557
17	M-MNIST4	7	1.035	98.41	0.315
18		13	1.489	97.27	0.669
19		17	2.610	74.56	0.280
20		33	2.994	60.04	0.125
21	M-MNIST5	7	1.654	98.24	0.357
22		13	1.246	97.25	0.437
23		17	3.362	78.99	0.965
24		33	3.864	60.46	0.609
25	M-MNIST6	7	0.565	98.29	0.253
26		13	0.311	98.85	0.824
27		17	1.112	76.04	0.733
28		33	2.223	63.04	0.633
29	M-MNIST7	7	0.827	98.25	0.085
30		13	0.972	96.27	0.388

31		17	1.77	74.49	0.878
32		33	2.940	59.57	0.242

C.3 DBN

C.3.1 Technical Details

Table C-3: Technical details of various parameters used for the experiments using DBN for experiments using MNIST dataset

Parameter	Value
Training algorithm	Contrastive Divergence (CD)
Number of epochs	500
Learning rate	0.5
Momentum	0.5
Softmax (training)	Back Propagation
Training and testing ratio	70:30
Resampling	Cross validation (3-fold)
Software	MATLAB 2015a, Python
Average number of runs	50
Topology	785-1024-2048..2048-1024-785

C.3.2 Classification Results

Table C-4: Classification results for MNIST and modified MNIST datasets using DBNs

Experiment No	Dataset	No. of Hidden Layers	Root Mean Squared Error	Accuracy (%)	T-Test
1	MNIST	7	0.0596	99.72	0.885
2		13	0.0981	97.65	0.057
3		17	2.236	76.7	0.805
4		33	3.458	58.5	0.678
5	M-MNIST1	7	0.989	98.52	0.056
6		13	1.247	97.36	0.176
7		17	2.200	82.6	0.854
8		33	6.048	61.72	0.202
9	M-MNIST2	7	0.0938	99.02	0.397
10		13	0.985	98.72	0.595
11		17	2.014	73	0.896
12		33	4.036	54	0.113
13	M-MNIST3	7	0.692	98.14	0.490
14		13	0.564	98.27	0.473
15		17	1.582	80.33	0.125
16		33	5.810	57.17	0.557
17	M-MNIST4	7	0.601	98.06	0.315
18		13	0.436	98.76	0.669
19		17	1.244	86.15	0.280
20		33	4.070	53.44	0.125
21		7	0.773	98.99	0.357

22	M-MNIST5	13	0.920	97.79	0.437
23		17	1.115	84.83	0.965
24		33	2.269	64.56	0.609
25	M-MNIST6	7	0.0565	99.31	0.253
26		13	1.128	98.24	0.824
27		17	2.064	72.38	0.733
28		33	3.143	66.21	0.633
29	M-MNIST7	7	0.0888	99.51	0.085
30		13	0.550	98.92	0.388
31		17	2.059	73.67	0.878
32		33	4.747	55.31	0.242

C.4 DAE

C.4.1 Technical Details

Table C-5: Technical details of various parameters used for the experiments using DAE

Parameter	Value
Training algorithm	Greedy layer wise
Number of epochs	500 for first and last autoencoders, 1000 for the middle autoencoders
Learning rate	0.4-0.6
Momentum	0.2-0.4
Softmax (training)	Unsupervised Back Propagation (only for validation)
Training and testing ratio	70:30
Resampling	Cross validation (3-fold)
Software	MATLAB 2014a, MATLAB 2017a
Average number of runs	50
Topology	500-1000….500

C.4.2 Classification Results

Table C-6: Classification results for MNIST and modified MNIST datasets using DAEs

Experiment No	Dataset	No. of Hidden Layers	Root Mean Squared Error	Accuracy (%)	T-Test
1	MNIST	3	0.0684	99.72	97.96
2		5	0.550	97.65	99.43
3		9	1.144	76.7	79.69
4	M-MNIST1	3	0.0474	98.52	97.3
5		5	0.739	97.36	98.45
6		9	1.171	82.6	81.91
7	M-MNIST2	3	0.0423	99.02	96.8
8		5	0.262	98.72	98.71
9		9	1.740	73	75.37
10	M-MNIST3	3	0.084	98.14	95.33
11		5	0.034	98.27	98.21
12		9	1.130	80.33	76.31
13	M-MNIST4	3	0.095	98.06	97.4
14		5	0.0760	98.76	99.77
15		9	1.772	86.15	82.19
16	M-MNIST5	3	0.0596	98.99	97.72
17		5	0.0897	97.79	99.5
18		9	0.963	84.83	78.83
19	M-MNIST6	3	0.0296	99.31	97.9
20		5	0.278	98.24	98

21		9	1.462	72.38	84.63
22	M-MNIST7	3	0.0105	99.51	95.24
23		5	0.67	98.92	99.79
24		9	2.022	73.67	80.52

C.4 Components (Variance based)

Table C-7: Experimental results of variance based component extraction

Layer No.	**Average Variance**		**Components**
	M-MNIST5	**M-MNIST6**	
1	2.40	4.31	2
2	2.74	4.00	2
3	2.80	4.79	2
4	2.86	4.45	2
5	2.20	4.90	2
6	2.42	4.46	2
7	2.34	4.59	2
8	2.37	4.23	2
9	2.94	4.58	2
10	2.18	4.32	2
11	2.34	4.49	2
12	2.64	4.90	2
13	2.15	4.10	2

D. AN4 Dataset

D.1 Technical Details

Table D-1: Technical details of various parameters used for the experiments using AN4 Dataset

Parameter	ANN	DAE
Training algorithm	Back Propagation	Greedy layer wise
Number of epochs	500	100
Learning rate	0.05	0.05
Momentum	0.2	0.2
Softmax (training)	-	Back Propagation
Training: Validation: Testing	70:15:15	
Resampling	Cross validation (3-fold)	
Software	MATLAB 2017a	MATLAB 2017a
Average number of runs	30	40

D.2 Classification Results of AN4 Datasets

Table D-2: Classification results for AN4 speaker dataset

Experiment No	Classifier	No. of Hidden Layers	Hidden Layer Number	Number of Neurons	Root Mean Squared Error	Accuracy (%)	T-Test
1	ANN	1	1	16	0.24	83.7	0.015
2		2	1	16	0.39	71.15	0.029
			2	22			
3		3	1	16	4.15	39.0	0.03
			2	12			
			3	22			
4	DAE	1	1	16	0.19	79.4	0.027
5		**3**	**1**	**16**	**0.112**	**98.8**	**0.003**
			2	**20**			
			3	**20**			
6		5	1	16	0.34	69.16	0.022
			2	20			
			3	20			
			4	20			
			5	20			

E. TIMIT dataset

E.1 Technical Details

Table E-1: Technical details of various parameters used for the experiments using TIMIT Dataset

Parameter	ANN	DAE
Training algorithm	Back Propagation	Greedy layer wise
Number of epochs	500	500
Learning rate	0.05	0.05
Momentum	0.2	0.2
Softmax (training)	-	Back Propagation
Training: Validation: Testing	70:15:15	
Resampling	Cross validation (3-fold)	
Software	MATLAB 2017a	MATLAB 2017a
Average number of runs	40	

E.2 Classification Results

Table E-2: Classification results for TIMIT speaker dataset

Exp. No	Classifier	No. of layers	Layer number	No. of neurons	Training Error	Testing Error	Accuracy (%)	T-Test
1	ANN	1	1	124	0.19	0.312	86.5	0.031
2		2	1	124	0.51	0.476	81.1	0.025
			2	200				
3		3	1	124	1.15	1.845	63.15	0.038
			2	88				
			3	124				
4	DAE	1	1	780	0.42	0.31	91.2	0.029
5		3	1	780	0.13	0.116	98.2	.0091
			2	1024				
			3	780				
6		5	1	780	0.24	0.27	89.5	0.014
			2	1024				
			3	1024				
			4	1024				
			5	780				
7		7	1	780	0.85	0.65	57.61	0.081
			2	824				
			3	1024				
			4	1024				
			5	1024				
			6	824				
			7	780				

F. Image Datasets

F.1 Technical Details

Table F-1: Technical details of various parameters used for the experiments using Image Datasets

Parameter	DNN	DBN
Training algorithm	SGD	CD
Number of epochs	500	500
Learning rate	0.05	0.05
Momentum	0.2	0.2
Softmax (training)	Back Propagation	
Training: Validation: Testing	70:15:15	
Resampling	Cross validation (3-fold)	
Software	MATLAB 2018a, Python (Tensor Flow)	
Average number of runs	50	
Topology	500-750-1000..2000..1000-750-500	

F.2 Classification Results for CIFAR10

Table F-2: Classification results for CIFAR-10 image dataset

Experiment No	Classifier	No. of Hidden Layers	Root Mean Square Error	Classification Accuracy (%)	T-Test
1	DNN	7	6.53	50.24	0.092
2		12	0.72	86.9	0.081
3		18	2.45	72.6	0.073
4	DBN	7	6.01	51.9	0.071
5		12	0.45	84.3	0.051
6		18	2.2	70.9	0.073

F.3 Classification Results for CIFAR10-M

Table F-3: Classification results for modified CIFAR-10 image dataset

Experiment No	Classifier	No. of Hidden Layers	Root Mean Square Error	Classification Accuracy (%)	T-Test
1	DNN	7	3.89	69.1	0.082
2		12	1.13	88.6	0.064
3		18	5.57	53.09	0.085
4	DBN	7	6.24	43.2	0.006
5		12	0.85	89.8	0.085
6		18	4.1	64.7	0.057

F.4 Classification Results for ImageNet

Table F-4: Classification results for ImageNet image dataset

Experiment No	Classifier	No. of Hidden Layers	Root Mean Square Error	Classification Accuracy (%)	T-Test
1	DNN	7	8.78	39.5	0.083
2		12	1.3	81.8	0.044
3		18	3.34	66.8	0.081
4	DBN	7	8.88	39.2	0.011
5		12	1.19	86.4	0.043
6		18	1. 56	81.4	0.099

F.5 Classification for layer Transfer: layer replacement

Table F-5: Experimental results for transfer of layers experiments on ImageNet and CIFAR-10 datasets

Architecture	ImageNet			CIFAR-10		
	DNN	DBN	DAE	DNN	DBN	DAE
Untrained	24.6	19.6	11.1	32.1	38.6	14.8
First Layer	25.1	20.2	14.5	39.4	41.3	16.5
Fifth Layer	37.8	45	29.9	42.2	49.2	33.1
Middle Layer	65.3	78.3	90.6	79.5	78.6	89.4

F.6 Classification for layer Transfer: middle layer

Table F-6: Experimental results for transfer of middle layer experiments on ImageNet and CIFAR-10 datasets

Architecture	ImageNet			CIFAR-10		
	DNN	DBN	DAE	DNN	DBN	DAE
Untrained	48.3	47.5	16.5	33	32.4	8.34
Middle Layer	36	34.76	11	26.6	29	6.6

F.7 Results of Component Model

Table F-7: Experiment results for component extraction experiments using image datasets (CIFAR-10, ImageNet)

n features	ImageNet			CIFAR-10		
	DNN	DBN	DAE	DNN	DBN	DAE
Full features	24	21	16	20	21	14
Strategy1	19	17	12	17	17	11
Strategy2	11	9	3	14	15	8

G. Air Pollution Dataset

G.1 Technical Details

Table G-1: Technical details of various parameters used for the experiments using Air Pollution Dataset

Parameter	Value
Training algorithm	SGD
Number of epochs	500
Learning rate	0.4-0.6
Momentum	0.2-0.4
Softmax (training)	Back Propagation
Training and testing ratio	70:30
Resampling	Cross validation (3-fold)
Software	MATLAB 2017a, .NET
Average number of runs	50
Topology	300 (selected after several experiments)

G.2 Classification Results

Table G-2: Classification results for Air Pollution dataset

Experiment No	No. of Hidden Layers	Root Mean Squared Error	Accuracy (%)	T-Test
1	7	0.62	90.85	0.035
2	13	2.29	68.15	0.017
3	18	4.94	54.86	0.082

H. Gene Expression Dataset (Prostate Cancer)

H.1 Technical Details

Table H-1: Technical details of various parameters used for the experiments using Gene expression Dataset

Parameter	Value
Training algorithm	SGD
Number of epochs	500
Learning rate	0.4-0.6
Momentum	0.1-0.3
Softmax (training)	Back Propagation
Training and testing ratio	70:30
Resampling	Cross validation (3-fold)
Software	MATLAB 2017a, .NET
Average number of runs	50
Topology	30 (selected after several experiments)

H.2 Classification Results

Table H-2: Classification results for Gene expression dataset

Experiment No	Dataset	No. of Hidden Layers	Root Mean Squared Error	Accuracy (%)	T-Test
1	Gene	1	0.34	93.1	0.02
2		2	0.193	94.7	0.013
3		3	0.013	100	0.017
4		4	0.021	97.0	0.014
5		5	0.055	96.1	0.03
6	Gene-M	1	15.1	34.6	0.021
7		2	10.9	43.1	0.11
8		3	2.6	75.3	0.13
9		4	4.33	62.9	0.27
10		5	5.32	51.2	0.14

I. Hardware and Software Specifications

I.1 Hardware Specifications

Table I-1: Technical details of the hardware used for the experiments

Desktop (GPUs)	GTX1080Ti(GPU) GTX 1060 LINUX RTX 2080 • CPU Family: Intel Core i7 • GPU Model:RTX2080 • SSD Capacity:256 GB • Memory Size:32GB Memory • HDD Capacity:2TB HDD • Operating System: Linux • Optical Drive: DVD-Drive • VR Ready: Yes
	University Desktop Windows 10 Inter I5 CPU, 3.3 16 GB DDR3 RAM, 8GB Cache 1TB Hard Disk
Laptops	**Microsoft Surface Book, Windows 10** Intel i7-465OU 1.7GH up to 3.3 GHz, 4MB Cache 8GB LPDDR3 RAM 1600 MHz, 256 GB SSD Storage Intel Integrated HD5000 CPU
	Microsoft Surface Pro 2, Windows 10 Intel i5-4300U 1.9GH up to 2.9 GHz, 4MB Cache 4GB LPDDR3 RAM 1600 MHz, 128GB SSD Storage Intel Integrated HD4400 CPU
	MacBook pro 2015 1.2GHz dual-core Intel Core M processor (Turbo Boost up to 2.6GHz) with 4MB shared L3 cache 8GB of 1600MHz LPDDR3 onboard memory 1.2GHz 512GB PCIe-based onboard flash storage Intel HD Graphics 5300

I2. Software Specifications

Table I-2: Technical details of various software used for the experiments

Software	Technical details
Weka	Ver. 3.8.3 (stable) and 3.9.3 (developer) Windows and Mac IA-32, x86-64; Java SE
MATLAB	R2013b through R2018b Windows and Linux
Python	Python 3.7.2 NumPy 1.5
TensorFlow (Keras)	TensorFlow 1.1 Keras neural network io framework
Microsoft .NET	ML.NET framework (Machine Learning framework by Microsoft) Auto ML.

J. Experiment Results: Initial Experiments

J1. Classification accuracy for MNIST, SYN and IRIS Dataset

Table J-1: Classification accuracy and T-Test values for MNIST, Synthetic and IRIS Dataset

	Original	Replacing M	T-Test
MNIST	90.2	93.8	0.002
SYN	91.4	96.5	0.0012
IRIS	89.2	98.6	0.012

J2. Execution time for MNIST, SYN and IRIS Dataset

Table J-2: Execution time for MNIST, Synthetic and IRIS Dataset

Dataset	Training time in hours
MNIST	76
MNIST(M)	41
SYN	124
SYN(M)	87
IRIS	49
IRIS(M)	29

J3. Execution time for MNIST, SYN and IRIS Dataset

Table J-3: Classification accuracies and T-Test values for MNIST, Synthetic and IRIS Datasets

No. of Layers	Classification Accuracies (%)											
	IRIS				MNIST				Synthetic			
	No training		Transferred		Random		Transferred		Random		Transferred	
	%	T Test	%	T Test	%	T Test	%	T Test	%	T Test	%	T Test
5	51.2	0.037	76.5	0.002	33.5	0.022	80.2	0.006	44.6	0.004	85.2	0.003
9	43.5	0.046	7.1.2	0.032	45.3	0.081	76.8	0.0058	57.4	0.006	79.7	0.008