OBJECT CLASSIFICATION USING STACKED AUTOENCODER AND CONVOLUTIONAL NEURAL NETWORK

A Paper
Submitted to the Graduate Faculty
of the
North Dakota State University
of Agriculture and Applied Science

By

Vijaya Chander Rao Gottimukkula

In Partial Fulfillment of the Requirements
for the Degree of
MASTER OF SCIENCE

Major Department:
Computer Science

November 2016

Fargo, North Dakota

North Dakota State University
Graduate School

Title

OBJECT CLASSIFICATION USING STACKED AUTOENCODER AND CONVOLUTIONAL NEURAL NETWORK

By

Vijaya Chander Rao Gottimukkula

The Supervisory Committee certifies that this ***disquisition*** complies with North Dakota State University's regulations and meets the accepted standards for the degree of

MASTER OF SCIENCE

SUPERVISORY COMMITTEE:

Dr. Simone Ludwig
Chair

Dr. Anne Denton

Dr. María de los Ángeles Alfonseca-Cubero

Approved:

11/15/2016
Date

Dr. Brain Slator
Department Chair

ABSTRACT

In the recent years, deep learning has shown to have a formidable impact on object classification and has bolstered the advances in machine learning research. Many image datasets such as MNIST, CIFAR-10, SVHN, Imagenet, Caltech, etc. are available which contain a broad spectrum of image data for training and testing purposes. Numerous deep learning architectures have been developed in the last few years, and significant results were obtained upon testing against datasets. However, state-of-the-art results have been achieved through Convolutional Neural Networks (CNN). This paper investigates different deep learning models based on the standard Convolutional Neural Networks and Stacked Auto Encoders architectures for object classification on given image datasets. Accuracy values were computed and presented for these models on three image classification datasets.

ACKNOWLEDGEMENTS

I would like to express my gratitude to Dr. Simone Ludwig for her constant encouragement and support towards the successful completion of my masters paper. Her useful insights and positive critique have helped me understand and develop my skills in the subject. I would also like to thank Dr. Anne Denton and Dr. María de los Ángeles Alfonseca-Cubero for their willingness to serve on my committee and provide useful inputs.

Also, I want to thank my parents and friends for their unending support and inspiration.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

AI	Artificial Intelligence
SVM	Support Vector Machines
CNN	Convolutional Neural Networks
DBN	Deep Belief Nets
MNIST	Mixed National Institute of Standards and Technology
CIFAR	The Canadian Institute for Advanced Research
SVHN	The Street View House Numbers
KDD	Knowledge Discovery in Databases
SAE	Stacked Auto Encoder
SGD	Stochastic Gradient Descent
RBM	Restricted Boltzmann Machines
ReLu	Rectified Linear Unit

1. INTRODUCTION

Man is by nature a social animal. Over the past few centuries, he achieved many remarkable feats and fuelled many significant technological innovations. The advent of computers heralded a new era in the field of technology. In spite of making breakthrough technologies, it is essential to understand that machines are a creation of mankind and do not primarily work as human brains do. Therefore, it is imperative that the machine needs to be furnished with appropriate instructions to get our job done.

1.1. Artificial Intelligence and Machine Learning

Replicating the mode of working of a human brain is a daunting task. In the past few decades, computer systems have been programmed to perform simple tasks which require knowledge, perception, reasoning and other such cognitive abilities. Artificial Intelligence (AI) attempts to build these systems and provide a degree of "intelligence" to the computers. From the computational point of view, intelligence can be implemented either through symbolism or connectionism. While the former is through the use of symbols, the latter is by associated weights and connections [1]. Natural language processing, computer vision, automated programming, intelligent computer- assisted programming and robotics are some of the broad- based applications of Artificial Intelligence.

Over the years approaches in AI have focused on retrieving results based on knowledge inputted by the human. Hence the machine extracts answer based on the open data but falters when it encounters a question, for which it does not have an answer in the knowledge base. Machine learning is an area of artificial intelligence, which attempts to recognize objects by using the method of pattern classification [2]. Therefore, machine learning comes up with an answer by initially recognizing the pattern, sorting it and finally classifying it into a predictable pattern.

Until the introduction of this learning technique, human intervention was essential to deal with unknown data, but machine learning accelerated the capability of dealing with unknown data without human aid. Machine learning technology has made inroads into almost every sphere of learning, ranging from bioinformatics and web searches to consumer product technology such as mobile technology. Machine learning has its share of shortcomings too [2]. Feature engineering is one such area, which remains unaddressed since recognition of objects or features in an image was not possible.

1.2. Computer Vision and Object Recognition

Human Vision is unique in a way that it readily identifies objects in an image with little or no difficulty. The human vision effortlessly captures the scale, size and angularity of the objects in the image. In comparison, computer vision faces significant challenges due to variations in real world images. Recognition algorithms need to be robustly engineered to overcome these difficulties [3]. These algorithms need to be trained to use digital images and correctly classify objects. In this technological era where life is increasingly dependent on search engines where object identification, image recognition, speech recognition and its transcription into text and signal processing are becoming essential in turning in relevant results for the end user.

1.3. Shortcomings of Conventional Techniques

Appropriate data has to be provided by the human for the machine to chart out the differences and these values given as inputs are often referred to as features. Nevertheless, the machine can achieve the degree of precision through feature engineering which seems to be a tough proposition conventional machine learning techniques suffer some significant drawbacks which include their inability to process natural data from the raw state. Considerable domain expertise is also needed to extract the features from the raw data. The raw data often needs to be processed into

an initial representational state, which will make it easier for building classifiers or predictors. This state is achieved by coalescing multiple nonlinear transformations giving rise to a much abstract representation. The shortcomings in conventional techniques have gradually transformed into better methods, which can process raw data much more efficiently [4]. Though feature engineering is an important aspect to be considered, construction of learning algorithms through deep learning helps to broaden the applicability of machine learning and reduce dependence on them.

1.4. Evolution of Deep Learning

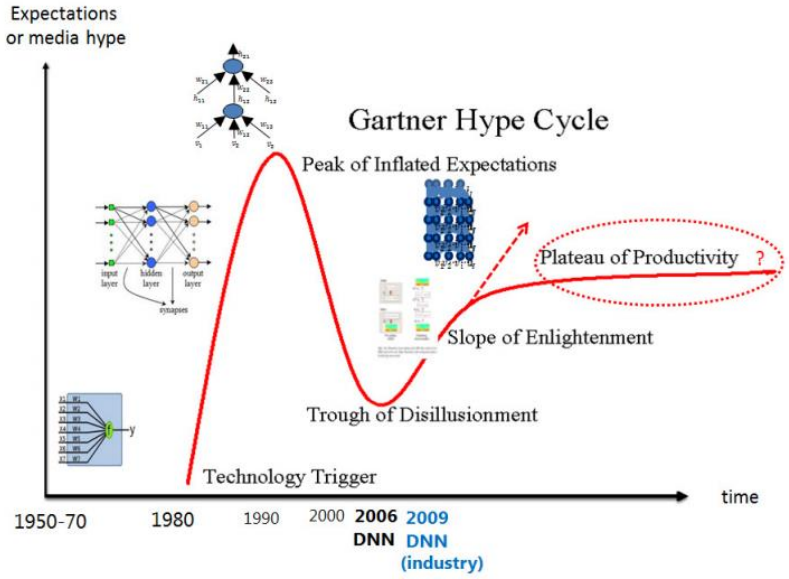

Figure 1. History of neural networks [5]

The discovery of the perceptron in the late 1950s, by Frank Rosenblatt, generated considerable interest in neural networks but many nonlinear decision functions such as the XOR function cannot be approximated. After a gap of almost two decades, the concept of the backpropagation algorithm again brought neural networks to the fore. In the 1980s, researchers introduced the concept of multilayer perceptrons, which are essentially a stack of linear classifiers. This discovery tried to address the solution of approximating nonlinear decision functions. These systems suffered disadvantages such as inability in training unlabeled data, weakening of the signal

upon passing through different layers and poor local minima [6]. Several improvements to solve nonlinear problems were made such as the Support Vector Machines (SVMs) but slowed down the research on neural networks [7]. Except Convolutional Neural Networks (CNNs) none of them were very accurate and efficient.

Geoffrey Hinton in 2006 proposed a model called Deep Belief Nets (DBN), a machine learning algorithm which triggered interest in deep learning [8]. Deep learning refers to the presence of more than two layers in a neural network. The term "deep" signifies the importance of the use of unlabeled data without human intervention. Several deep learning architectures surfaced later, but CNNs, and DBNs are the most significant architectures. This first breakthrough of DBNs led to similar gains by Autoencoders [6]. Figure 1 represents a hype curve and gives an overview of the different phases in conception and development of neural networks. In the present paper, we will discuss Convolutional Neural Network and Stacked Autoencoders and apply them to popular image datasets such as Mixed National Institute of Standards and Technology (MNIST) [9], The Street View House Numbers (SVHN) [10], and The Canadian Institute for Advanced Research (CIFAR-10) [11].

2. RELATED WORK

An enormous amount of data is continuously being added to the existing databases. Consequently, this data needs to be analyzed to extract significant information e.g., for business organizations. The past few years has seen a surge in the number of techniques for storing, retrieving and manipulating data. Knowledge Discovery in Databases (KDD) refers to the overall process of discovering useful information from data [12]. The KDD process is represented diagrammatically in Figure 2. Data mining is one of the important steps of KDD focusing on algorithms, which discover new patterns. It deals with the exploration and analysis of large amounts of data by automatic or semi-automatic means, in order to discover meaningful patterns and rules.

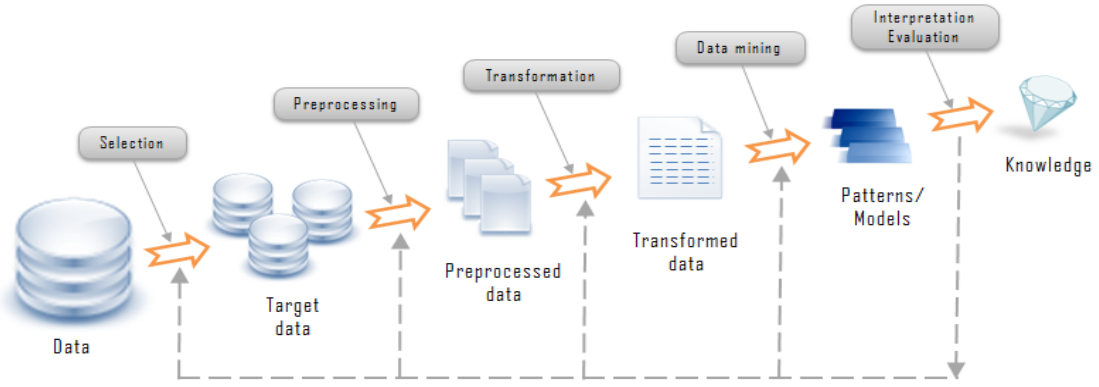

Figure 2. Data mining as a core process in KDD [13]

The task of achieving data mining objectives is classified into two methods namely descriptive and predictive as shown in Figure 3. While the former aims to characterize the properties of the dataset, the latter focuses on drawing inferences from the data to make meaningful predictions [12]. One important and useful predictive technique is classification.

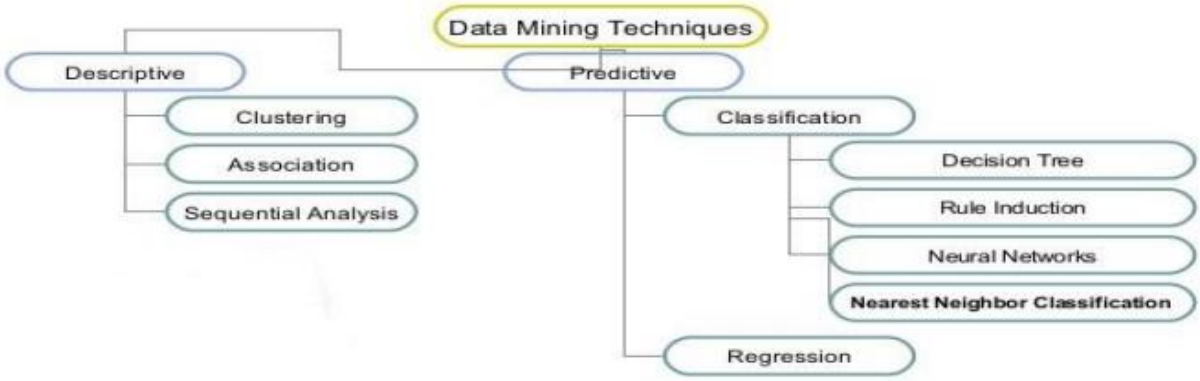

Figure 3. Descriptive and predictive data mining techniques [13]

The classification technique is a data mining function that aims to assign items in a collection to target classes or categories. For every individual case in the data, this technique tries to predict the target class accurately. According to Weiss and Kuulikowski "Classification is learning a function that maps (classifies) a data item into one of several predefined classes" [14].

Image classification is an essential step in image processing. It bridges the gap between the image and the presented object. Classification of an image is not an easy procedure since the image is essentially a group of pixels to the untrained machines and often contains noise or blurred contents. The task becomes more complex due to the presence of different objects in the image. Predefined patterns or images present in a database are used to test the unknown images for their classification [15]. Three main image classification techniques are available [16]:

- Supervised classification consisting of labelled data points;
- Unsupervised classification consisting of random, unlabeled data;
- Semi-supervised classification, which uses unlabeled data points and removes interaction between domain and user to get rid of bias.

2.1. Classification Algorithms

In the first step, a model is created by applying an algorithm on the given dataset. In the second step, a predefined test dataset is used to test the model. The testing step aims to measure the performance and accuracy of the model. MNIST dataset is one of the most famous and frequently used datasets on which different classifiers were trained and tested. The following are the most popular approaches to training the classifiers.

2.1.1. K-Nearest Neighbors

The k-nearest neighbor algorithm is an example of instance-based learning or memory-based learning [17]. It takes advantage of the fact that the hypotheses are constructed from the training instances. Prediction of objects is based on k-nearest neighbors in the feature space. It is the simplest non-parametric method for classification and regression.

Classification of the object is usually done by a majority vote of the k-nearest neighbors. Therefore, if k =1, the object is usually assigned to the closest neighboring class. The same method is applied to regression too. The distance metric is calculated using the Euclidean distance, which is calculated using the formula

$$\boldsymbol{d(x,y) = \sqrt{\sum_{i=1}^{m} (x_i - y_i)^2}}$$

where x is the test set and y is the training set, xi and yi represent the same features belonging to their respective classes.

The greatest advantage of this approach is its adaptability in dealing with data that has never been encountered before. It also differs from other approaches in discarding the old instances and simply storing the new instances. Some of the downsides are the increasing complexity of hypotheses with the increasing amount of data and presence of irrelevant features [18].

2.1.2. SVM

SVM stands for Support Vector Machines. This classification method, invented by Vladmir Vapnik in 1979, divides data points or classes using hyperplanes. These hyperplanes maximize the margins between the differing classes. This method ensures that the generalization error is minimized [19]. Figure 4 shows the classification two different classes using SVM. Though SVM classification originally developed as a binary classifier, was further applied to multiclass problems too. Kernel functions were used to project the input data into a three-dimensional space and then applying a linear classification problem in the feature space. Since the feature space is larger than the dataset, several decomposition algorithms are proposed. Therefore, the decomposition problems are applied to decompose the multiclass into two class problems and further implementing them through SVMs.

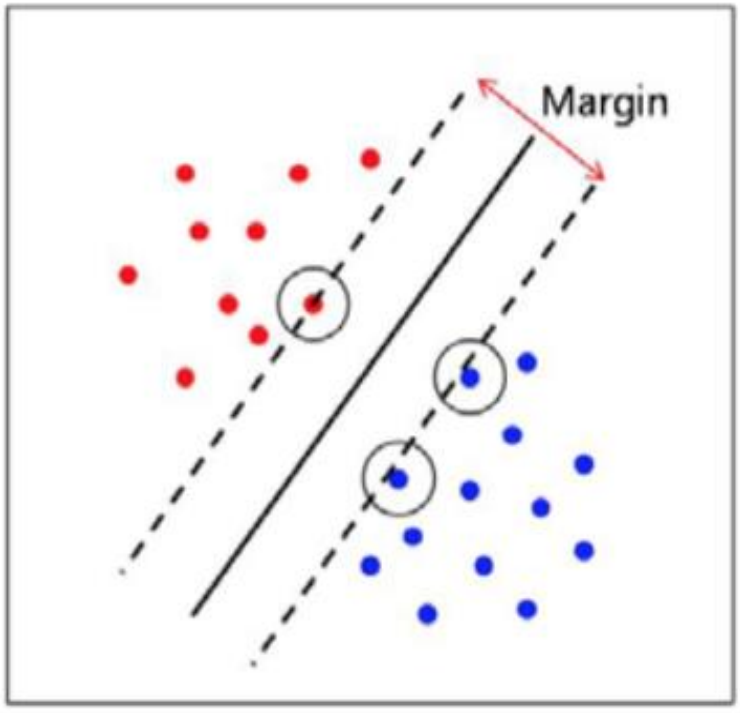

Figure 4. SVM Classification [18]

SVM is a perfect fit for datasets having multiple attributes, but it has disadvantages too. Selection of kernel function parameters and lesser computational speed are some of the disadvantages [18].

2.1.3. Boosted Stumps

Decision trees are learning algorithms, which sort the instances depending on their feature values. Each feature is represented by a node on the decision tree, and the branch represents the value of the node [20]. Though decision trees are easy to interpret, efficient and flexible, they are not very accurate compared to other learning algorithms due to high variance. Boosting is one of the most popular ensemble methods in machine learning, which aid to improve the decision tree learning algorithms [21]. Boosting "boosts" the prospects of decision trees by increasing their accuracy and combines many weak learners into a strong one.

Boosting is a deterministic algorithm and builds the models sequentially. The algorithm is applied sequentially to the misclassified instances by increasing their weights [20]. These instances are therefore focused on for every iteration. Computer vision, document ranking, and behavior analysis are a few important applications.

3. APPROACH

Classification is a technique of building classification models from a given input dataset. The learning algorithm employed by the technique should possess the capability of assigning and predicting labels for unknown records.

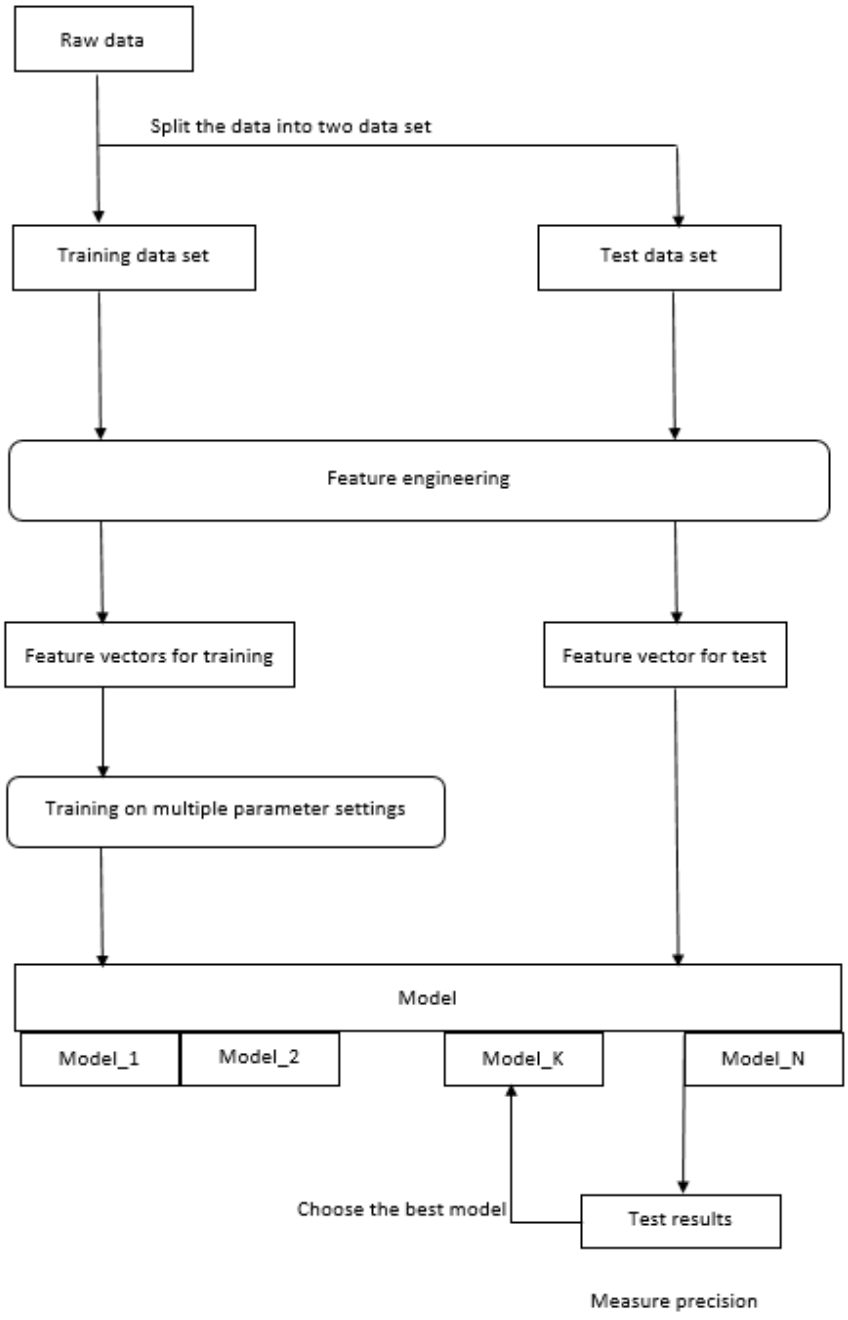

Figure 5. Steps to solve classification problem [2]

Figure 5 displays the steps involved in image classification technique. The learning phase involves two stages:

- Training
- Testing

The training and the testing datasets first need to undergo feature engineering - a process that transforms input patterns into low-dimensional vector representations [2]. In the training dataset, class labels of known records are used to build a classification model. This model is applied to a test set with records containing unknown labels. Choice of the best model is made by evaluating its performance based on the number of records in the test set predicted correctly and incorrectly. The count of the number of correct and incorrect records is tabulated in a table known as confusion matrix. Accuracy and error rate are calculated to compare various models [22]. Models attaining the lowest error rate and the highest possible accuracy are usually the most desirable.

$$Accuracy = \frac{Number\ of\ correct\ predictions}{Total\ number\ of\ predictions}$$

$$Error\ rate\ = \frac{Number\ of\ wrong\ \ predictions}{Total\ number\ of\ predictions}$$

As discussed earlier, deep learning uses representational learning to represent the features and therefore does not require to undergo the process of feature extraction which was deemed as one of the shortcomings of machine language and therefore paved the way to deep learning.

This paper attempts to implement the deep learning architectures - Stacked Auto Encoder [SAE] and Convolutional Neural Networks [CNN] and test them on the following popular image datasets.

- MNIST [9]
- SVHN [10]
- CIFAR-10 [11]

Before we delve into the details about SAE and CNN, we will discuss the following concepts/terms, which are frequently used to implement deep learning architectures.

Forward Propagation: Forward propagation also known as feed forward computation or forward pass is a two-step process. The first step comprises of extracting the values from the hidden layer nodes to compute values of the output layer. In the second step, those values are pitted against the desired outputs and the error is calculated [23]. Since there is no feedback and the information is passed to subsequent layers, this is known as forward propagation.

Backward Propagation: Back propagation is a common method used, along with an optimization method such as stochastic gradient descent, to train deep neural networks. The errors calculated in the forward propagation step are used to adjust the weights of the network, this complete process repeats until the error is reduced to a negligible amount [24]. The gradients or first derivative of loss function/error function with respect to all the weights in the network are calculated, and the chain rule is applied to obtain error derivatives starting from the outermost output layer to the initial input layer backward, hence the name "backward propagation". All the weights in the network are updated through the optimization method after calculating the gradient loss function. This robust method has been found to minimize the error through gradient descent.

Stochastic Gradient Descent: In a deep learning system, several adjustable parameters are present which are often referred to as "weights" and influence the input-output functions of the machine. A typical deep learning architecture consists of a number of adjustable weights. For each of these weights, the learning algorithm computes a gradient vector, which estimates the weight increase or decrease, upon the increase in weight by small amounts. This step is followed by weight vector adjustment in the direction opposite to that of the gradient vector. The average of all the examples in the training set is essentially the objective function and is represented by a hilly

landscape. The steepest descent in this landscape is closer to the minimum and has a low output error. Stochastic Gradient Descent (SGD) is the most commonly used gradient vector [24]. It is known as stochastic since the small set of examples used, gives a noisy estimate of the average gradient of all the total examples. This gradient vector technique results in a good set of weights in comparison to several other optimizing techniques. SGD has an advantage of exhibiting faster convergence rate than that of the batch gradient descent on large datasets.

Momentum: Momentum is an enhancement for SGD and helps in speeding up the convergence process by preserving a portion of the prior weight adjustments [25]. It also helps to prevent convergence to local minima. Nestrovs momentum is used in our experiments, which first takes a step into velocity direction and corrects the velocity vector based on the new location as opposed to the classical momentum which corrects the velocity first and then takes a step in that direction.

Learning Rate: With respect to the adjustable weights, the SGD algorithm moves in the direction of the negative gradient by approaching a local minimum [2]. The size of this step is called the learning rate.

Xavier Rate Initialization: Initialization of weights is an important step towards learning. Updating the weights becomes difficult with wrong weight initializations and makes the gradients either too large or too small.

Xavier rate initialization [26] strives to maintain the same distribution of activations and prevents the gradients from being too small i.e. mean zero with a small variance or too large i.e. mean zero with a large variance.

Overfitting: Due to the presence of multiple non-linear hidden layers, a complex relationship exists between inputs and outputs in a neural network. It also results in the so called “over

expressive" models. Since the training set contains few data, these relationships exist due to the noise factor. The same is not applicable in real data and leads to a phenomenon called "overfitting". Several methods have been proposed to reduce overfitting such as regularization, data augmentation, early stopping and dropout [27]. Regularization is one of the processes implemented to reduce overfitting in the current paper.

3.1. Stacked Autoencoders (SAE)

A solution to the problem of "backpropagation without a teacher" was first attempted by Hinton in the 1980s and Rumelhart in 1986 [28]. Autoencoders were proposed as a solution to this problem, and the input data was used as the teacher. The backpropagation algorithm is traditionally used to train neural networks and literally "back propagates" the error in the network backward from the output layer to the input layer. Since labeled data is required for this task, this is categorized as a supervised learning algorithm. The capability of deep neural networks is limited due to weaknesses of this algorithm. Backpropagation was not effective in deep layers and also most of the available input data was unlabeled [29]. In 2006, Hinton attempted to overcome these problems through Deep Belief Networks (DBN), which are composed of a stack of Restricted Boltzmann Machines (RBM) [8]. Greedy layer-by-layer training is one of the core concepts of DBNs. A similar strategy is used by stacking autoencoders yielding similar results.

3.1.1. Autoencoders

High dimensional data is difficult to store and makes classification and visualization difficult. Reducing the dimensionality of such data is the key to such problems (Hinton 2006). Autoencoders are alternatively known as "Autoassociators" or "Diabolo networks" [30]. They work by encoding the input into a representation, and a decoder works to reconstruct the representation into an output. The output is, therefore, the input itself.

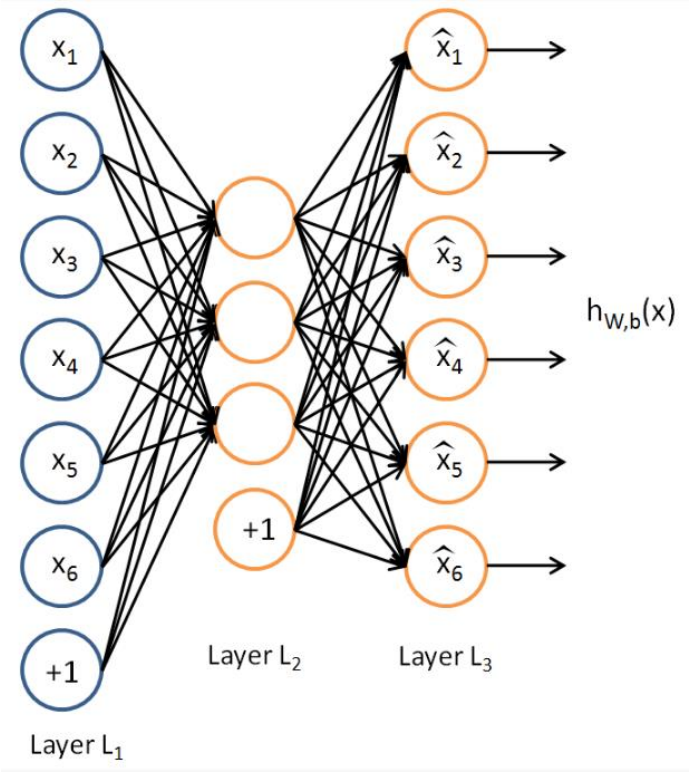

Figure 6. Autoencoder [31]

3.1.2. Stacked Autoencoders

Basic autoencoders act as building blocks to create a deep network in which the autoencoders are stacked on each other. Hence the name Stacked Autoencoder. This multiple layers of autoencoders is widely used in reducing dimensionality [32]. They are trained one layer at a time (layer-wise training) and fine-tuned using backpropagation.

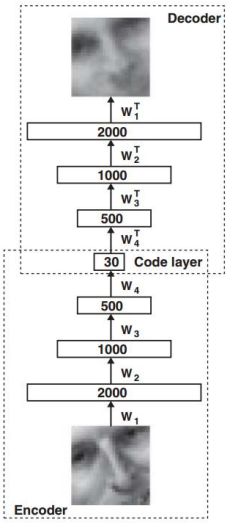

Figure 7. Autoencoder with many layers (SAE) [32]

3.1.3. SAE Architecture for Classification

The stacked autoencoder as the name suggests is comprised of several individual autoencoders similar to Deep Belief Networks, which is comprised of stacked RBMs [8]. These several layers of autoencoders encode and decode the input leading to better representational learning. Better representation refers to "the one which yields a good classifier" [33]. Retaining most of the information from the input is a measure of a good representation since it is essentially the reconstruction of the input. Finally, the SAE consists of a classification layer, which classifies the images. In the current paper, we have implemented the basic model and introduced several layers with multiple nodes. The better performing models are listed in the experimental section.

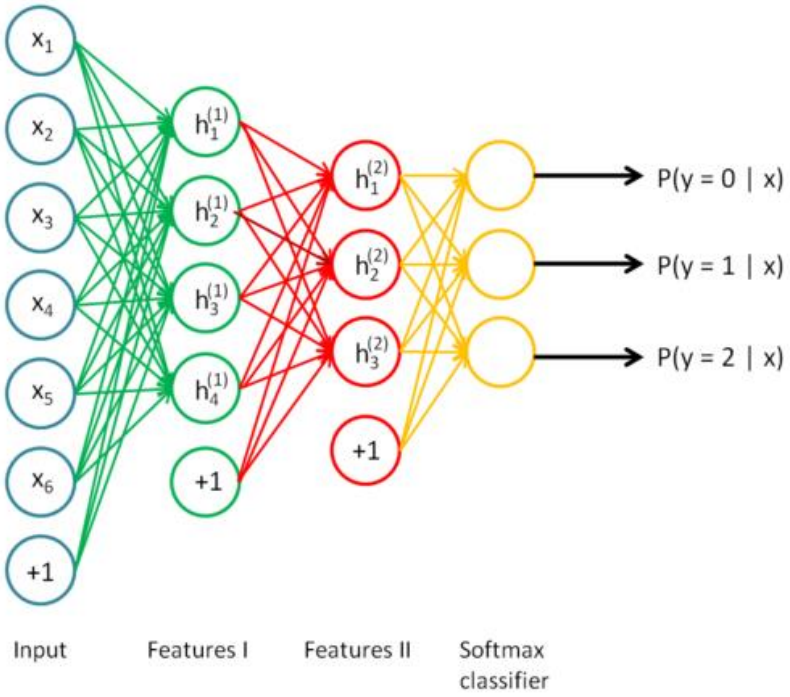

Figure 8. Stacked Autoencoder with classification as output layer [31]

3.1.4. Training Details of Stacked Autoencoder

The presence of hidden layers presents a daunting task in training a deep network. This is attributed to two problems:

Poor local minima: As the depth increases with the addition of layers, there is an increased probability of finding poor local minima [34]. The present paper implements regularization as a solution to this problem.

Vanishing gradient problem: Vanishing Gradient Problem is known as "diminishing gradient flow" or "long time lag" [35]. The neurons in the higher layers pass on the errors to the lower layers. Based on the errors in the output layers, the weights are updated directly in the higher layer. As subsequent layers are added, they are updated based on the error in the higher layer. Therefore, decay is observed in the error rate.

Greedy Layer Wise Training was introduced as a solution to the vanishing gradient problem by Hinton et al. in 2006. Layer wise training involves two stages:

a) Pre-Training: In this step, one layer is trained at a time using unsupervised learning. Use of unlabeled data makes this phase more appealing since this is unsupervised. The representation from each layer serves as input for the next layer, and a new representation is learned which can predict the variables of interest. By training one layer at a time, fewer local minima are involved and also cleaner gradients [8]. This optimization strategy also helps by initializing the weights in a region of the good local minimum. The tougher problems are taken care of in the fine-tuning step.

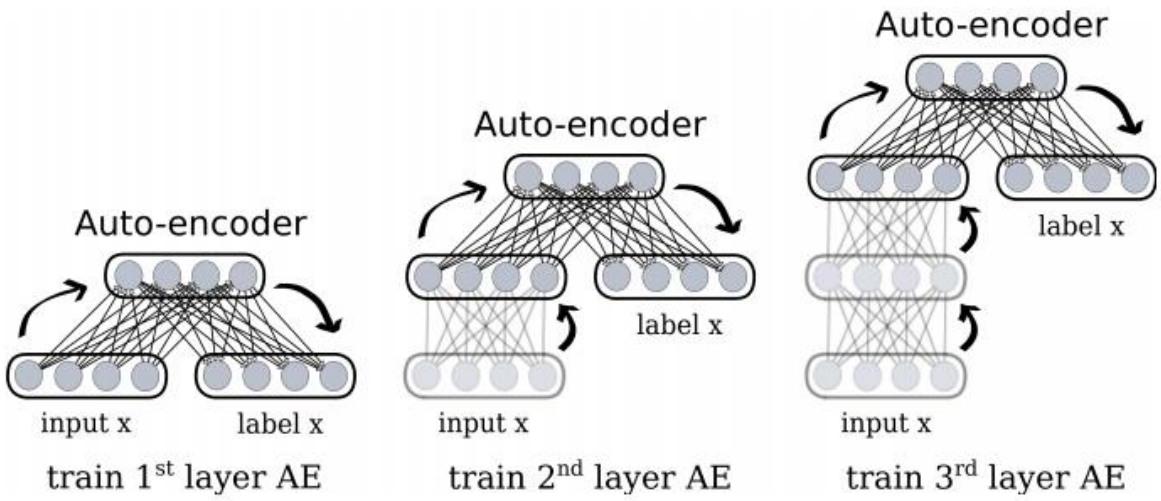

Figure 9. Greedy Layer-wise training in Stacked Autoencoders [36]

Figure 9 explains the layer wise deep training involving the different hidden layers. As the pre-training progresses through the different layers, the previous layers from which the representation output serves as the input are greedily and conveniently ignored.

b) Fine tuning: The greedy layer supervised strategy provides a good initialization step for fine-tuning [8]. After the unsupervised pre-training step, the whole system is subject to supervised fine-tuning which helps to optimize the lower levels in the feature hierarchy in addition to the

classifiers. Stochastic Gradient Descent is used to optimize the loss function by calculating the gradients and fine-tune the network using backpropagation.

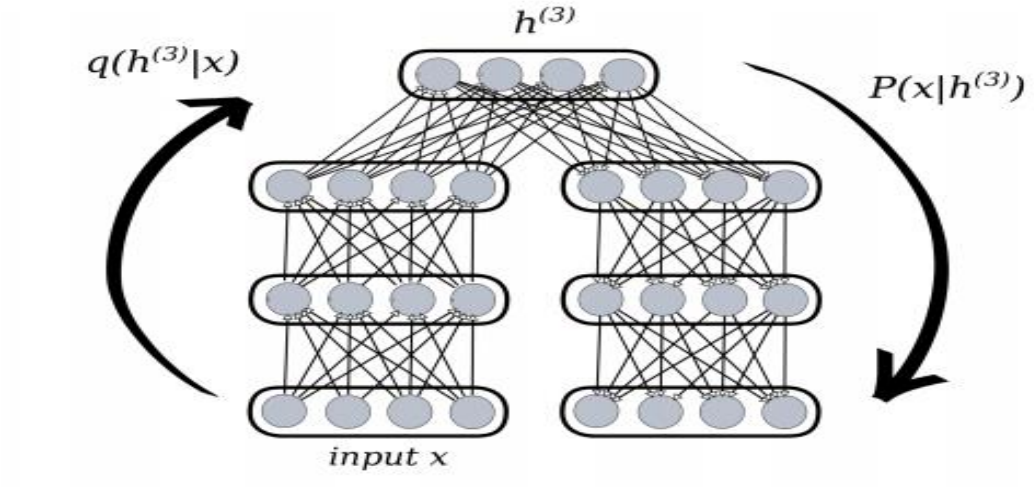

Figure 10. Fine-tuning all the layers in Stacked Autoencoder [36]

3.2. Convolutional Neural Networks (CNN)

Neural networks are characterized by several layers where every individual neuron is connected to all the neurons in the previous layer, but neurons in the same layer are independent and are not connected. The "output layer" is the final fully connected layer and represents the class scores. Neural networks do not take into account the spatial structure of images. A larger number of weights in hidden layers makes it tougher for neural networks to increase the scalability of the image. The presence of many parameters often leads to "overfitting." CNN reduce the number of parameters in the network by weight sharing.

CNN is heavily inspired by the visual mechanisms in living organisms. The visual cortex in the brain is composed of a large number of cells, which function to detect light in small overlapping regions of the visual field known as receptive fields. CNNs in a similar fashion consist of multiple layers of neuron clusters which scan smaller portions of the image known by the same name 'receptive fields'. The resultant collections are designed in a clever way to gain a better

representation of the image [37]. Though the history of CNNs dates back to 1980s through the works of Kunihiko Fukushima on Neocognitrons, it was later refined by Yan LeCun in 1998 [38].

3.2.1. Convolutional Layer

One of the primary functions of this layer is the extraction of features from the provided input. The initial layers extract low level features while the additional layers in the network extract the higher level features. This is achieved by applying filters referred to as kernels and the images convolved are referred to as "feature maps." The input image is scanned by sliding the kernel over it, and the summation of the values is obtained as a multiplication filter within the kernel [24]. Various features can be extracted by considering different kernel values.

A certain portion of the image is referred to as a "channel," and a standard digital image comprises of three channels - red, blue and green as opposed to a grayscale image which is comprised of just one channel. The main advantage of this layer is the establishment of local connectivity through correlation of neighboring pixels [38].

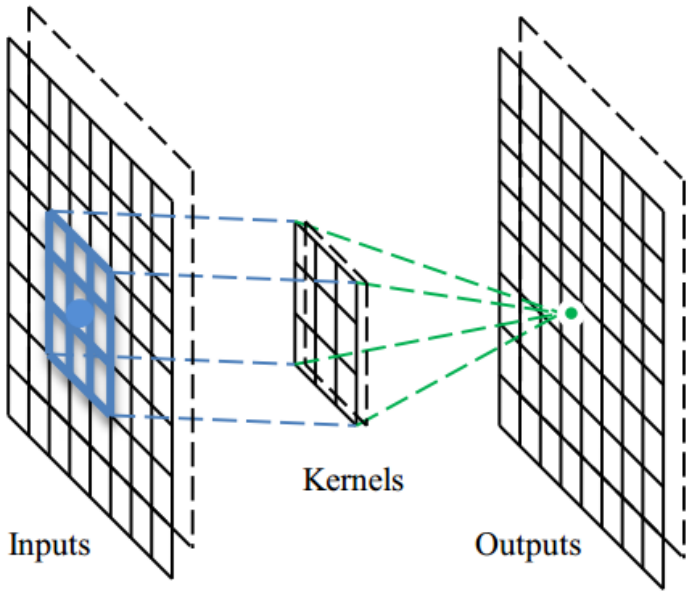

Figure 11. Operation of the convolutional layer [38]

3.2.2. ReLu Layer

ReLU stands for rectified linear unit. Nonlinearity is introduced into the layer of neurons without affecting the receptive layers in the convolutional layer. Hidden units are the ones, which are present neither in the input nor output. The input is subject to non-linear distortion by ReLU in the hidden layers, and as a result, the linear separation can be observed in the final layer categories. Until the advent of ReLUs, smoother non-linearities such as tanh (z) were used [24]. It was shown that with ReLUs, the training error rate on CIFAR-10 is six times faster than the network with non-linearity induced by tanh(z). Convoluted neural networks with the help of ReLUs learn faster in networks, which contain many layers [39]. Faster learning is especially useful when training large datasets.

$$f(x) = \max(0, x)$$

3.2.3. Pooling Layer

A pooling layer is often introduced between two successive convolution layers [24]. The introduction of this layer merges two semantically similar features into a single entity. Images from convolutional layers are downsampled and hence also known as downsampling. Pooling reduces variance by calculating the average or the maximum value of a feature over an image and ensures that the distortions and translations in the image are negligible. This routine is important for object detection and classification. It progressively reduces the parameters and spatial size of the input and makes it more manageable.

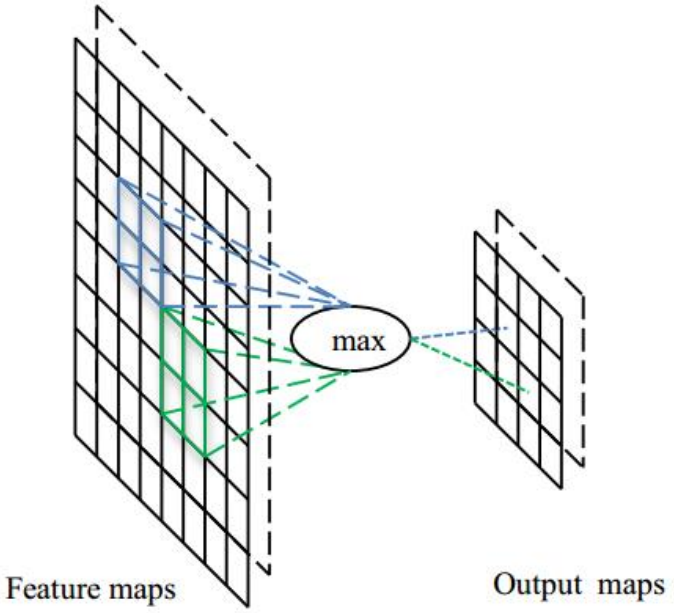

Figure 12. The operation of the max-pooling layer [38]

Max–pooling is the most applied downsampling method, where the image from the convolutional layer is subset in non-overlapping data and the maximum value is outputted from each data [38]. This way the upper layer computation is reduced and maintains the translation invariance.

3.2.4. Fully Connected Layer

In this layer, each neuron in the previous layer is connected to every neuron in the subsequent layer. The image output from the pooling layer serves as the input for the fully connected layer [38]. This layer is typically a traditional neural network and contains most of the parameters of CNN. For image classification, the vector is enabled to feed forward into different categories, and for follow-up processing, it is usually considered as a feature vector.

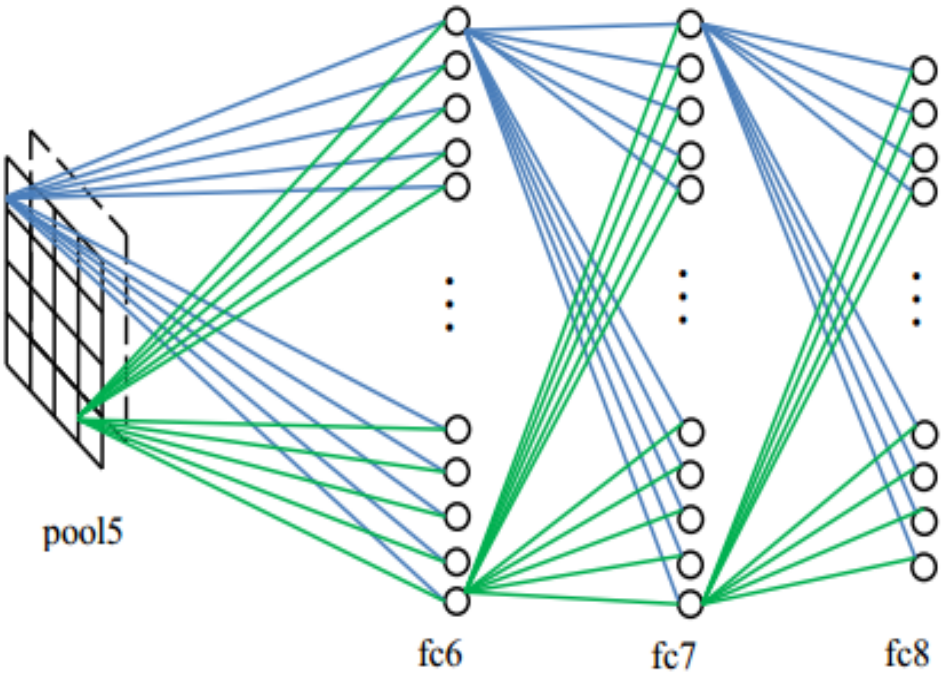

Figure 13. Operation of the fully-connected layer [38]

3.2.5. Output Layer

This layer is unique in a way for the requirement of a specific property as compared to other layers. The probabilities in the output classes should sum to one. This property is achieved through a linear classifier, which uses a log probability distribution such as softmax. The softmax layer is the loss function used in our present experiment.

$$L = \frac{1}{N}\sum_{i} -\log(e^{f_{y_i}} / \sum_{j} e^{f_j})$$

where fj denotes the j^{th} element (j ∈ [1, K], K is the number of classes) of the vector of class scores f, and N is the number of training data [40].

3.2.6. Convolutional Neural Network Architecture for Classification

A typical convolutional network is composed of stacked feature stages or layers. Multiple convolutional layers are interspersed by pooling or subsampling layers followed by a multiple fully connected layers and finally a single output layer that performs the classification. Yan LeCun introduced the LeNet architecture, which was initially applied to character recognition tasks. Over the period several revised architectures have been developed on the basis of LeNet architecture such as AlexNet, Clarifai, SPP, VGG, GoogLeNet, NiN. Further details about these architectures and their contributions can be obtained from the following papers [38]. For the current paper, we have developed two architectural models, which will be discussed in the experimental section.

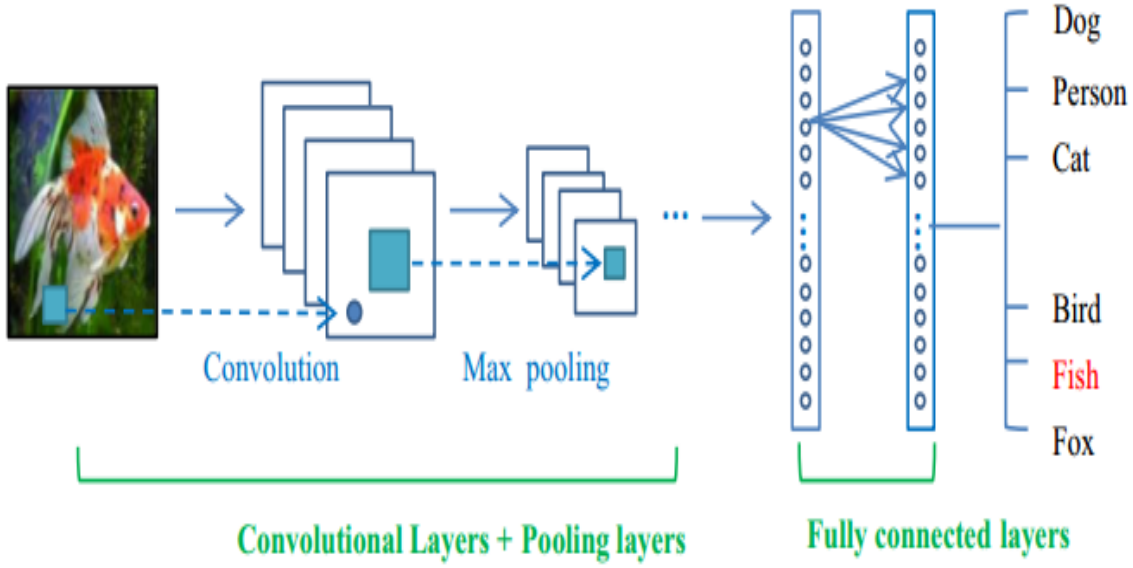

Figure 14. Pipeline of the general CNN architecture [38]

3.2.7. Training Details of CNN

As we have discussed in the different layers in CNN, the convolution and subsampling layers perform feature extraction from the input image while the fully connected layer classifies the image. Training of CNN involves the following steps [41]:

a. **Initialization:** All the filters and weights are initialized to random values.

b. **Forward Propagation:** The convolutional, ReLu, subsampling/pooling layers and the fully connected layers carry on the forward propagation step on the input training image and computes the output probabilities for the different classes.

c. **Error Calculation:** Summation of the different classes in the output layer can be used to compute the total error.

d. **Backpropagation:** It is used to calculate the error gradients of the weights and gradient descent is used to update the weights for output error minimization.

e. **Iteration:** The forward propagation, error calculation, and back propagation steps are repeated for the remaining images in the training set.

4. EXPERIMENT AND RESULTS

In the current set of experiments related to this paper, a regular CPU system with a 2.4 GHz Intel i7 octa-core processor, 12GB RAM, and Ubuntu 16.04 operating system was used.

We have used Deeplearning4j [42], an open source deep learning library for executing deep learning algorithms. The library is written for Java Virtual Machine and Java. It works with both CPUs and GPUs and is powered by its own open source numerical computing library ND4J. The ND4J library provides scientific computing for Java and Scala.

Mini-batch SGD- an optimization technique, a weight decay of 0.00001 – a component of regularization, Xavier weight initialization and a momentum of 0.9 are used throughout our experiments.

The model CNN and SAE architectures developed in this paper were tested on the following three image datasets:

MNIST: MNIST stands for Mixed National Institute of Standards and Technology dataset and was developed by LeCun in the 1980s [9]. It is a dataset of handwritten digits and contains an extensive 60,000 example training set and a 10,000 example test set with different distortions and levels of noise. Each of these examples has 28 x 28 pixel grayscale values, most of them set to zero. The dataset has ten classes for the ten digits - 0 to 9 where digit ‘0’ has a label 10 and digit ‘1’ to ‘9’ have labels 1 to 9, respectively. This dataset is widely used for testing real-world data in relation to pattern recognition and learning techniques. Figure 15 depicts a sample image of MNIST digits.

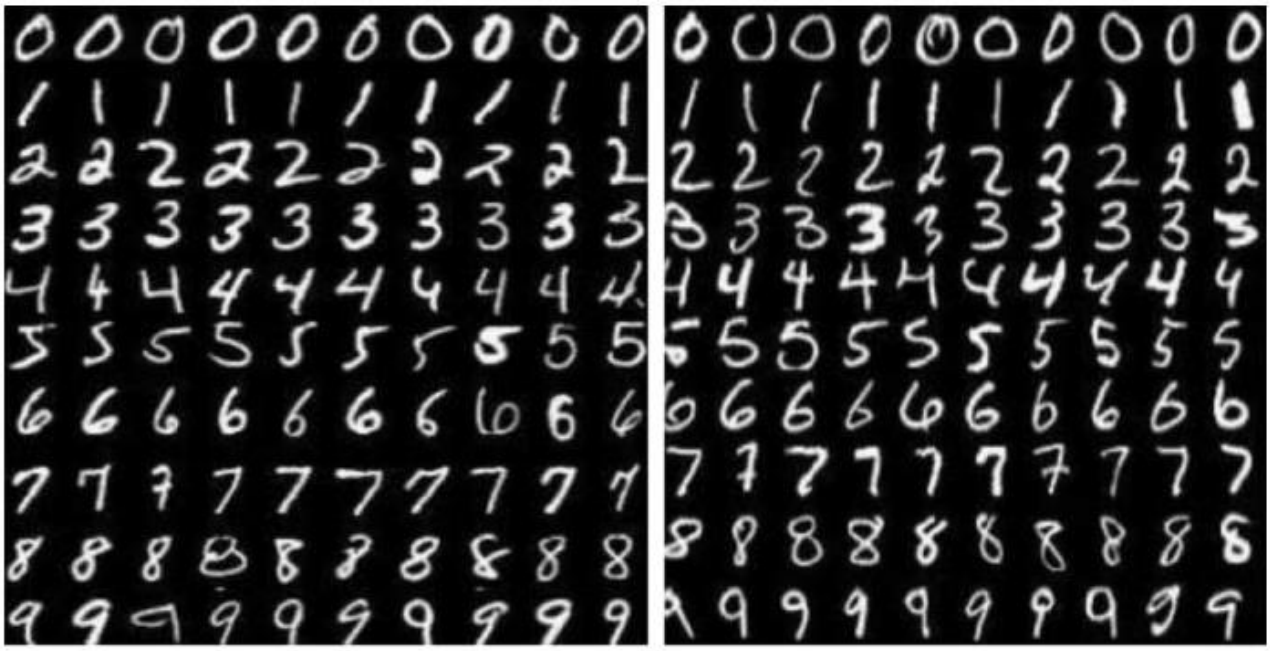

Figure 15. Standard MNIST data a) training samples, b) testing samples [43]

SVHN: SVHN stands for Street View House Numbers dataset [10]. House numbers in Google Street View images have been used to obtain SVHN. Machine learning and object recognition algorithms are developed for this real word image dataset. The dataset has ten classes for the ten digits - 0 to 9 where digit '0' has a label 10 and digit '1' to '9' have labels 1 to 9, respectively. The dataset has a staggering 73,257 digits in the training set, 26,032 digits in the testing set and an additional 531,131 in the extra training dataset. The cropped digits format, with a fixed resolution of 32 x 32 pixels is used for our experimental conditions. Figure 16 depicts a sample image of SVHN digits.

Figure 16. SVHN dataset – Cropped Digits [10]

CIFAR-10: CIFAR stands for Canadian Institute for Advanced Research [11]. 10 refers to the "10 classes" present in this dataset to distinguish it from CIFAR -100 which contains 100 classes. For our current experiment, we will be using the CIFAR-10 dataset. The dataset consists of 50,000 training images and 10,000 test images. Unlike the other two image datasets, CIFAR-10 has 32 x 32 color real world objects such as an airplane, automobile, cat, etc. Figure 17 shows the different classes in the dataset and ten random images from each class.

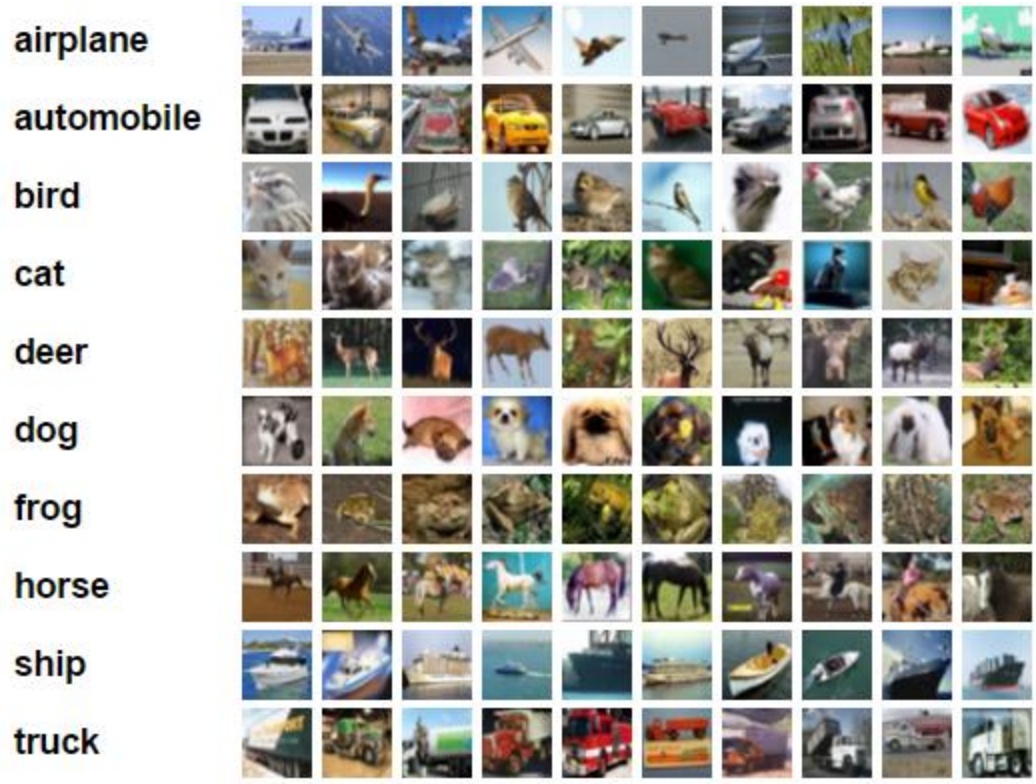

Figure 17. CIFAR-10 dataset [11]

4.1. Convolutional Neural Networks (CNN) Models

Most of the state-of-art-results in object classification have been achieved using the CNN architecture. In the current experiment, we have engineered two CNN models based on the basic architecture.

4.1.1. CNN Model-1 Architecture

Figure 18 represents the overall architecture of Model 1. This model consists of 6 layers -two convolutional layers interspersed by two subsampling layers, a fully connected layer, and an output layer. Input, which is essentially the image, holds the raw pixel values with three color channels R, G, B. In this model, each convolutional layer has a kernel size 5, stride 1 and ReLu, used as an activation function. The first and second convolutional layers have 60 and 90 nodes, respectively. In the subsampling layer, max pooling is used along with the kernel size 5 and stride 1. The fully

connected layer has 500 nodes and uses the activation function ReLu. The final output layer uses the softmax function and contains ten nodes, each corresponding to a class score such as the ten categories of CIFAR-10. The model has been tested on the three datasets and results are presented below.

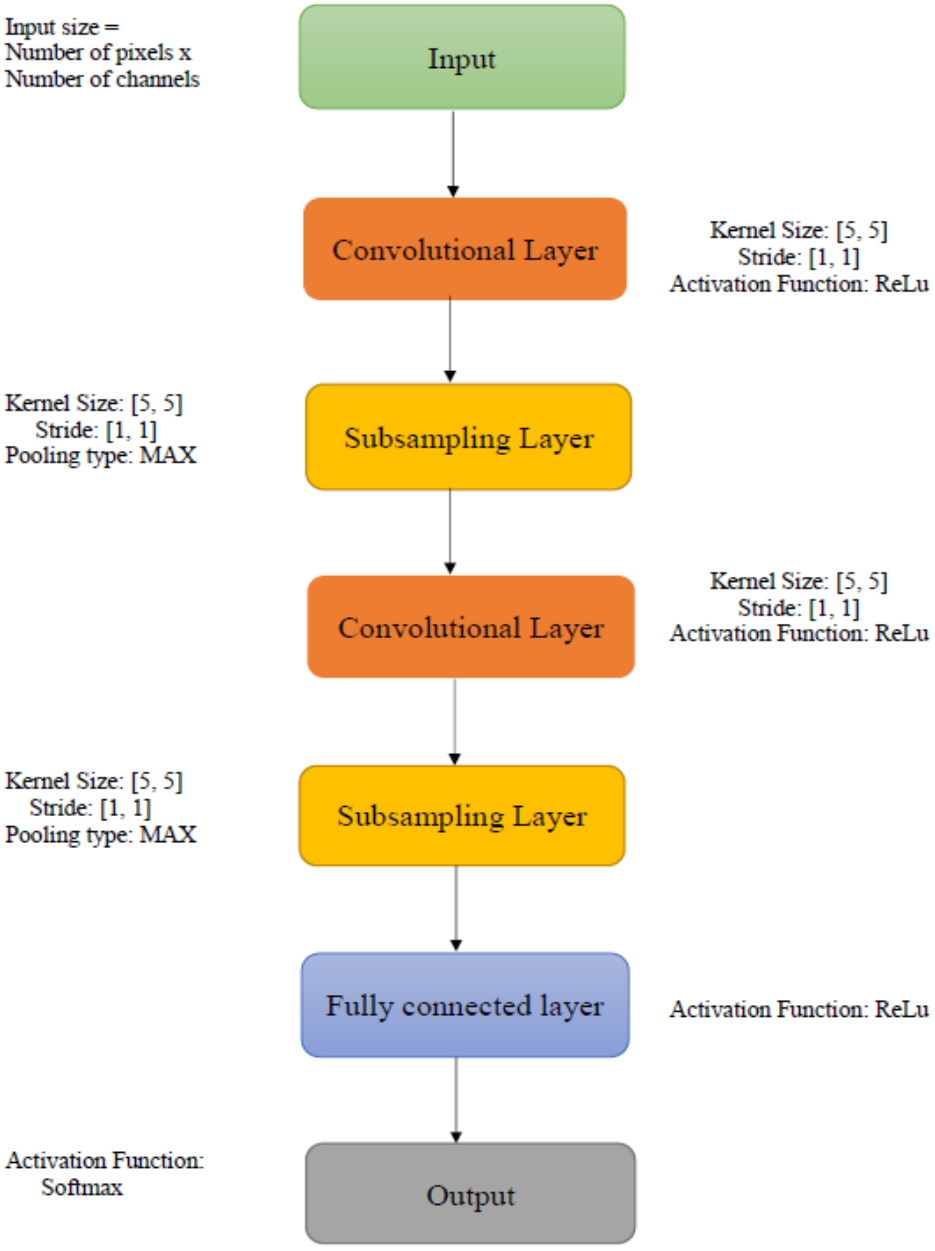

Figure 18. CNN Model-1 architecture

4.1.1.1. Results of CNN Model-1 on MNIST dataset

The following results were achieved on the MNIST dataset after 20 epochs, upon testing the CNN Model 1 with the batch size = 32 and learning rate = 0.001.

Table 1. Results of CNN - Model 1 on MNIST dataset

		TARGET CLASS									
		0	1	2	3	4	5	6	7	8	9
OUTPUT CLASS	0	975	0	0	0	0	0	2	2	1	0
	1	0	1133	0	1	0	0	1	0	0	0
	2	1	0	1023	0	2	0	2	3	1	0
	3	0	0	1	1002	0	4	0	0	3	0
	4	0	0	0	0	979	0	1	0	0	2
	5	1	0	0	4	0	884	1	0	0	2
	6	3	2	0	0	2	1	949	0	1	0
	7	0	2	4	0	0	0	0	1021	0	1
	8	3	0	1	1	0	1	0	0	966	2
	9	1	0	0	0	5	3	0	3	1	996
Overall Accuracy: 99.28%						**Overall Error rate: 0.72%**					

Table 1 shows the results of the run, with the target class and the output class. The numbers 0 to 9 in the target class represent the category in each class. Similarly, categories are represented in the output class. For example, in the first column the model classified 980 images in class 0, out of which 975 images that belong to class zero are classified correctly and the remaining 5 are incorrectly classified out of which 2 images belonging to class 6, 2 belonging to class 7, and 1

belonging to class 8. The green shading represents the number of correctly classified images, and the remaining fields represent incorrectly classified images. Overall, the accuracy and error rate are calculated using the equations below and are presented as a percentage in the blue box.

$$Accuracy = \frac{Number\ of\ correct\ predictions}{Total\ number\ of\ predictions} = \frac{9928}{10000} = 0.9928$$

$$Error\ rate = \frac{Number\ of\ wrong\ predictions}{Total\ number\ of\ predictions} = \frac{72}{10000} = 0.0072$$

An accuracy of 99.28% and error rate of 0.72% was achieved for the MNIST dataset.

4.1.1.2. Results of CNN Model-1 on SVHN dataset

The following results were achieved on the SVHN dataset after 20 epochs, upon testing the CNN Model 1 with the batch size = 16 and learning rate = 0.001.

Table 2. Results of CNN - Model 1 on SVHN dataset

		TARGET CLASS									
		0	1	2	3	4	5	6	7	8	9
OUTPUT CLASS	0	1644	14	23	19	13	16	32	33	45	49
	1	8	2368	23	7	2	28	47	9	23	19
	2	10	10	2388	10	94	21	80	81	159	18
	3	4	10	20	1887	2	34	72	4	15	3
	4	3	18	71	12	2107	8	25	17	20	55
	5	49	167	60	94	11	4867	99	34	8	19
	6	6	34	30	30	13	20	3532	33	21	9
	7	24	31	49	2	14	5	28	1453	23	63
	8	21	10	18	11	13	8	82	21	1454	12
	9	38	39	23	7	43	11	17	34	16	1646
Overall Accuracy: 89.68%						**Overall Error rate: 10.32%**					

The accuracy and error rates were calculated as before. An accuracy of 89.68% and error rate of 10.32% was achieved for the SVHN dataset.

4.1.1.3. Results of CNN Model-1 on CIFAR-10 dataset

The following results were achieved on the CIFAR-10 dataset after 30 epochs, upon testing the CNN Model 1 with the batch size = 16 and learning rate = 0.0015.

Table 3. Results of CNN - Model 1 on CIFAR-10 dataset

		TARGET CLASS									
		0	1	2	3	4	5	6	7	8	9
OUTPUT CLASS	0	786	29	7	22	32	5	11	99	6	7
	1	39	771	4	11	83	6	9	35	4	1
	2	5	0	767	117	7	2	29	23	43	13
	3	7	4	35	845	15	13	30	11	27	40
	4	26	75	24	23	777	8	16	32	0	8
	5	9	11	29	75	8	772	49	10	37	12
	6	15	5	42	109	13	44	580	106	57	15
	7	25	22	10	11	23	10	12	892	16	4
	8	11	1	71	95	2	47	80	32	686	16
	9	26	9	53	393	24	62	50	26	53	260
Overall Accuracy: 71.36%						**Overall Error rate: 28.64%**					

The accuracy and error rates were calculated as before. An accuracy of 71.36% and error rate of 28.64% was achieved for the CIFAR-10 dataset.

4.1.2. CNN Model-2 Architecture

Figure 19 represents the overall architecture of Model 2. The model has a similar architecture as Model 1 with an additional fully connected layer containing 500 nodes. The kernel size, stride and activation functions used are similar to the CNN Model 1. The model has been tested on the three datasets and the results are presented as the following.

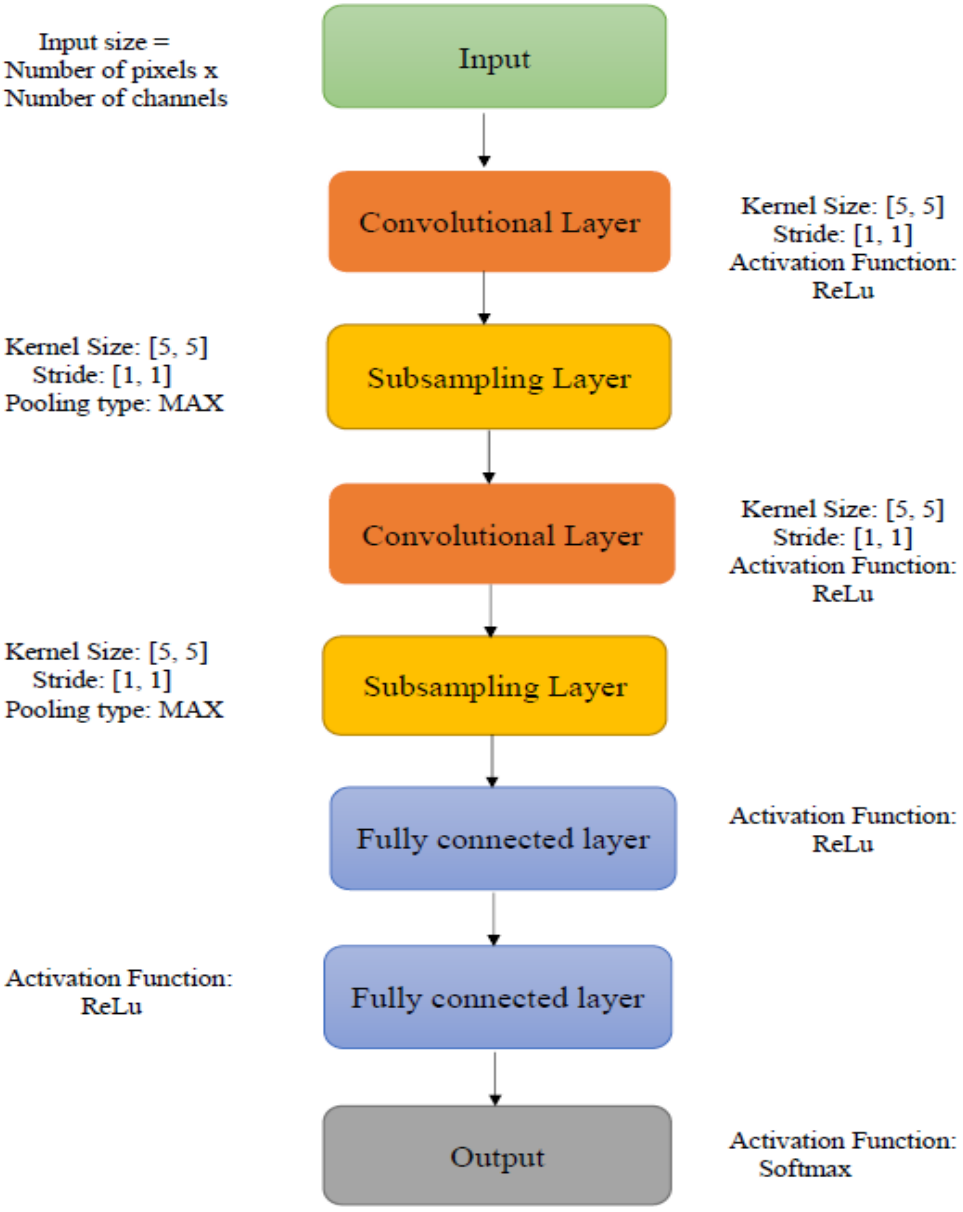

Figure 19. CNN Model-2 architecture

4.1.2.1. Results of CNN Model-2 on MNIST dataset

The following results were achieved on the MNIST dataset after 20 epochs, upon testing the CNN Model 2 with the batch size = 32 and learning rate = 0.001.

Table 4. Results of CNN - Model 2 on MNIST dataset

		TARGET CLASS									
		0	1	2	3	4	5	6	7	8	9
OUTPUT CLASS	0	978	0	0	0	0	0	0	1	1	0
	1	0	1133	0	1	0	0	0	1	0	0
	2	1	0	1022	1	2	0	0	5	1	0
	3	0	0	0	1005	0	4	0	1	0	0
	4	0	0	0	0	978	0	1	0	0	3
	5	2	0	0	6	0	883	1	0	0	0
	6	3	2	0	0	1	3	946	0	3	0
	7	0	0	1	0	0	0	0	1025	1	1
	8	2	0	1	0	0	1	0	1	967	2
	9	1	1	0	1	4	4	0	1	0	997
Overall Accuracy: 99.34%						**Overall Error rate: 0.68%**					

The accuracy and error rates were calculated as before. An accuracy of 99.34% and error rate of 0.68% was achieved for the MNIST dataset.

4.1.2.2. Results of CNN Model-2 on SVHN dataset

The following results were achieved on the SVHN dataset after 20 epochs, upon testing the CNN Model 2 with the batch size = 16 and learning rate = 0.001.

Table 5. Results of CNN - Model 2 on SVHN dataset

		TARGET CLASS									
		0	1	2	3	4	5	6	7	8	9
OUTPUT CLASS	0	1840	10	53	7	3	38	17	53	64	59
	1	5	2090	39	2	5	66	17	3	18	11
	2	10	9	2713	2	21	65	29	22	83	6
	3	6	8	76	1884	2	117	43	4	15	5
	4	9	34	200	1	2138	30	15	22	29	34
	5	23	65	57	20	2	4983	31	21	6	8
	6	2	39	83	27	8	50	3642	22	22	9
	7	10	24	95	1	5	13	9	1525	48	38
	8	9	6	31	2	2	15	35	14	1159	6
	9	13	62	79	2	44	18	4	78	20	1513
Overall Accuracy: 90.22%							**Overall Error rate: 9.78%**				

The accuracy and error rates were calculated as before. An accuracy of 90.22% and error rate of 9.78% was achieved for the SVHN dataset.

4.1.2.3. Results of CNN Model-2 on CIFAR-10 dataset

The following results were achieved on the CIFAR-10 dataset after 30 epochs, upon testing the CNN Model 2 with the batch size = 16 and learning rate = 0.0015.

Table 6. Results of CNN - Model 2 on CIFAR-10 dataset

		TARGET CLASS									
		0	1	2	3	4	5	6	7	8	9
OUTPUT CLASS	0	821	27	12	8	12	7	15	63	6	8
	1	26	833	10	6	36	8	1	17	4	1
	2	1	1	827	61	6	4	22	10	26	14
	3	4	3	70	658	5	20	50	6	35	75
	4	28	103	31	19	770	9	11	34	2	6
	5	7	3	23	33	3	829	48	8	24	31
	6	17	7	57	88	6	47	691	43	54	33
	7	65	21	26	8	18	13	50	767	24	13
	8	9	0	115	54	5	22	84	21	762	21
	9	9	12	64	219	26	64	89	30	80	419
Overall Accuracy: 73.77%						**Overall Error rate: 26.23%**					

The accuracy and error rates were calculated as before. An accuracy of 73.77% and error rate of 26.23% was achieved for the CIFAR-10 dataset.

4.2. Stacked Autoencoder Architectures

Unlike the CNN models, a separate model has been designed for each of the three datasets. Figure 8 represents the complete architecture of the Stacked Auto Encoder for the MNIST dataset. The input, which is essentially the image, holds the raw pixel values with three color channels R, G, B. The final output layer uses the softmax function and contains ten nodes each representing a class. Each layer has a specified number of nodes. The models for the datasets with the number of nodes in each layer are listed in Table 7.

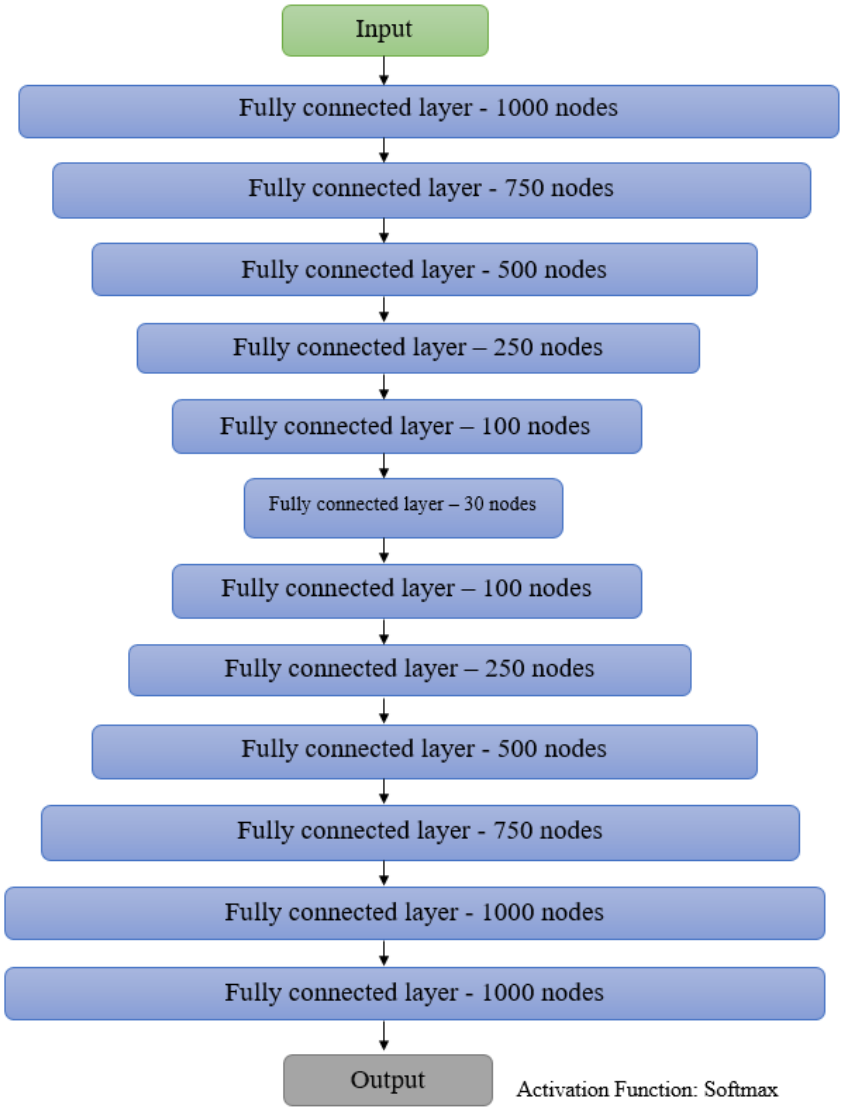

Figure 20. SAE architecture for MNIST dataset

Table 7. Number of nodes in each layer for different datasets

Layers	**MNIST – Number of nodes**	**SVHN – Number of nodes**	**CIFAR-10 – Number of nodes**
Layer 1	1000	5000	5000
Layer 2	500	2500	2500
Layer 3	250	1000	1000
Layer 4	100	500	500
Layer 5	30	250	250
Layer 6	100	100	100
Layer 7	250	50	50
Layer 8	500	100	100
Layer 9	1000	250	250
Layer 10	1000	500	500
Layer 11	-	1000	1000
Layer 12	-	2500	2500
Layer 13	-	5000	5000
Layer 14	-	5000	10000
Layer 15	-	-	5000

Table 7 lists the number of nodes in each layer, which is the distinguishing feature of the models among the datasets. After experimentation, these numbers achieved the best accuracy results.

4.2.1. Results of SAE on MNIST dataset

The following results were achieved on the MNIST dataset after 30 epochs, upon testing the SAE with the batch size = 32 and learning rate = 0.001.

Table 8. Results of SAE on MNIST dataset

		TARGET CLASS									
		0	1	2	3	4	5	6	7	8	9
OUTPUT CLASS	0	821	27	12	8	12	7	15	63	6	8
	1	26	833	10	6	36	8	1	17	4	1
	2	1	1	827	61	6	4	22	10	26	14
	3	4	3	70	658	5	20	50	6	35	75
	4	28	103	31	19	770	9	11	34	2	6
	5	7	3	23	33	3	829	48	8	24	31
	6	17	7	57	88	6	47	691	43	54	33
	7	65	21	26	8	18	13	50	767	24	13
	8	9	0	115	54	5	22	84	21	762	21
	9	9	12	64	219	26	64	89	30	80	419
Overall Accuracy: 98.23%							**Overall Error rate: 1.77%**				

The accuracy and error rates were calculated as before. An accuracy of 98.23% and error rate of 1.77 % was achieved for the MNIST dataset.

4.2.2. Results of SAE on SVHN dataset

The following results were achieved on the SVHN dataset after 100 epochs, upon testing the SAE with the batch size = 16 and learning rate = 0.001.

Table 9. Results of SAE on SVHN dataset

		TARGET CLASS									
		0	1	2	3	4	5	6	7	8	9
OUTPUT CLASS	0	849	134	132	66	101	123	153	35	7	149
	1	14	1968	86	11	32	236	60	1	2	39
	2	15	103	1946	40	114	339	197	14	6	18
	3	17	40	116	1389	26	285	136	2	0	15
	4	27	183	231	29	1549	203	101	4	5	92
	5	27	167	276	71	45	4285	153	2	0	42
	6	30	248	199	193	31	341	3215	11	0	34
	7	66	205	105	19	181	182	73	504	1	308
	8	164	153	151	44	156	195	163	45	428	95
	9	93	293	81	17	208	222	80	9	0	981
Overall Accuracy: 65.74%						**Overall Error rate: 34.26%**					

The accuracy and error rates were calculated as before. An accuracy of 65.74 % and error rate of 34.26 % was achieved for the SVHN dataset.

4.2.3. Results of SAE on CIFAR-10 dataset

The following results were achieved on the CIFAR-10 dataset after 150 epochs, upon testing the SAE with the batch size = 16 and learning rate = 0.0015.

Table 10. Results of SAE on CIFAR-10 dataset

		TARGET CLASS									
		0	1	2	3	4	5	6	7	8	9
OUTPUT CLASS	0	683	46	17	17	50	13	12	94	33	35
	1	97	870	23	30	209	15	29	52	10	49
	2	11	10	623	68	35	25	54	36	68	70
	3	15	10	87	420	14	80	63	21	49	209
	4	66	103	25	18	503	18	10	51	16	30
	5	11	5	16	57	16	393	55	5	80	58
	6	32	12	52	89	29	73	467	85	171	102
	7	84	26	10	21	31	22	33	412	28	13
	8	35	12	81	90	18	116	192	67	579	89
	9	27	29	55	232	38	107	79	38	60	375
Overall Accuracy: 53.25%						**Overall Error rate: 46.75%**					

The accuracy and error rates were calculated as before. An accuracy of 53.25% and error rate of 46.75% was achieved for the CIFAR-10 dataset.

Table 11. Comparison of Results

Deep learning architectures	**MNIST**	**SVHN**	**CIFAR-10**
CNN Model-1 results	99.28 %	89.68 %	71.36 %
CNN Model-2 results	99.34 %	90.22 %	73.77 %
SAE results	98.23 %	65.74 %	53.25 %
State-of-art results	**99.79 %**	**98.31 %**	**96.53 %**

Table 11 presents the comparison of the three architectural models with the state-of-the-art results across the three datasets. As mentioned before two CNN models were devised for the three datasets, but three independent SAE models were developed for each of the three datasets. The state-of-the-art results represent the best results obtained for the three datasets and were obtained from a crowd sourced list [44] of accuracy results for the image classification datasets.

A large number of independent researchers achieved best results with the implementation of the CNN model. However, over the past few years, a number of techniques were adopted to improve accuracies. The models implemented in this paper relied on the standard CNN and SAE architectures and differed in the addition of extra layers and number of nodes. The basic techniques such as Stochastic Gradient Descent (SGD) along with Nestrovs momentum, Xavier weight initialization, and regularization were applied to our models to improve results. Based on the results presented in the table the following inferences can be made:

- Comparison of results of all the there models (CNN model-1, CNN model-2, and SAE model) show that CNN model-2 performs better on all the three datasets.
- Overall, both CNN models performed better in comparison to the SAE models across all datasets.

- Among the three datasets, all the models performed better on the MNIST dataset and the accuracies are closer to the state-of-the-art results.
- Across the three datasets, the CNN model 2 exhibited a negligible increase in accuracies as compared to model 1.
- The SAE model designed for MNIST performed better in comparison to SAE models designed for SVHN and CIFAR-10 datasets.

As mentioned earlier, the SVHN dataset contains an additional 531,131 images as part of the extra training data. Most of the models tested on the SVHN dataset achieved best results by using the extra training dataset. In the present experimental setup, we have trained our models on the SVHN dataset containing the regular 76,032 image datasets. Since we have a limited memory on the machine, the extra training dataset could not be further used. Lesser accuracies on SVHN, obtained in this experiment, can be attributed to this reason.

Though it is a tough proposition to achieve the state-of-the-art results, it is not impossible altogether. The addition of an extra layer in the CNN model 2, did not significantly increase the accuracies. Making the models "deeper" will yield better results [24], but doing so significantly increases the training time and also requires higher processing machines with a good memory. As a solution to this problem, training deep models on GPUs and application of batch normalization [45] will reduce the training time significantly. Techniques such as data augmentation and dropout to reduce overfitting [27], and preprocessing techniques such as ZCA whitening [11] can be introduced to increase accuracies.

5. CONCLUSION AND FUTURE WORK

Deep learning has a profound impact on machine learning technologies. The historical development graph in machine learning has seen many highs and lows owing to the technological difficulties and the limited infrastructure availability. Nevertheless, the evolution of new concepts over the decades has led to improvements over the earlier proposed learning methods. The increasing demand for newer technologies and their cross-functionality across different domains has also catapulted machine learning to an altogether new level. Deep learning found inroads into almost every existing technology with a wide variety of applications. The famous Google Brain Project started in 2011, voice recognition in Apple Siri, and the concept of self-driven cars [46] are few important applications.

In our present paper, based on the standard CNN architecture**,** we have implemented two CNN models, which differ in the number of deep layers and tested these models on the three image datasets namely MNIST, SVHN and CIFAR-10 and achieved an accuracy of 99.34%, 90.22%, and 73.77%, respectively. Similarly, three models based on the SAE architecture were implemented and tested on the three respective datasets, achieving an accuracy of 98.23 %, 65.74 % and 53.25%.

Our present work has opened up new avenues in implementing techniques to increase accuracy. As mentioned earlier, techniques such as dropout and data augmentation can be applied to reduce overfitting. ZCA whitening, a pre-processing technique can be further implemented to improve accuracy. Deeper models can be generated by addition of layers and can be trained on GPUs. All these techniques aim to improve accuracies. Alternatively, the CNN models can be adapted and tested on datasets for natural language processing, sentence classification, and speech recognition. Apart from the regular autoencoders used in this study, it can also be extended to

denoising autoencoders, sparse autoencoders, and variational autoencoders. Comparative analyses of models can also be done by testing them on different image datasets.

REFERENCES

[1] Saloky, Tomáš, and Jaroslav Šeminský. "Artificial Intelligence and Machine Learning." (2005). http://uni-obuda.hu/conferences/SAMI2005/SALOKY.pdf

[2] Sugomori, Yusuke. "*Java Deep Learning Essentials*." Birmingham: Packt Publishing Limited, 2016.

[3] Kurian, M.Z. "Various Object Recognition Techniques for Computer Vision." *Journal of Analysis and Computation* 7, no. 1 (2011): 39-47.

[4] Bengio, Yoshua, Aaron Courville, and Pascal Vincent. "Representation Learning: A Review and New Perspectives." *IEEE transactions on pattern analysis and machine intelligence* 35, no. 8 (2013): 1798-828.

[5] Deng, Li, and Dong Yu. "Deep Learning." *Signal Processing* 7 (2014): 3-4.

[6] Le, Quoc V. "A Tutorial on Deep Learning Part 1: Nonlinear Classifiers and the Backpropagation Algorithm." (2015). https://cs.stanford.edu/~quocle/tutorial1.pdf

[7] Mo, Dandan. "A Survey on Deep Learning: One Small Step toward Ai." *Dept. Computer Science, Univ. of New Mexico, USA* (2012).

[8] Bengio, Yoshua, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. "Greedy Layer-Wise Training of Deep Networks." *Advances in neural information processing systems* 19 (2007): 153.

[9] LeCun, Yann, Corinna Cortes, and Christopher JC Burges. "The Mnist Database of Handwritten Digits." (1998). http://yann.lecun.com/exdb/mnist/

[10] Netzer, Yuval, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. "Reading Digits in Natural Images with Unsupervised Feature Learning." *In NIPS Workshop on Deep Learning and Unsupervised Feature Learning*, 2011.

[11] Krizhevsky, Alex, and Geoffrey Hinton. "Learning Multiple Layers of Features from Tiny Images." *Tech. Rep. 001*, Department of Computer Science, University of Toronto, 2009.

[12] Zhu, Xingquan. *Knowledge Discovery and Data Mining: Challenges and Realities: Challenges and Realities*. Igi Global, 2007.

[13] Dogra, Ashish Kumar, and TanujWala. "A Review Paper on Data Mining Techniques and Algorithms." *International Journal of Advanced Research in Computer Engineering & Technology (IJARCET)* 4, no. 5 (May 2015).

[14] Fayyad, Usama, Gregory Piatetsky-Shapiro, and Padhraic Smyth. "From Data Mining to Knowledge Discovery in Databases." *AI magazine* 17, no. 3 (1996): 37.

[15] Lin, Yuanqing, Fengjun Lv, Shenghuo Zhu, Ming Yang, Timothee Cour, Kai Yu, Liangliang Cao, and Thomas Huang. "Large-Scale Image Classification: Fast Feature Extraction and Svm Training." Paper presented at the Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on, 2011.

[16] Jain, Maneela, and Pushpendra Singh Tomar. "Review of Image Classification Methods and Techniques." Paper presented at the International Journal of Engineering Research and Technology, 2013.

[17] Kotsiantis, Sotiris B, I Zaharakis, and P Pintelas. "Supervised Machine Learning: A Review of Classification Techniques." 2007. *Informatica* 31 (2007): 249–268.

[18] Kim, Jinho, Byung-Soo Kim, and Silvio Savarese. "Comparing Image Classification Methods: K-Nearest-Neighbor and Support-Vector-Machines." *Ann Arbor* 1001 (2012): 48109-2122.

[19] Weis, Martin, Till Rumpf, Roland Gerhards, and Lutz Plümer. "Comparison of Different Classification Algorithms for Weed Detection from Images Based on Shape Parameters." *Bornimer Agrartechn. Ber* 69 (2009): 53-64.

[20] Appel, Ron, Thomas J Fuchs, Piotr Dollár, and Pietro Perona. "Quickly Boosting Decision Trees-Pruning Underachieving Features Early." Paper presented at the ICML (3), 2013.

[21] Marée, Raphaël, Pierre Geurts, Giorgio Visimberga, Justus Piater, and Louis Wehenkel. "A Comparison of Generic Machine Learning Algorithms for Image Classification." In *Research and Development in Intelligent Systems Xx*. 169-82: Springer London, 2004.

[22] Tan, Pan-Ning, Michael Steinbach, and Vipin Kumar. "Classification: Basic Concepts, Decision Trees, and Model Evaluation." *Introduction to data mining* 1 (2006): 145-205.

[23] Cilimkovic, Mirza. "Neural Networks and Back Propagation Algorithm." *Institute of Technology Blanchardstown, Blanchardstown Road North Dublin* 15 (2015).

[24] LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton. "Deep Learning." *Nature* 521, no. 7553 (2015): 436-44.

[25] Sutskever, Ilya, James Martens, George E Dahl, and Geoffrey E Hinton. "On the Importance of Initialization and Momentum in Deep Learning." *ICML (3)* 28 (2013): 1139-47.

[26] Glorot, Xavier, and Yoshua Bengio. "Understanding the Difficulty of Training Deep Feedforward Neural Networks." Paper presented at the Aistats, 2010.

[27] Srivastava, Nitish, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting." *Journal of Machine Learning Research* 15, no. 1 (2014): 1929-58.

[28] Baldi, Pierre. "Autoencoders, Unsupervised Learning, and Deep Architectures." *ICML unsupervised and transfer learning* 27, no. 37-50 (2012): 1.

[29] Meyer, David. "Introduction to Autoencoders." (2015). http://www.1-4-5.net/~dmm/papers/intro_to_autoencoders.pdf

[30] Bengio, Yoshua. "Learning Deep Architectures for Ai." Foundations and trends® in Machine Learning 2, no. 1 (2009): 1-127.

[31] Ng, Andrew, Jiquan Ngiam, Chuan Yu Foo, Yifan Mai, Caroline Suen, Adam Coates, Andrew Maas, et al. "Deep Learning Tutorial." 2015. http://ufldl.stanford.edu/wiki/index.php/UFLDL_Tutorial

[32] Hinton, Geoffrey E, and Ruslan R Salakhutdinov. "Reducing the Dimensionality of Data with Neural Networks." *Science* 313, no. 5786 (2006): 504-07.

[33] Vincent, Pascal, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. "Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion." *Journal of Machine Learning Research* 11, no. Dec (2010): 3371-408.

[34] Schulz, Hannes, and Sven Behnke. "Deep Learning." *KI-Künstliche Intelligenz* 26, no. 4 (2012): 357-63.

[35] Van Doorn, Joost. "Analysis of Deep Convolutional Neural Network Architectures." (2014). http://referaat.cs.utwente.nl/conference/21/paper/7438/analysis-of-deep-convolutional-neural-network-architectures.pdf

[36] Arnold, Ludovic. "Learning Deep Representations: Toward a Better New Understanding of the Deep Learning Paradigm." Université Paris Sud-Paris XI, 2013.

[37] LeCun, Yann, Léon Bottou, Yoshua Bengio, and Patrick Haffner. "Gradient-Based Learning Applied to Document Recognition." *Proceedings of the IEEE* 86, no. 11 (1998): 2278-324.

[38] Guo, Yanming, Yu Liu, Ard Oerlemans, Songyang Lao, Song Wu, and Michael S Lew. "Deep Learning for Visual Understanding: A Review." *Neurocomputing* 187 (2016): 27-48.

[39] Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet Classification with Deep Convolutional Neural Networks." Paper presented at the Advances in neural information processing systems, 2012.

[40] Liu, Weiyang, Yandong Wen, Zhiding Yu, and Meng Yang. "Large-Margin Softmax Loss for Convolutional Neural Networks." Paper presented at the Proceedings of The 33rd International Conference on Machine Learning, 2016.

[41] Bouvrie, Jake. "Notes on convolutional neural networks", MIT CBCL Tech Report (2006). 38- 44.

[42] Team, DJD. "Deeplearning4j: Open-Source Distributed Deep Learning for the Jvm." *Apache Software Foundation License* 2.

[43] Rezai, Abdalhossein, Parviz Keshavarzi, and Reza Mahdiye. "A Novel Mlp Network Implementation in Cmol Technology." *Engineering Science and Technology, an International Journal* 17, no. 3 (2014): 165-72.

[44] Benenson, Rodrigo. "What Is the Class of This Image?" (2016). http://rodrigob.github.io/are_we_there_yet/build/classification_datasets_results.html

[45] Ioffe, Sergey, and Christian Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift." *arXiv preprint arXiv: 1502.03167* (2015).

[46] Parloff, Roger, and Justin Metz. "Why Deep Learning Is Suddenly Changing Your Life." (2016). Published electronically September 28, 2016. http://fortune.com/ai-artificial-intelligence-deep-machine-learning/