A GENERALIZED DEEP LEARNING MODEL FOR DENOISING IMAGE DATASETS

FINAL REPORT

Submitted in partial fulfilment of the requirements of the degree of

MASTER OF TECHNOLOGY
In
ELECTRONICS AND COMMUNICATION ENGINEERING WITH SPECIALIZATION IN WIRELESS TECHNOLOGY

Submitted by

PRANAV E

Reg. No. 45318011

DIVISION OF ELECTRONICS AND COMMUNICATION

SCHOOL OF ENGINEERING

COCHIN UNIVERSITY OF SCIENCE AND TECHNOLOGY

KOCHI- 682022

APRIL 2020

DIVISION OF ELECTRONICS ENGINEERING

SCHOOL OF ENGINEERING

COCHIN UNIVERSITY AND TECHNOLOGY

KOCHI- 682022

BONAFIDE CERTIFICATE

It is certified that the report titled "**A GENERALIZED DEEP LEARNING MODEL FOR DENOISING IMAGE DATASETS**" is a bonafide record of the MAIN PROJECT done by **PRANAV E** and is forwarded towards the partial fulfillment of the requirements for the award of M. Tech degree in Electronics and Communication Engineering of the Cochin University of Science and Technology.

Dr. R Gopikakumari	**Dr. Supriya M.H**	**Dr. Babita Roslind Jose**
Professor	Professor	Professor
Internal Guide	Head of the Department,DOE External Guide	Head of the Department,SOE

DISSERTATION APPROVAL FOR M.TECH

This dissertation entitled "**A GENERALIZED DEEP LEARNING MODEL FOR DENOISING IMAGE DATASETS**" by **Pranav E** is recommended for the award of the degree of Master of Technology.

Members of the Examination Committee

_____

_____

_____

_____

Date: _____

Place: _____

DECLARATION

I, Pranav E, student of M.Tech in Electronics and Communication Engineering with specialization in Wireless Technology hereby declare that this written submission represents my ideas in my own words and where others' ideas or words have been included, I have adequately cited and referenced the original sources. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission. I understand that any violation of the above will be cause for disciplinary action by the University and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

Place: Signature:

Date: Name: Pranav E

Reg. No.: 45318011

ACKNOWLEDGEMENT

First and foremost, I thank The Almighty for His blessings showered on me for making me capable of doing this project work. I would like to express my sincere thanks to **Dr. George Mathew**, Principal, SOE for his kind support.

I am indebted to my project guide **Dr. R.Gopika Kumari**, Professor, for her guidance and suggestions. I also extend my gratitude toward our Head of Department**, Dr. Babita Roslind Jose**, Professor for her leadership and guidance.

I extend my deep respect and gratitude to **Dr. Supriya M.H**, Head of Department, Department of Electronics, CUSAT for her invaluable guidance and motivation in this endeavour

I am indebted to all teaching and non-teaching staff of the Division of Electronics and Communication Engineering, SOE for their cooperation and support. I also wish to express my sincere thanks to my family and all my friends for their goodwill and constructive ideas.

PRANAV E

ABSTRACT

The rapid growth of artificial intelligence has contributed a lot to the technology world. As the traditional algorithms failed to meet the human needs in real time, Machine learning and deep learning algorithms have gained great success in different applications such as classification systems, recommendation systems, pattern recognition etc. Application of machine learning and deep learning algorithms with images can be challenging. Images can be noisy, and you likely want to have this noise removed. Traditional noise removal filters can be used for this purpose, but they're not data-specific and hence may remove more noise than you wish, or leave too much when you want it gone. Deep convolutional neural networks can be used to the noise removal filter based on the dataset you wish noise to disappear from. The goal of this work is to find whether a deep convolutional neural network (DCNN) performs better than the existing algorithms for image denoising. The proposed model is a generalized DCNN model which can recognize and classify any type of noisy image given. Two types of model were compared where one model 1 uses the Adam optimizer and model 2 uses the Stochastic Gradient Descent (SGD) optimizer. The image dataset used here is MNIST handwritten dataset, which is trained, tested and validated with both the models by adding 3 different types of noise viz, Gaussian Noise, Salt and Pepper Noise and Poisson Noise. More accuracy and better results were given by the model 2 which uses the SGD optimizer.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

TABLE **PAGE NO.**

CHAPTER 1
INTRODUCTION

Artificial Intelligence is the science of the 21st century. Artificial Intelligence (AI) is defined as the ability for a machine to “think or act humanly or rationally”. Machines are now able to process the vast amount of data in real time and respond accordingly. However, these machines with high IQ (Intelligence Quotient) were always lacking Emotional Intelligence (EI)/ Emotional Quotient (EQ). As technology progresses and the world becomes more and more virtual, there is a fear that we will lose the human connection and communication; but what if our devices could replace those interactions?

Noise accumulation in images is caused at different stages of capturing an image, which includes acquisition, quanti-zation, formatting and compression. In some cases, noise accumulation is unavoidable due to environmental constraints, for example, there is a maximum radiation limit for computer tomography (CT) scan which results in low-contrast images,or if we take the case of outdoor surveillance, haze or other weather conditions result in inconsistency in image quality.Noisy medical imaging may lead to inaccurate diagnosis of the patient. On the other hand, if satellite images are not denoised, several crucial geographical information will be lost. Denoising is equally important for mission-critical defence applications and for non-critical industrial applications .Noise classification allows us to identify the most probable noise distribution present in a noisy image after which an optimal denoising algorithm may be applied to denoise the image. The method is also used across other signal processing disciplines. In digital image processing, whether image noise can be effectively filtered out or not will directly affect the subsequent processing such as object segmentation, edge detection, feature extraction and so on. Therefore, filtering the image noise is thought as a meaningful work.For the 2D visual signals, we usually have two forms of filtering which are based on spatial domain and frequency domain.

Image filtering in spatial domain closely relates to the pixel intensity and its neighbors; meanwhile, image filtering in frequency domain utilizes the coefficients of multiple frequencies after visual signal decomposition. For these two domains, the corresponding denoising methods can be categorized into spatial domain-based or frequency domain-based methods. The spatial domain-based methods directly tackle the intensity of each pixel of an

image. The frequency domain-based methods handle the issue of image denoising by adjusting corresponding coefficients of multiple frequencies after image decomposition in frequency domain. The inverse transformation is usually accomplished to reconstruct the image using those modified coefficients and achieve the purpose of image denoising.

In the field of computer science, machine learning is one of the emerging technologies that is considered to have an impact of 90% in the next 4 years. Deep learning, a subset of machine learning uses artificial neural network, which is an algorithm inspired from the human brain. Convolutional Neural Network (CNN) is a class of deep neural network that uses convolution as the mathematical operation. As the dataset consists of images, the system uses a 2D CNN for the recognition task. The proposed deep convolutional neural network model classifies and recognizes 10 different classes of images of handwritten digits which have been added with noise. However, because these traditional denoising methods based on image filtering could not tune the parameters during filtering, in this project, I introduce a linear CNN for image noise removal.

CHAPTER 2

LITERATURE SURVEY

Author [1] demonstrates the possibilities of using denoising convolutional neural networks to solve one of the most difficult tasks that developers face when performing the transfer of graphical information in infocommunication systems - denoising. Unlike traditional algorithms, denoising convolutional neural networks have architecture features that allow them to perform image filtering effectively with unknown noise level. They use denoising convolutional neural networks to generate a correction signal in the infocommunication system, which transmits a noisy image. The article proposes to use denoising convolutional neural networks to generate a correction signal in the infocommunication system, which transmits a noisy image. Such pre-trained correction elements on a large and diverse image dataset can be easily fine-tuned for special filtering tasks.

Author [2] propose an improved training loss function for Denoising Auto-encoders based on Method noise and entropy maximization principle, with residual statistics as constraint conditions. They compare it with conventional denoising algorithms including original Denoising Auto-encoders, BM3D, total variation (TV) minimization, and non-local mean (NLM) algorithms. Experiments indicate that the Improved Denoising Auto-encoders introduce less non-existent artifacts and are more robustness than other state-of-the-art denoising methods in both PSNR and SSIM indexes, especially under low SNR.

Author [3] proposed a vision based system, a Convolutional Neural Network (CNN) model, to identify six different sign languages from the images captured. The two CNN models developed have different type of optimizers, the Stochastic Gradient Descent (SGD) and Adam.

Author [4] uses a CNN model in deep learning for image denoising. Compared with traditional image denoising methods such as average filtering, Wiener filtering and median filtering, the advantage of using this CNN model is that the parameters of this model can be optimized through network training; whereas in traditional image denoising, the parameters of these algorithms are fixed and cannot be adjusted during the filtering, namely, lack of adaptivity. They designed and implemented the denoising method based on a linear CNN

model. Their experimental results show that the proposed CNN model can effectively remove Gaussian noise and improve the performance of traditional image filtering methods significantly.

Author [5] first aimed to reply to probably the most critical design questions through theoretical analysis and extensive experiments: How deep and wide a deep denoiser should and can be? Does denoising performance get improved by using residual learning? Can and should we switch from the region based to image-based models? And second, based on their analysis, they designed a deep neural network for natural image denoising which was hundred-layer deep, exploited both internal and external residual learning, and was trained in an image-based fashion. Their deep denoiser achieved the state-of-the-art results quantitatively and qualitatively on multiple datasets including one with more than 10,000 images.

Author [6] provides answers to several questions related to WT technique such as what WT is, how and why WT emerged, what WT types currently available. The main advantages like noise reduction and compression of WT are also explained in this study. A set of MATLAB experiments were carried out in order to illustrate the use of WT as a signal denoising tool. Analysis on different signals contaminated with noise are performed. Different types of thresholding and mother wavelets were applied and the outcome of the experiments indicate that Daubechies family along with the soft thresholding technique suited our application the most. The study proves that choosing the right thresholding technique and wavelet family is vital for the success of signal denoising applications.

CHAPTER 3

MACHINE LEARNING AND DEEP LEARNING

The key concept that we will need to think about for our machines is learning from data, since data is what we have. However, it isn't too large a step to put that into human behavioural terms, and talk about learning from experience. Hopefully, we all agree that humans and other animals can display behaviours that we label as intelligent by learning from experience. Learning is what gives us flexibility in our life; the fact that we can adjust and adapt to new circumstances, and learn new tricks, no matter how old a dog we are! The important parts of animal learning are remembering, adapting, and generalising: recognising that last time we were in this situation (saw this data) we tried out some particular action (gave this output) and it worked (was correct), so we'll try it again, or it didn't work, so we'll try something different. The last word, generalising, is about recognising similarity between different situations, so that things that applied in one place can be used in another. This is what makes learning useful, because we can use our knowledge in lots of different places. Of course, there are plenty of other bits to intelligence, such as reasoning, and logical deduction, but we won't worry too much about those. We are interested in the most fundamental parts of intelligence - learning and adapting - and how we can model them in a computer. There has also been a lot of interest in making computers reason and deduce facts. This was the basis of most early Artificial Intelligence, and is sometimes known as symbolic processing because the computer manipulates symbols that reflect the environment. In contrast, machine learning methods are sometimes called sub symbolic because no symbols or symbolic manipulation are involved.

3.1 Artificial Intelligence

The field of Artificial Intelligence encompasses multiple sub-fields including Machine Learning, natural language processing, data mining, and so on. It can be defined as the art, science and engineering of making intelligent agents, machines and programs. The field aims to provide solutions for one simple yet extremely tough objective, "Can machines think, reason, and act like human beings?" AI in fact existed as early as the 1300s when people started asking such questions and conducting research and development on building tools that could work on concepts instead of numbers like a calculator does. Progress in AI took place

in a steady pace with discoveries and inventions by Alan Turing, McCullouch, and Pitts Artificial Neurons. AI was revived once again after a slowdown till the 1980s with success of expert systems, the resurgence of neural networks thanks to Hopfield, Rumelhart, McClelland, Hinton, and many more. Faster and better computation thanks to Moore's Law led to fields like data mining, Machine Learning and even Deep Learning come into prominence to solve complex problems that would otherwise have been impossible to solve using traditional approaches. Figure 3.1 shows some of the major facets under the broad umbrella of AI.

Some of the main objectives of AI include emulation of cognitive functions also known as cognitive learning, semantics, and knowledge representation, learning, reasoning, problem solving, planning, and natural language processing. AI borrows tools, concepts, and techniques from statistical learning, applied mathematics, optimization methods, logic, probability theory, Machine Learning, data mining, pattern recognition, and linguistics. AI is still evolving over time and a lot of innovation is being done in this field including some of the latest discoveries and inventions like self-driving cars, chatbots, drones, and intelligent robots.

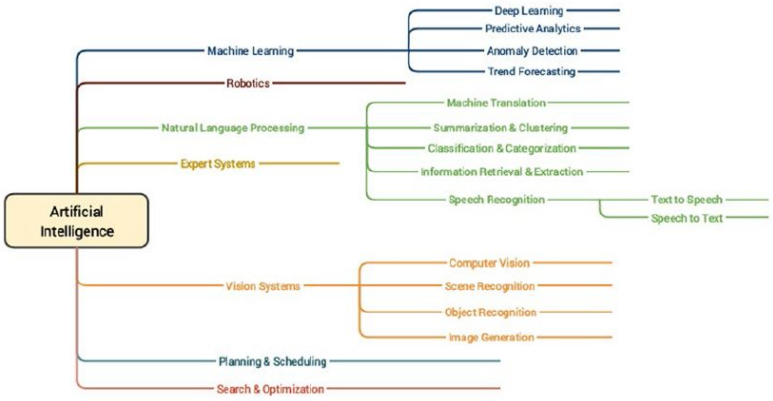


Fig 3.1: Some of the major facets under the broad umbrella of AI

3.2 Machine Learning

Why Machine Learning?

Computers, while being extremely sophisticated and complex devices, are just another version of our well known idiot box, the television! "How can that be?" is a very valid question at this point. Let's consider a television or even one of the so-called smart TVs,

which are available these days. In theory as well as in practice, the TV will do whatever you program it to do. It will show you the channels you want to see, record the shows you want to view later on, and play the applications you want to play! The computer has been doing the exact same thing but in a different way. Traditional programming paradigms basically involve the user or programmer to write a set of instructions or operations using code that makes the computer perform specific computations on data to give the desired results. Figure 3.2 depicts a typical workflow for traditional programming paradigms.

The traditional programming paradigm is quite good and human intelligence and domain expertise is definitely an important factor in making data-driven decisions we need Machine Learning to make faster and better decisions. The Machine Learning paradigm tries to take into account data and expected outputs or results if any and uses the computer to build the program, which is also known as a model. This program or model can then be used in the future to make necessary decisions and give expected outputs from new inputs. Figure 3.3 shows how the Machine Learning paradigm is similar yet different from traditional programming paradigms.

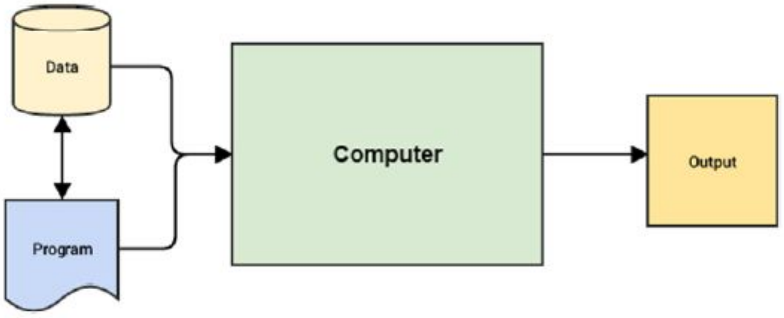


Fig 3.2: Traditional programming paradigms.

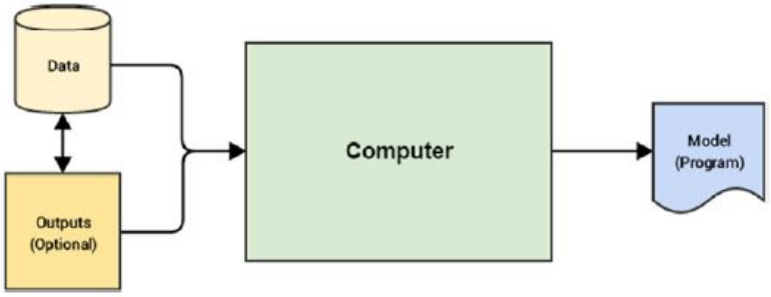


Fig 3.3: Machine Learning Paradigm

Figure 3.3 reinforces the fact that in the Machine Learning paradigm, the machine, in this context the computer, tries to use input data and expected outputs to try to learn inherent patterns in the data that would ultimately help in building a model analogous to a computer program, which would help in making data-driven decisions in the future (predict or tell us the output) for new input data points by using the learned knowledge from previous data points (its knowledge or experience).

Formal Definition:

The best way to define Machine Learning would be to start from the basics of Machine Learning as defined by renowned professor Tom Mitchell in 1997. The idea of Machine Learning is that there will be some learning algorithm that will help the machine learn from data. Professor Mitchell defined it as follows.

"A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E."

The tasks that can be solved by machine learning are very diverse. In general, machine learning techniques prove extremely useful to execute tasks for which no explicit and/or viable programming approach exists to date, e.g. classification, 10 regressions, pattern recognition, automatic language translation, anomaly detection, etc. As diverse as the task to perform may be, a machine learning algorithm can be mathematically described by the map

$$F : x \in X \subseteq R\,n \rightarrow y \in Y \subseteq R\,m \qquad (3.1)$$

wherein x is a data vector whose components are the features describing the task to be solved, y is the output produced by the machine learning algorithm representing the answer to the problem at hand, X and Y are the sets in which x and y may vary. It is important not to confuse the task performed by a machine learning technique with the action of learning. The former is the final objective of the algorithm, while the latter is the method that is used to carry out the task. In order to evaluate the ability of a machine learning algorithm to solve the assigned task, i.e. to produce output vectors close to the desired ones, a performance criterion P must be defined. Several performance measures can be considered and typically the best choice is application-dependent. Typical choices are the mean squared error (MSE) or the cross-entropy functions.

The last component of a machine learning algorithm to be introduced is the experience E, i.e. the knowledge and data that the algorithm can exploit to carry out the task. Machine learning algorithms typically experience a set of data points ST R, called training set. Depending on the information contained in S, machine learning algorithms can be grouped into two main categories:

• **Unsupervised learning:** the experienced data training set ST R contains only input features, i.e. ST R = {x1, . . . , xN }. Based on ST R, the machine learning algorithm must be able to extrapolate the statistical structure of the input or any other information needed to carry out the desired task.

• **Supervised learning:** the experienced data training set ST R contains both input features and the corresponding desired outputs, referred to as labels or targets, i.e. S = {(x1, y1), . . . ,(xN , yN)}. Thus, in supervised learning, the training set provides a series of examples to instruct the algorithm how to behave when some specific inputs are considered.

In both supervised and unsupervised learning, the available dataset is fixed. Classical decision/estimation theory assumes that the probability distributions of the output vector given the input p(y|x) and that of the input vector p(x) are known. Instead, machine learning does not need this assumption and is able to operate based only on some realizations of the underlying distributions, even though the distributions themselves are not known.

i. Overfitting and Underfitting

Any machine learning algorithm experiences a training set ST R that contains some input features x1, . . . , xN . In the supervised scenario, each input feature is also accompanied by the corresponding desired output. While this information is essential to configure the learning scheme, the key problem of any machine learning algorithm is to perform well on previously unseen inputs. This means that the algorithm needs to be able to grasp from ST R a general rule to produce a suitable output y also when $\tilde{x} \in X/$. This is referred to as the algorithm generalization capability. During the training phase, the information in the training set is used to set the algorithm parameters in order to minimize any desired performance metric. As it will be detailed in the sequel, this amounts to solving an optimization problem. Machine learning however, is fundamentally different from optimization theory: its ultimate goal is to make the algorithm able to generalize well to new data inputs. In order to evaluate its

generalization capability, after the algorithm has been designed as a result of the training phase, its performance is tested over a new set of different inputs ST , called the test set. For any given error measure, the error evaluated over the test set is called generalization error or test error. Similarly, the error evaluated over the training set is called the training error. Clearly, in order for the algorithm to generalize well, the data samples in the training set ST R and in the test set ST need to be drawn from the same distribution, called data generating distribution, even though they should be drawn independently of each other. Clearly, the expected generalization error will be larger than the expected training error, and the gap between the two is called the generalization gap. Thus, minimizing the training error can be regarded as a necessary but not sufficient condition to obtain also a low generalization error. A machine learning algorithm is said to be:

- **Underfitting** if it is not able to make the error over the training set small.
- **Overfitting** if it is not able to make the gap between the training and test error small.

The factor that controls whether overfitting or underfitting occurs is the capacity of the algorithm, i.e. the ability of the algorithm to properly fit the training set. Intuitively, the capacity of the algorithm is related to the degrees of freedom or parameters that can be chosen when designing the algorithm. If the algorithm does not have enough free parameters, it will not have enough degrees of freedom to capture the structure of the training set and the algorithm will underfit.

Instead, the overfitting scenario is subtler. One may think that increasing the number of free parameters will always lead to better performance, and that an upper limit is represented only by the computational complexity that we can sustain. This is, however, not the case. If the algorithm has too many degrees of freedom, it will learn the structure of the training set too well, memorizing specific properties that are peculiar only to the training set, but that do not hold in general. As a result, there is an optimal capacity that a machine learning algorithm should have to minimize the generalization gap.

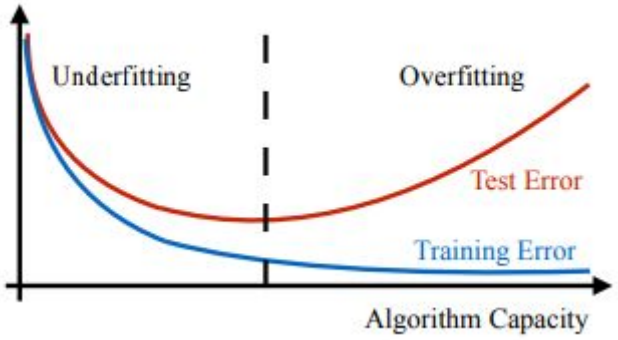


Figure 3.4 : Typical behaviors of the training and test errors

As shown in Fig. 3.4, the training error decreases with the algorithm capacity, asymptotically reaching its minimum value. Instead, the test error has a U-shaped behavior, following the training error up to a capacity value, and then increasing, thereby originating the generalization gap. Fundamental results from statistical learning theory have established that the generalization gap is bounded from above, with the upper bound increasing for larger model capacity, and decreasing for larger training sets. On the other hand, a lower-bound to both the training and test error is given by the well-known Bayes error, i.e. the error obtained by an oracle with access to the true underlying distribution sampling from which the training and test set are obtained.

Another way to interpret the phenomenon of overfitting is to observe that any finite training set will also contain atypical realizations of the underlying distribution, that should be overlooked or given little importance when adjusting the algorithm parameters. However, if too many parameters to optimize are available, the algorithm will try to perfectly fit the complete training set, thus originating the overfitting phenomenon. This concept is illustrated in the example shown in Fig. 5, where it is assumed that a machine learning classifier must output a decision boundary to separate objects belonging to two different classes. It can be seen how a linear decision boundary is not able to properly separate the samples in the training set, thus causing underfitting. On the other hand, having enough degrees of freedom, one can design a complex boundary to perfectly separate the samples in the training set, even those samples that happen to be surrounded by samples of the other class. However, this leads to including in both decision regions areas that are likely to contain samples from the

wrong class, thus causing overfitting. Instead, the curved, but more regular, decision region in the middle better captures the structure of the underlying distribution.

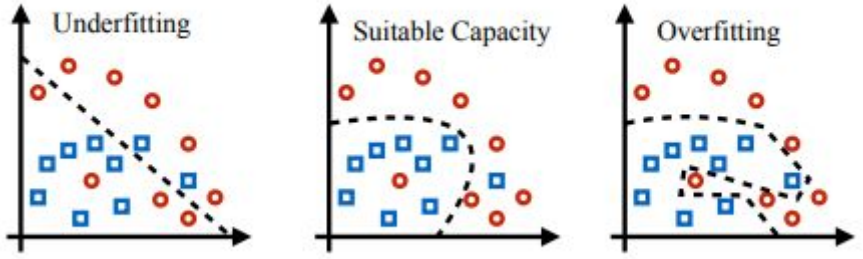


Figure 3.5 : Three possible decision boundaries for a classification problem. The left and right figures show the underfitting and overfitting scenarios. The middle figure shows classifier with the proper capacity.

It is interesting to observe that choosing the decision boundary in the middle illustration of Fig. 3.5 is in agreement with the Occam's razor principle, stating that among different and equally motivated explanations of a phenomenon, one should choose the simplest one. Of course one should also be careful not to oversimplify the model, so as not to underfit.

ii. Hyperparameters and Validation

Besides the parameters that are to be optimized by the training procedure, machine learning algorithms also have hyperparameters, i.e. parameters that are not directly set during the training phase, either because they are difficult to optimize, or because they should not be learnt from the training set. The latter case corresponds to the optimization of the parameters that directly affect the capacity of the model. In fact, if a parameter that affects the model capacity is tuned based only on the training set, the result will be that it will be chosen in order to minimize the training error as much as possible. However, we have seen how this would lead to a poor generalization error, due to overfitting.

To be more specific, anticipating some notions about ANNs to be discussed in the next section, an ANN is composed of several nodes whose input-output relationship is defined by some weights and bias terms, which are the parameters to be tuned during the training phase. On the other hand, the total number of nodes in the network and the way in which the nodes

are interconnected are hyperparameters that are considered fixed while the training algorithm is executed.

Besides the difficulty to optimize these discrete parameters, a critical problem is that the number of nodes in an ANN is directly related to the capacity of the network, since more nodes imply more degrees of freedom. Therefore, if we optimized the number of nodes based only on the training set, the optimum would be to use as many nodes as physically possible, thus causing overfitting.

On the other hand, it is also not possible to use the test set to tune the hyperparameters, because all choices pertaining to the algorithm design must be independent of the data set that is used to assess the performance of the algorithm. Otherwise, the estimation of the generalization error will be biased. This implies that we need a third data set for hyperparameter tuning, the validation set. The validation set is typically obtained by partitioning the training data into the training set and the validation set. The training procedure fixes some values of the hyperparameters and optimizes the network parameters based only on the training set. Afterwards, an estimate of the generalization error obtained with the considered hyperparameter configuration is obtained through the validation set. This procedure is repeated for different hyperparameter configurations to identify the best model to use. After both the parameters and hyperparameters have been set, the true generalization error is computed by using the test set. The main steps of the whole procedure are summarized in Algorithm 1.

while Error on validation set not satisfactory **do**

Choose a set of hyperparameters;

Given the chosen hyperparameters run the learning procedure for parameter optimization using the training set;

Evaluate the error on the validation set;

end while

While Algorithm 1 provides one with a systematic procedure for training a machine learning algorithm, it does not address how to update the hyperparameter configuration in

each loop. In general, there is no simple, algorithmic way to do this, and indeed hyperparameter tuning is more an art than a science

Machine Learning Methods

1. Methods based on the amount of human supervision in the learning process

a. Supervised learning

i. Classification

ii. Regression

b. Unsupervised learning

i. Clustering

ii. Dimensionality Reduction

iii. Anomaly Detection

c. Semi-supervised learning

d. Reinforcement learning

2. Methods based on the ability to learn from incremental data samples

a. Batch learning

b. Online learning

3. Methods based on their approach to generalization from data samples

a. Instance based learning

b. Model based learning

3.3 Deep Learning

The field of Deep Learning is a sub-field of Machine Learning that has recently come into much prominence. Its main objective is to get Machine Learning research closer to its true goal of “making machines intelligent”. Deep Learning is often termed as a rebranded fancy term for neural networks. This is true to some extent but there is definitely more to

Deep Learning than just basic neural networks. Deep Learning based algorithms involves the use of concepts from representation learning where various representations of the data are learned in different layers that also aid in automated feature extraction from the data. In simple terms, a Deep Learning based approach tries to build machine intelligence by representing data as a layered hierarchy of concepts, where each layer of concepts is built from other simpler layers. This layered architecture itself is one of the core components of any Deep Learning algorithm.

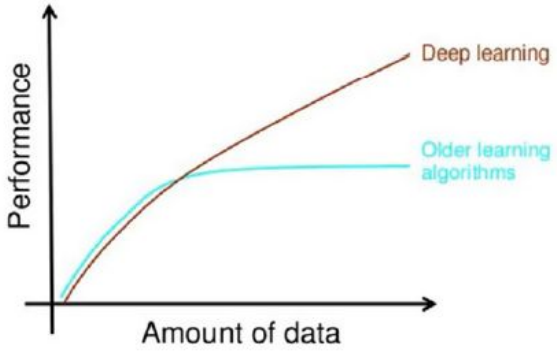


Fig 3.6 : Amount of data vs Performance

In any basic supervised Machine Learning technique, we basically try to learn a mapping between our data samples and our output and then try to predict output for newer data samples. Representational learning tries to understand the representations in the data itself besides learning mapping from inputs to outputs. This makes Deep Learning algorithms extremely powerful as compared to regular techniques, which require significant expertise in areas like feature extraction and engineering. Deep Learning is also extremely effective with regard to its performance as well as scalability with more and more data as compared to older Machine Learning algorithms. This is depicted in Figure 3.6 based on a slide from Andrew Ng's talk at the Extract Data Conference.

3.3.1 The Brain and The Neuron

In animals, learning occurs within the brain. If we can understand how the brain works, then there might be things in there for us to copy and use for our machine learning systems. While the brain is an impressively powerful and complicated system, the basic building blocks that it is made up of are fairly simple and easy to understand. We'll look at them shortly, but it's worth noting that in computational terms the brain does exactly what we

want. It deals with noisy and even inconsistent data, and produces answers that are usually correct from very high dimensional data (such as images) very quickly.

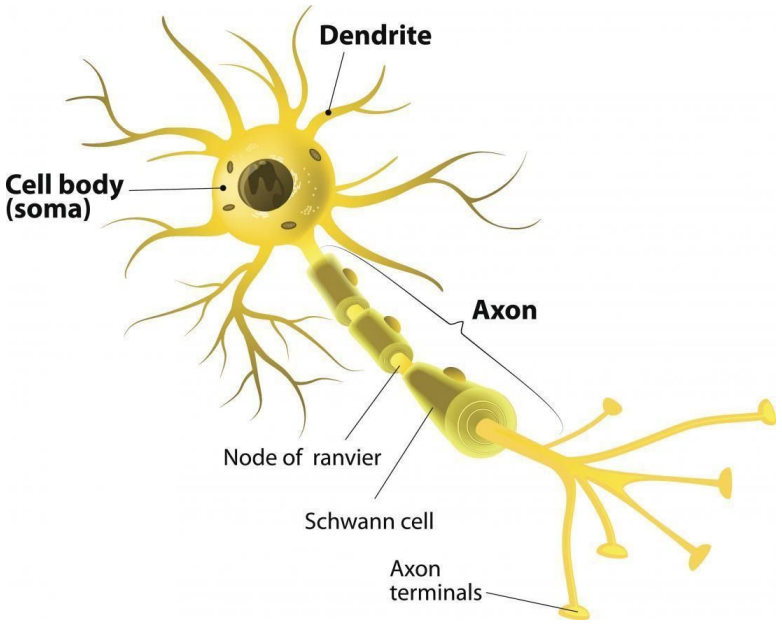


Fig 3.7: Neuron Structure

3.3.2 Perceptron

Rosenblatt's perceptron is built around a nonlinear neuron, namely, the McCulloch–Pitts model of a neuron. From the introductory chapter we recall that such a neural modelling consists of a linear combiner followed by a hard limiter (performing the signum function), as depicted in Figure 3.8. The summing node of the neural model computes a linear combination of the inputs applied to its synapses, as well as incorporates an externally applied bias. The resulting sum, that is, the induced local field, is applied to a hard limiter.

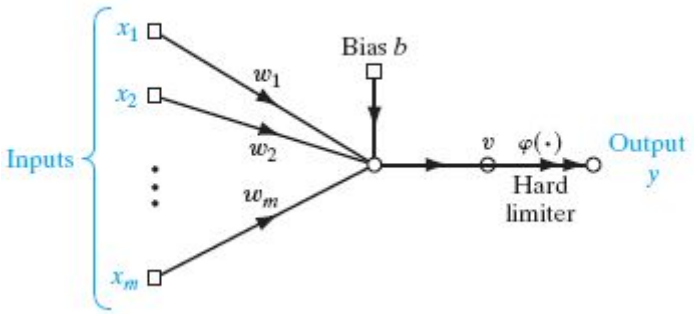


Fig 3.8 : Neuron Model

3.3.3 Artificial Neural Network (ANN)

An Artificial Neural Network (ANN) is a computational model and architecture that simulates biological neurons and the way they function in our brain. Typically, an ANN has layers of interconnected nodes. The nodes and their inter-connections are analogous to the network of neurons in our brain. A typical ANN has an input layer, an output layer, and at least one hidden layer between the input and output with inter-connections, as depicted in Figure 3.9.

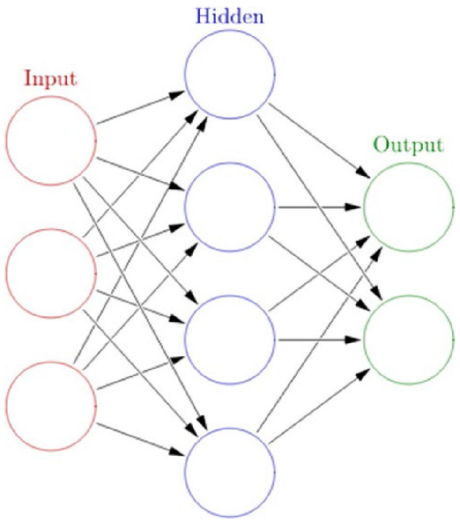


Fig 3.9 : Typical ANN

Any basic ANN will always have multiple layers of nodes, specific connection patterns and links between the layers, connection weights and activation functions for the nodes/neurons that convert weighted inputs to outputs. The process of learning for the network typically involves a cost function and the objective is to optimize the cost function (typically minimize the cost). The weights keep getting updated in the process of learning.

Artificial neural networks with multiple hidden layers between the input and output layers are called deep neural networks (DNNs), and they can model complex nonlinear relationships.

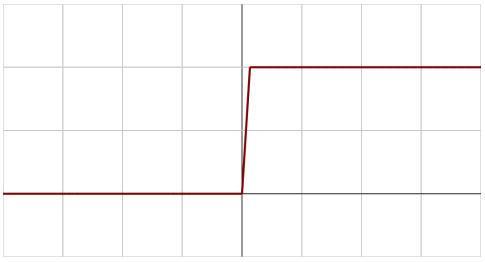
3.3.4 Activation Functions

The activation layer controls how the signal flows from one layer to the next, emulating how neurons are fired in our brain. Output signals which are strongly associated with past references would activate more neurons, enabling signals to be propagated more

efficiently for identification. Convolutional Neural Network (CNN) is compatible with a wide variety of complex activation functions to model signal propagation, the most common function being the Rectified Linear Unit (ReLU), which is favoured for its faster training speed. Some examples of the activation functions are listed below:

1. Identity:

Output, f (x) = *x*, the input itself. It will have the same effect of not connecting any activation functions. It's a linear function and is an Ideal condition. In practical cases, the required function will always be non-linear.

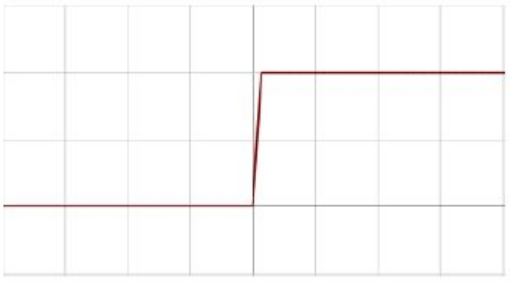


2. Binary Step:

The output will always be a binary, either 0 or 1.

Output, f (x) = { 0 *for x* > 0 1 *for x*≤0

This function is very useful for classification.

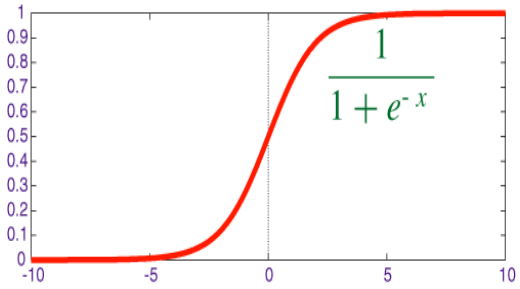


3. Logistic/ Sigmoid:

Using this activation function,

output f(x) = 1/1+e-x

It helps the output to be contained between 0 and 1, however large is the input may be.

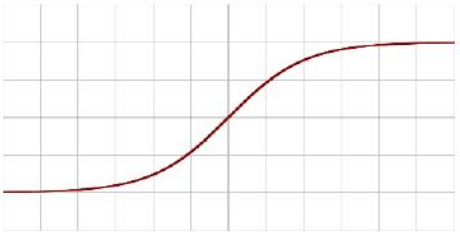


4. Tanh:

The output is given by

$f(x) = \tanh tanh\ (x) = \frac{2}{1+e^{2x}} - 1$

Tanh activation function is similar to Sigmoid and used interchangeably depending on the accuracy levels.

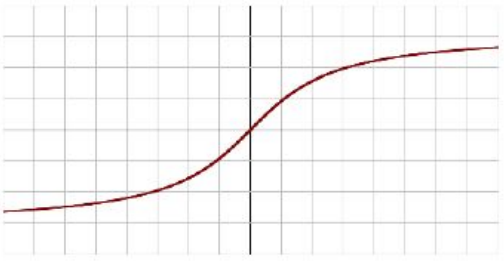


5. Arc Tan

Arc Tan activation function is also similar to sigmoid and tanh.

$Output, f(x) = tan^{-1}(x)$

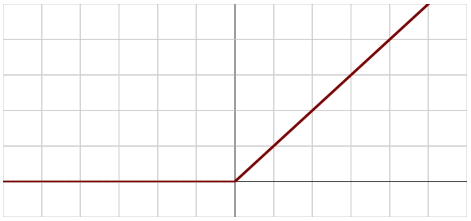
This function helps to contain the output between –pi/2 and + pi/2



6. ReLU (Rectified Linear Unit):

It helps to remove the negative portion of the function and the positive part remains the same.

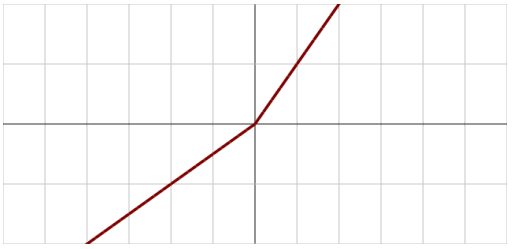
Output, f (x) = { 0 $for\ x > 0\ x\ for\ x \leq 0$



7. Leaky ReLU

Leaky ReLU reduces the magnitude of the negative portion.

Output, f (x) = { $0.01x\ for\ x > 0\ x\ for\ x \leq 0$



8. Softmax

Softmax activation function is used to impart probabilities. If we have multiple outputs, and when Softmax is used, we get the probability distribution for each of them.

$$softmax(x)i = \frac{exp(xi)}{\sum_{j} exp(xj)}$$

3.3.5 Back Propagation (Gradient Descent)

Back Propagation is a method used in artificial neural networks to calculate a gradient that is needed in the calculation of the weights to be used in the network.

The overall training process of the Convolution Network may be summarized as below:

Step1: We initialize all filters and parameters/weights with random values

Step2: The network takes a training image as input, goes through the forward propagation step (convolution, ReLU and pooling operations along with forward propagation in the Fully Connected layer) and finds the output probabilities for each class.

- Let's say the output probabilities for the boat image above are [0.2, 0.4, 0.1, 0.3]
- Since weights are randomly assigned to the first training example, output probabilities are also random.

Step3: Calculate the total error at the output layer (summation over all 4 classes)

Total Error = Σ ½ (target probability – output probability) ²

Step4: Use Back propagation to calculate the *gradients* of the error with respect to all weights in the network and use *gradient descent* to update all filter values/weights and parameter values to minimize the output error.

- The weights are adjusted in proportion to their contribution to the total error.

$$Wi = Wi + \Delta Wi \tag{3.2}$$

Where, $\Delta Wi = n \frac{dE}{dW_i} x_i$

n = learning rate

E = Error between predicted output and actual output

x_i = input

- When the same image is input again, output probabilities might now be [0.1, 0.1, 0.7, 0.1], which is closer to the target vector [0, 0, 1, 0].
- This means that the network has *learnt* to classify this particular image correctly by adjusting its weights/filters such that the output error is reduced.

- Parameters like number of filters, filter sizes, the architecture of the network etc. have all been fixed before Step 1 and do not change during training process – only the values of the filter matrix and connection weights get updated.

Step5: Repeat steps 2-4 with all images in the training set until the error no longer reduces (remain constant). If we don't stop, it can bring more errors. Fig 3.10 Shows a typical graph of error vs training.

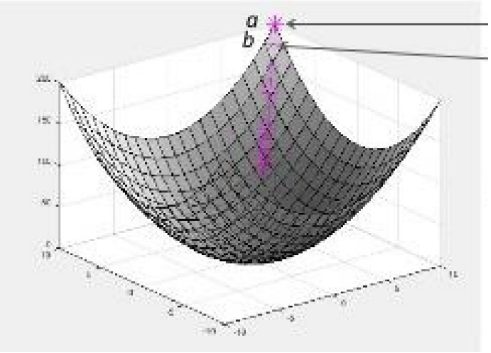


Fig 3.10 : Error vs Training Steps

3.3.6 Error Calculation

Machines learn by means of a loss function. There are many types of losses and optimizers designed for calculating the error value. Loss functions can be classified into two major categories depending upon the type of learning task we are dealing with — Regression losses and Classification losses. In classification, we are trying to predict output from set of finite categorical values i.e Given large data set of images of hand written digits, categorizing them into one of 0–9 digits. Regression, on the other hand, deals with predicting a continuous value for example given floor area, number of rooms, size of rooms, predict the price of room. Here Categorical Cross-Entropy, a type of Classification loss is used.

3.3.7 Cross-Entropy

In some cases, we have multiple outputs (Different Classes in Classification problem). In such cases, the prediction is by probability. The class with the highest probability will be the one which the model predicts the input belongs to. So the error is calculated as cross-entropy. Loss function *L* is given by,

$$L = -(y_i \log log\ (\hat{y_i}) + (1 - y_i) \log log\ (1 - \hat{y_i})) \qquad (3.3)$$

The calculated errors are taken partial derivatives with respect to each of the parameters (Weights and Biases). And the gradient is added to the weights (using the formula) for adjusting them. This type of optimizer is called gradient descent optimizer. There are other

types of optimizers also, which we are not discussing since they all are an extension or modification to the gradient descent algorithm itself.

The process of updating the weights requires calculation of error between the predicted output and the actual output. So this process begins in the last output layer of the network and passed onto the layers before them. Hence this process is called Back Propagation.

3.3.8 Optimizers

During the training process, we tweak and change the parameters (weights) of our model to try and minimize that loss function, and make our predictions as correct as possible. Optimizers tie together the loss function and model parameters by updating the model in response to the output of the loss function. In simpler terms, optimizers shape and mold your model into its most accurate possible form by futzing with the weights. The loss function is the guide to the terrain, telling the optimizer when it's moving in the right or wrong direction. Optimization problem is to minimize the cost function for finding the final weights for the network.

$$J(w)$$

where J(.) is the loss function and w is the weight vector.

Adam Optimizer

Adam is an optimization algorithm that can use instead of the classical stochastic gradient descent procedure to update network weights iterative based in training data. A learning rate is maintained for each network weight (parameter) and separately adapted as learning unfolds. The method computes individual adaptive learning rates for different parameters from estimates of first and second moments of the gradients.

When introducing the algorithm, the authors list the attractive benefits of using Adam on non-convex optimization problems, as follows:

- Straightforward to implement.
- Computationally efficient.
- Little memory requirements.

- Invariant to diagonal rescale of the gradients.
- Well suited for problems those are large in terms of data and/or parameters.
- Appropriate for non-stationary objectives.
- Appropriate for problems with very noisy/or sparse gradients.
- Hyper-parameters have intuitive interpretation and typically require little tuning.

Adam uses exponentially decaying average of past gradients, m_k (first moment) and past squared gradients, v_k (second moment) as given in equation (3.4) and (3.5) respectively. Adam weight update equation can be mathematically represented as equation (3.6)

$$m_k = \beta_1 m_{k-1} + (1 - \beta_1) \nabla J(w_k) \tag{3.4}$$

$$v_k = \beta_2 v_{k-1} + (1 - \beta_2) \quad \nabla^2 J(w_k) \tag{3.5}$$

$$w_{k+1} = w_k - \eta \cdot \frac{\sqrt{1-\beta_1^k}}{1-\beta_2^k} \cdot \frac{m_k}{\sqrt{v_k} + \varepsilon} \tag{3.6}$$

where η is the learning rate, w_k is the weight vector and $\beta 1$ $\beta 2 \in [0, 1)$ is a momentum parameter. (1- β_1^k) and (1- β_2^k) are provided to add the bias correction term for m_k and v_k.

Adam optimizer combines the benefits of two extensions of gradient descent called Adaptive Gradient Algorithm (AdaGrad) and Root Mean Square Propagation (RMSProp) optimizations. Stochastic gradient descent maintains a single learning rate (termed alpha) for all weight updates and the learning rate does not change during training. Unlike this method, Adam optimizer uses an adaptive learning rate method based on the average first moment (the mean) and the average of the second moments of the gradients as proposed in AdaGrad & RMSProp.

Stochastic Gradient Descent (SGD) Optimizer

Stochastic gradient descent (often shortened to SGD), also known as incremental gradient descent, is an iterative method for optimizing a differentiable objective function, a stochastic approximation of gradient descent optimization.

SGD is an optimization method which uses a fixed learning rate and the weight updation is done in every iteration as described in equation (3.7)

$$w_k = w_{k-1} - \eta \nabla J(w_{k-1}) \quad (3.7)$$

SGD is an optimization method which uses a fixed learning rate and the weight updation is done in every iteration .

Fig 3.11: Shows the comparison of different Optimizers and clearly shows the effectiveness of Adam optimizer.

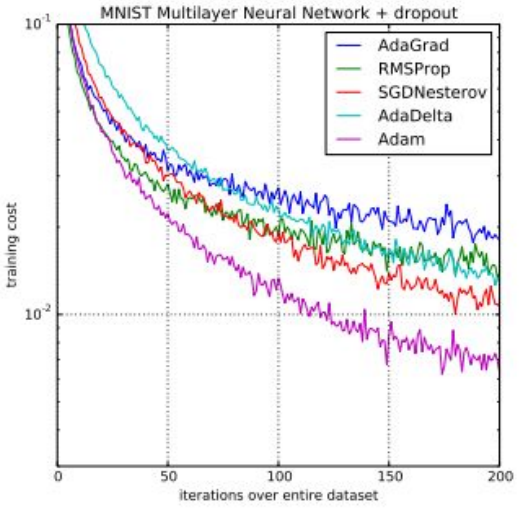


Fig 3.11 : Training Cost vs Iterations Graph

CHAPTER 4

DEEP CONVOLUTIONAL NEURAL NETWORK

4.1 Convolutional Neural Network

Neural Network performs as a Universal function approximator that learns the input-output relations by tuning the internal free parameters. CNN is modelled, inspired from the mammalian visual cortex and the hierarchical feature extraction from the image is incorporated into the neural network by exploiting the convolution operation.

The great success has been achieved in different areas such as Image Classification, Pattern Recognition, etc. by the application of CNN. CNN may contain several stacks of Convolutional layer, Pooling layer, Dropout layer and Fully Connected layer. The CNN layers are designed in such a way that all relevant features are extracted by bottom layers and abstract features are computed by high level layers. The spatial and local correlation of pixels is extracted and thus derived feature-map is mapped to the higher layer in CNN.

There are four main steps in CNN: convolution, subsampling, activation and full connectedness.

a) Convolution : A CNN usually takes an order 3 tensor as its input, e.g., an image with H rows,W columns, and 3 channels (R, G, B color channels). Higher order tensor inputs, however, can be handled by CNN in a similar fashion. The input then sequentially goes through a series of processing. One processing step is usually called a layer, which could be a convolution layer, a pooling layer, a normal-ization layer, a fully connected layer, a loss layer, etc. We will introduce the details of these layers later in this note.1For now, let us give an abstract description of the CNN structure first.

$$x^1 \longrightarrow \boxed{w^1} \longrightarrow x^2 \longrightarrow \cdots \longrightarrow x^{L-1} \longrightarrow \boxed{w^{L-1}} \longrightarrow x^L \longrightarrow \boxed{w^L} \longrightarrow z$$

(4.1)

The above Equation 4.1 illustrates how a CNN runs layer by layer in a forward pass. The input is $\boldsymbol{x}^1$, usually an image (order 3 tensor). It goes through the processing in the first layer, which is the first box. We denote the parameters involved in the first layer's processing

collectively as a tensor $\boldsymbol{w}^1$. The output of the first layer is $\boldsymbol{x}^2$, which also acts as the input to the second layer processing.

This processing proceeds till all layers in the CNN has been finished, which outputs $\boldsymbol{x}^L$. One additional layer, however, is added for backward error propagation, a method that learns good parameter values in the CNN. Let's suppose the problem at hand is an image classification problem with C classes. A com-monly used strategy is to output $\boldsymbol{x}^L$ as a C dimensional vector, whose i^{th} entry encodes the prediction (posterior probability of $\boldsymbol{x}^1$ comes from the i^{th} class). To make x^L a probability mass function, we can set the processing in the $(L-1)^{\text{th}}$ layer as a softmax transformation of $\boldsymbol{x}^{L-1}$ (cf. the distance metric and data transformation note). In other applications, the output $\boldsymbol{x}^L$ may have other forms and interpretations.

The last layer is a loss layer. Let us suppose t is the corresponding target(ground-truth) value for the input $\boldsymbol{x}^1$, then a cost or loss function can be used to measure the discrepancy between the CNN prediction $\boldsymbol{x}^L$ and the targett. For example, a simple loss function could be

$$z = \frac{1}{2}\|\boldsymbol{t} - \boldsymbol{x}^L\|^2$$

(4.2)

although more complex loss functions are usually used. This squared l^2 loss can be used in a regression problem. In a classification problem, the cross entropy loss is often used. The ground-truth in a classification problem is a categorical variable t. We first convert the categorical variable t to a C dimensional vector t (cf. the distance metric and data transformation note). Now both t and x^L are probability mass functions, and the cross entropy loss measures the distance between them. Hence, we can minimize the cross entropy (cf. the information theory note.) Equation 5 explicitly models the loss function as a loss layer,whose processing is modeled as a box with parameters w^L.

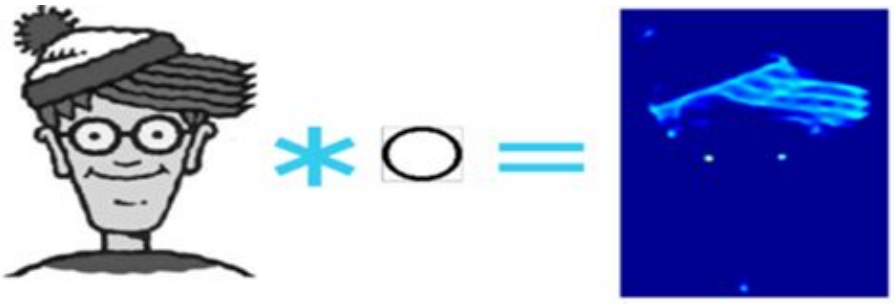


Fig 4.1: Convolving Wally with a circle filter. The circle filter responds strongly to the eyes.

The output of the convolution layer is called a feature map. The values of the feature-map derived from the convolutional layer (x,y) position can be calculated as equation (4.1)

$$f_{ij}^{xy} = \Phi \left\{ b_{ij} + \sum_{m} \sum_{p=0}^{P_i-1} \sum_{q=0}^{Q_i-1} w_{ijm}^{pq} f_{(i-1)m}^{(x+p)(y+q)} \right\} \tag{4.1}$$

where *fijxy* is the value of *jth* feature-map in the *ith* layer at (x,y) position, *Φ* is the activation function, *bij* is the bias corresponding to the feature-map, *m* is the index of the set of feature-maps in the (*i*−1)*th* layer, *Pi* x *Qi* is the dimension of the kernel, *wijmpq* is the kernel value at (p,q).

b) ReLU: ReLU is an activation function. Rectified Linear Unit (ReLU) transform function only activates a node if the input is above a certain quantity, while the input is below zero, the output is zero, but when the input rises above a certain threshold, it has a linear relationship with the dependent variable.

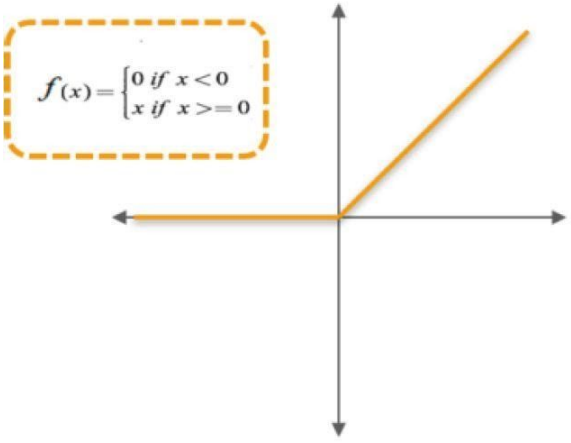


Fig 4.2: Rectified Linear Unit (ReLU) function

Inputs from the convolution layer can be "smoothened" to reduce the sensitivity of the filters to noise and variations. This smoothing process is called subsampling, and can be achieved by taking averages or taking the maximum over a sample of the signal. Examples of subsampling methods (for image signals) include reducing the size of the image, or reducing the color contrast across red, green, blue (RGB) channels.

c) Pooling: A pooling layer is another building block of a CNN. Its function is to progressively reduce the spatial size of the representation to reduce the amount of parameters and computation in the network. Pooling layer operates on each feature map independently. Pooling is done after passing through the activation layer. We do this by implementing the following 4 steps:

1. Pick a window size (usually 2 or 3)
2. Pick a stride (usually 2)
3. Walk your window across your filtered images
4. From each window, take the maximum value

The most common approach used in pooling is max pooling in which the maximum of a region is taken as its representative. For example in the following diagram a 2x2 region is replaced by the maximum value in it.

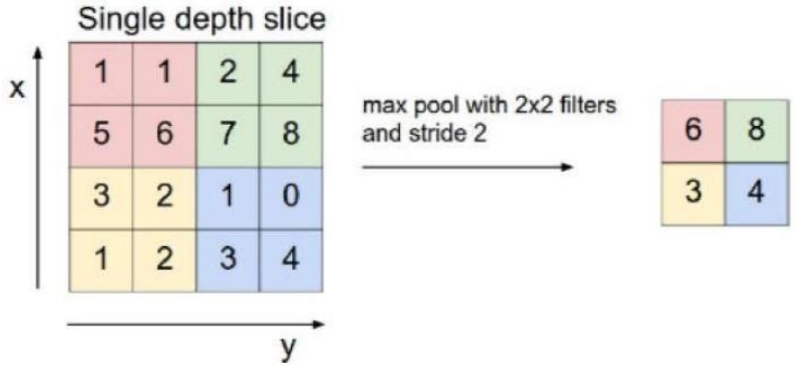


Fig 4.3: Max Pooling

d) Fully Connected: The last layers in the network are fully connected, meaning that neurons of preceding layers are connected to every neuron in subsequent layers. This mimics high level reasoning where all possible pathways from the input to output are considered.

CHAPTER 5
PROPOSED MODEL OF DEEP CONVOLUTIONAL NEURAL NETWORK DENOISER

5.1 ARCHITECTURE

The architecture of the proposed deep convolutional autoencoder model with the configurations as given in Table I is depicted in Fig 5.1.

Table I : Model Configurations

Layers	**Configuration**
Convolution(C1)	32 filters, 3x3 kernel, ReLU
Pooling(P1)	2 x 2 kernel
Convolution(C2)	64 filters, 3x3 kernel, ReLU
Pooling(P2)	2 x 2 kernel
Convolution(C3)	64 filters, 3x3 kernel, ReLU
Upsampling(U1)	2 x 2 kernel
Convolution(C4)	32 filters, 3x3 kernel, ReLU
Upsampling(U2)	2 x 2 kernels
Convolution(C5)	1 filter, 3 x 3 kernel, Sigmoid
Flatten(F)	784 unts
Dense(D1)	100 units
Dense(D2)	10 units

In this architecture, the input layer receives the resized image of dimension 28 x 28 pixels. A series of two convolution operations are then applied to the image to extract the features. The first convolution layer (C1) uses 32 kernels of size 3 x 3 to generate the feature maps and is activated with the Rectified Linear Unit (ReLU) function. Max-pooling (P1) layer of window size 2 x 2 is provided to subsample the obtained feature maps for a reduction in size to 14 x 14. The second convolution layer (C2) follows the procedure with 64 kernels of size 3 x 3 and ReLU activation. The Max-pooling (P2) sub-sampled with window size of 2 x 2 and yield 64 feature maps of size 7 x 7.The third convolution layer (C3) follows procedure with

64 kernels of size 3 x 3 and ReLU activation. The Upsampling (U1) is with the window size of 2 x 2 and yields 64 feature maps of size 14 x 14. The fourth convolution layer (C4) follows the procedure with 32 kernels of size 3 x 3 and ReLU activation. The Upsampling (U2) is with the window size of 2 x 2 and yields 32 feature maps of size 28 x 28.

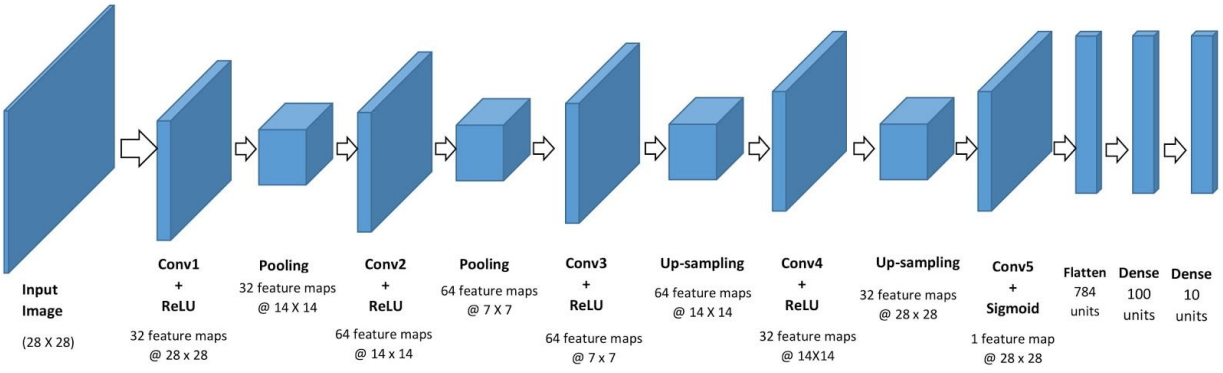


Figure 5.1 Architecture of DCNN Denoiser

The fifth convolution layer (C5) follows the procedure with 1 kernel of size 3 x 3 and Sigmoid activation. These feature maps are then flattened(F) resulting in a feature vector with a dimension of 1 x 784. The first Dense layer with activation function of ReLU works on this feature vector resulting in a 1 x 100 vector. The second dense layer (D2) using softmax activation, produces the probabilistic output corresponding to the ten classes of digits.

CHAPTER 6

METHODOLOGY

6.1 DATASET

The dataset used for training, validation and testing is the MNIST handwritten dataset with a total size of 70,000 images. Equal number of different classes of images were used for training to avoid biasing. Validation of the network performance was also measured along with the training phase to ensure proper learning. Out of the 70,000 samples, 60,000 samples were considered for training, 10,000 for testing as well as validation as shown in Table II. The images have been resized to 64 x 64 for reduction in computational complexity without loss of relevant information. A sample dataset is as displayed in Figure 6.1.

Table II :Dataset Split Details

Dataset	No. of Classes	No. of images	No. of training images	No. of validation images
MNIST Handwritten Image Dataset	10	70,000	60,000	10,000

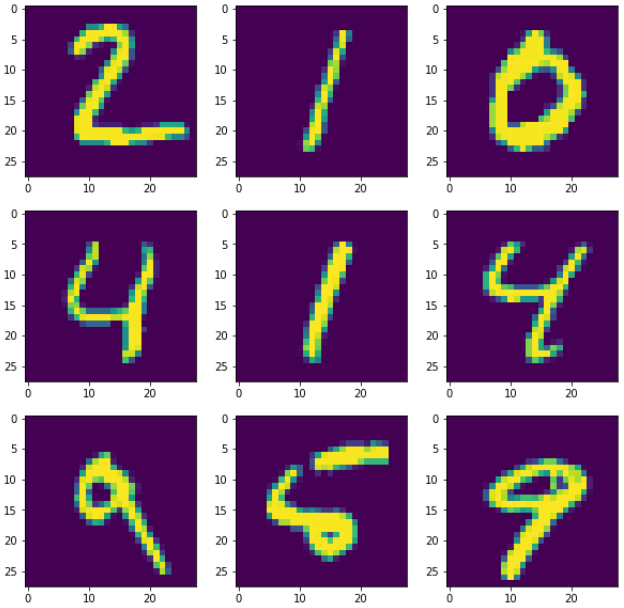


Figure 6.1 Samples from MNIST Dataset

6.2 NOISE IN IMAGES

Noise is always present in digital images during image acquisition, coding, transmission, and processing steps. It is very difficult to remove noise from the digital images without the prior knowledge of filtering techniques.Image noise is random variation of brightness or color information in the images captured. It is degradation in image signal caused by external sources.Images containing multiplicative noise have the characteristic that the brighter the area the noisier it. But mostly it is additive.

6.2.1 Gaussian Noise

Gaussian Noise is a statistical noise having a probability density function equal to normal distribution, also known as Gaussian Distribution. Random Gaussian function is added to Image function to generate this noise. It is also called as electronic noise because it arises in amplifiers or detectors. Source: thermal vibration of atoms and discrete nature of radiation of warm objects.The magnitude of Gaussian Noise depends on the Standard Deviation(sigma). Noise Magnitude is directly proportional to the sigma value.

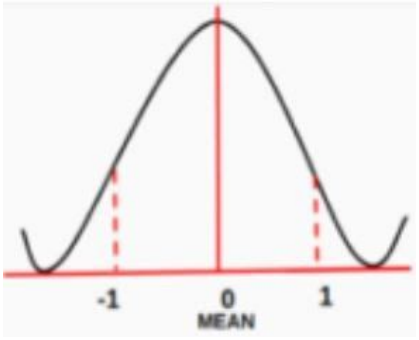


Fig 6.2 Plot of Probability Distribution Function

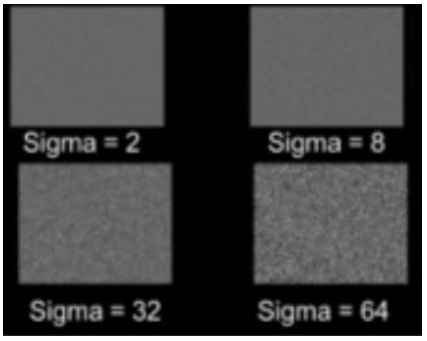


Fig 6.3 Effect of Sigma on Gaussian Noise

Gaussian noise generally disturbs the gray values in digital images. That is why Gaussian noise model essentially designed and characteristics by its PDF or normalizes histogram with respect to gray value. This is given as

$$P(g) = \sqrt{\frac{1}{2\pi\sigma^2}} e^{-\frac{(g-\mu)^2}{2\sigma^2}}$$

Where g = gray value, σ = standard deviation and μ = mean. Generally Gaussian noise mathematical model represents the correct approximation of real world scenarios.

6.2.2 Salt and Pepper Noise

It's a type of Impulse noise which is added to an image by addition of both random bright (with 255 pixel value) and random dark (with 0 pixel value) all over the image.This model is also known as data drop noise because statistically it drop the original data values. Let us consider 3x3 image matrices which are shown in the Fig. 3. Suppose the central value of matrices is corrupted by Pepper noise. Therefore, this central value i.e., 212 is given in Fig 6.4 is replaced by value zero. In this connection, we can say that, this noise is inserted dead pixels either dark or bright. So in a salt and pepper noise, progressively dark pixel values are present in bright regions and vice versa.

254	207	210
97	212	32
62	106	20

→

254	207	210
97	0	32
62	106	20

Fig 6.4 The central pixel value is corrupted by Pepper noise

Inserted dead pixel in the picture is due to errors in analog to digital conversion and errors in bit transmission. The percentage wise estimation of noisy pixels, directly determine from pixel metrics. The PDF of this noise is shown in the Fig 6.5

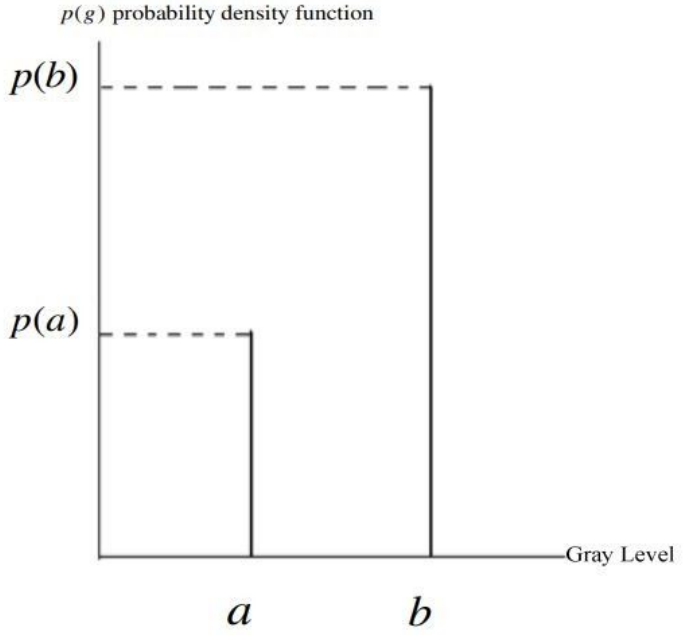


Fig 6.6 The probability distribution function of Salt and Pepper noise

6.2.3 Poisson Noise

The appearance of this noise is seen due to the statistical nature of electromagnetic waves such as x-rays, visible lights and gamma rays. The x-ray and gamma ray sources emitted a number of photons per unit time. These rays are injected into the patient's body from its source, in medical x rays and gamma rays imaging systems. These sources are having random fluctuations of photons. Result gathered image has spatial and temporal randomness. This noise is also called quantum (photon) noise or shot noise.

6.3 SOFTWARE REQUIREMENTS

a) IPython : Python is the programming language used for implementing this model. The Jupyter Notebook is the environment used for coding and analysis. Project Jupyter is a suite of software products used in interactive computing. IPython was originally developed by Fernando Perez in 2001 as an enhanced Python interpreter. A web based interface to IPython terminal in the form of IPython notebook was introduced in 2011. In 2014, Project Jupyter started as a spin-off project from IPython. IPython offers more features compared to the standard Python.

The libraries and packages used include keras, scikit-learn, matplotlib, numpy.

b) Keras : Keras is an open source deep learning framework for python. It has been developed by an artificial intelligence researcher at Google named Francois Chollet. Keras library is used to build the sequential network model and to add the convolutional layers in the model. Keras runs on top of open source machine libraries like TensorFlow, Theano or Cognitive Toolkit (CNTK). Keras is based on a minimal structure that provides a clean and easy way to create deep learning models based on TensorFlow or Theano. Keras is designed to quickly define deep learning models. Well, Keras is an optimal choice for deep learning applications. Features of Keras include:

- Consistent, simple and extensible API.
- Minimal structure -easy to achieve the result without any frills.
- It supports multiple platforms and backends.
- It is a user friendly framework which runs on both CPU and GPU.
- Highly scalability of computation.

c) Sckit-learn : Scikit-learn (Sklearn) is the most useful and robust library for machine learning in Python. It provides a selection of efficient tools for machine learning and statistical modeling including classification, regression, clustering and dimensionality reduction via a consistent interface in Python. This library, which is largely written in Python, is built upon NumPy, SciPy and Matplotlib. This library is used here for plotting the confusion matrix. The accuracy plot and loss plot are plotted using the matplotlib, which is a Python 2D plotting library.

d) NumPy : NumPy,which stands for Numerical Python, is a library consisting of multidimensional array objects and a collection of routines for processing those arrays. Using NumPy, mathematical and logical operations on arrays can be performed. Operations using NumPyUsing NumPy includes the following :

- Mathematical and logical operations on arrays.
- Fourier transforms and routines for shape manipulation.
- Operations related to linear algebra.
- NumPy has in-built functions for linear algebra and random number generation.

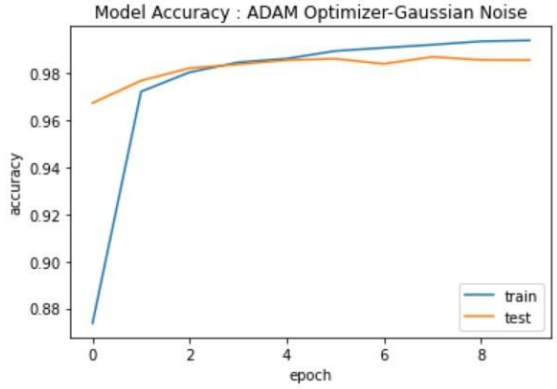
CHAPTER 7

RESULTS AND PERFORMANCE EVALUATION

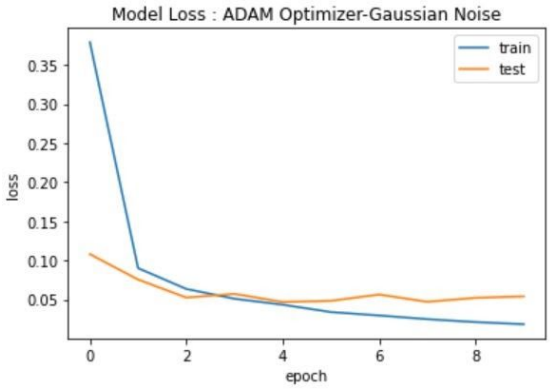
7.1 TRAINING AND TESTING RESULTS

The training and validation (test) was done for 10 epochs for both the models for all the 70000 images.

Figure 7.1(a) , 7.1(b) depicts the model accuracy and model loss for Adam model for Gaussian Noise

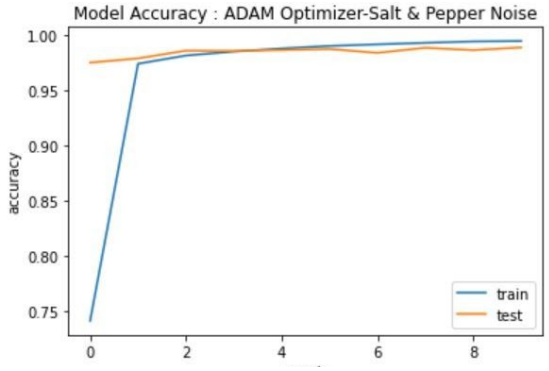


7.1 (a)

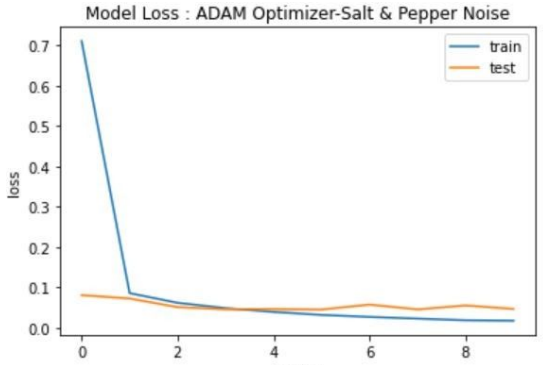


7.1 (b)

Figure 7.2(a) , 7.2(b) depicts the model accuracy and model loss for Adam model for Salt and Pepper Noise

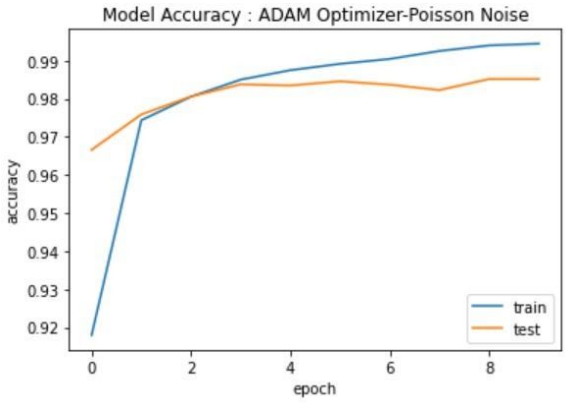


7.2 (a)

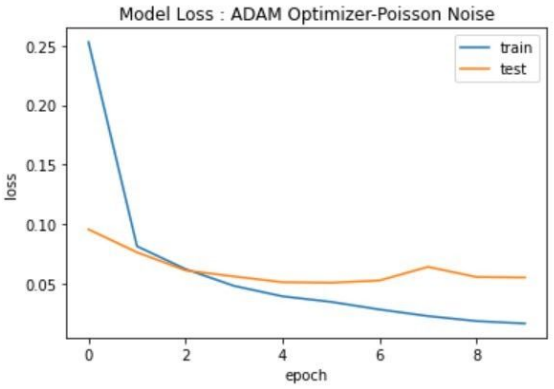


7.2 (b)

Figure 7.3(a) , 7.3(b) depicts the model accuracy and model loss for Adam model for Poisson Noise

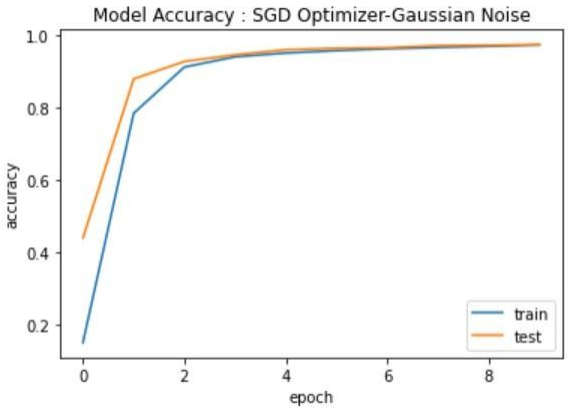


7.3 (a)

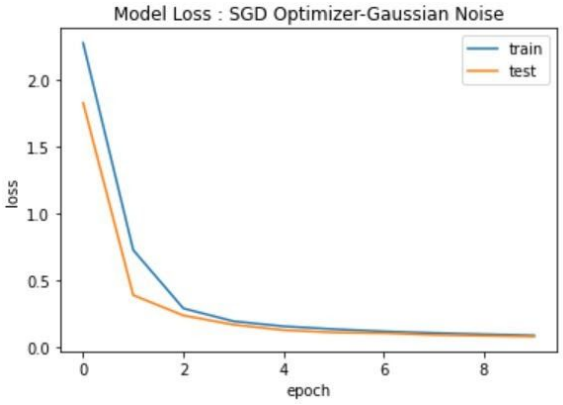


7.3 (b)

Figure 7.4(a) , 7.4(b) depicts the model accuracy and model loss for SGD model for Gaussian Noise

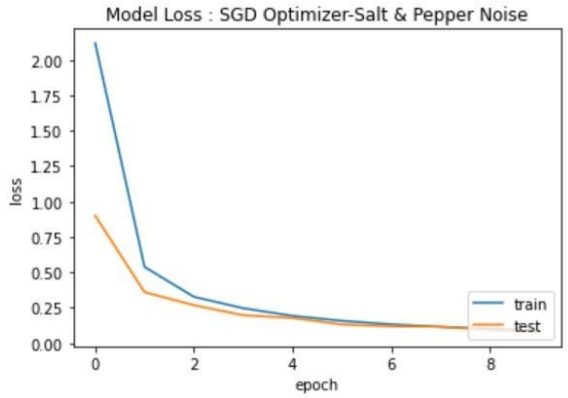


7.4 (a)

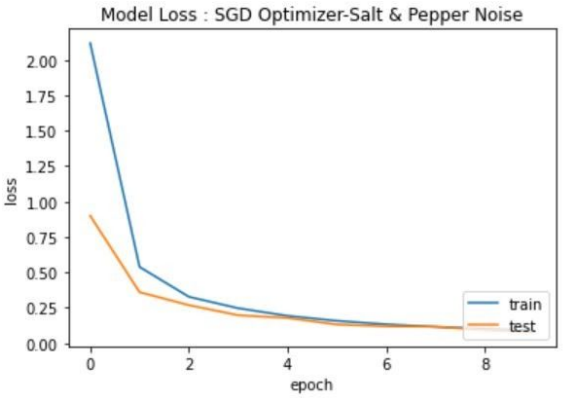


7.4 (b)

Figure 7.5(a) , 7.5(b) depicts the model accuracy and model loss for SGD model for Salt and Pepper Noise

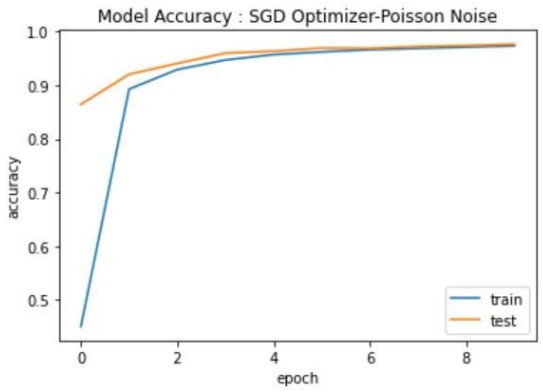


7.5 (a)

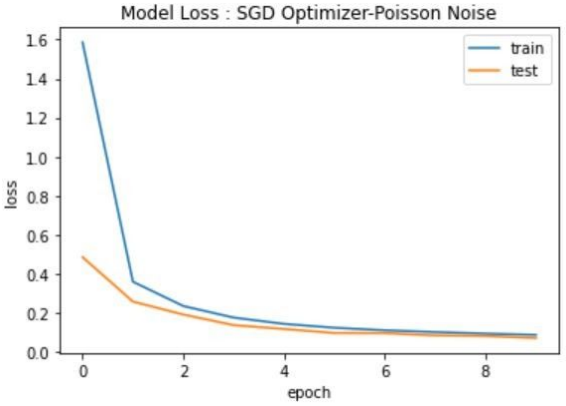


7.5 (b)

Figure 7.6(a) , 7.6(b) depicts the model accuracy and model loss for SGD model for Poisson Noise

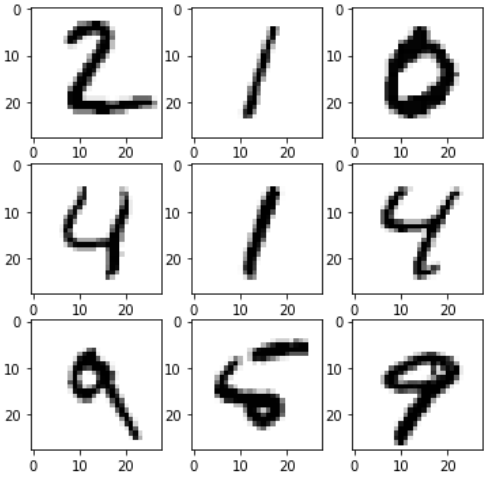


7.6 (a)

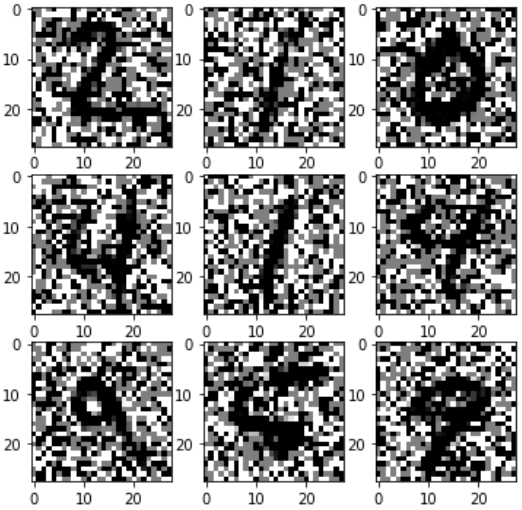


7.6 (b)

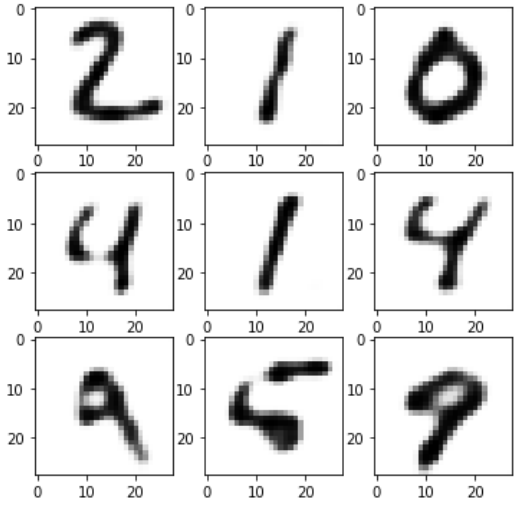
Figure 7.7 (a),7.7 (b) and 7.7 (c) shows the a sample of Actual Image, Noisy Image and the Predicted Image respectively



7.7 (a) Sample of actual image dataset without noise



7.7 (b) Sample of image dataset with noise added



7.7 (c) Sample of predicted image using the deep convolutional neural network model

7.2 CONFUSION MATRIX - PERFORMANCE EVALUATION

Once the training is completed, we need to know how well the trained model performs. The model was further tweaked for optimising these measures. Different measures were adopted to evaluate the performance of the CNN algorithms. Confusion matrix is one of them. It is a table that is often used to describe the performance of a classification model (or "classifier") on a set of test data for which the true values are known. It allows the visualization of the performance of an algorithm. These were Precision, Recall, F1-score, and Accuracy.

Accuracy: Accuracy is the overall performance of the model.

$$\text{Accuracy} = \frac{TP+FN}{TP+FP+TN+FN} \quad (1)$$

Precision: It is also called the Positive Predictive Value. Precision is the fraction of positive predictions divided by the total number of positive class values predicted, and it is calculated by (2)

$$\text{Precision} = \frac{TP}{TP+FP} \quad (2)$$

Where, TP and FP are True Positive and False Positive, respectively.

Recall: It is also known as Sensitivity. Recall is the fraction of positive predictions divided by the number of positive class values. Equation (3) is used to calculate Recall.

$$\text{Recall} = \frac{TP}{TP+FN} \quad (3)$$

Where, FN is False Negative.

Specificity: Specificity is the fraction of negative predictions divided by the number of negative class values.

$$\text{Specificity} = \frac{TN}{TN+FP} \quad (4)$$

F1 Score: F1 Score is also called the F-score or F-measure. F1 score conveys the balance between Precision and Recall. The value of F1-score becomes high only if the values of both Precision and Recall are high. F1-score values fall in the interval [0,1], and the highest the value, the better the classification accuracy. F1-score is calculated by

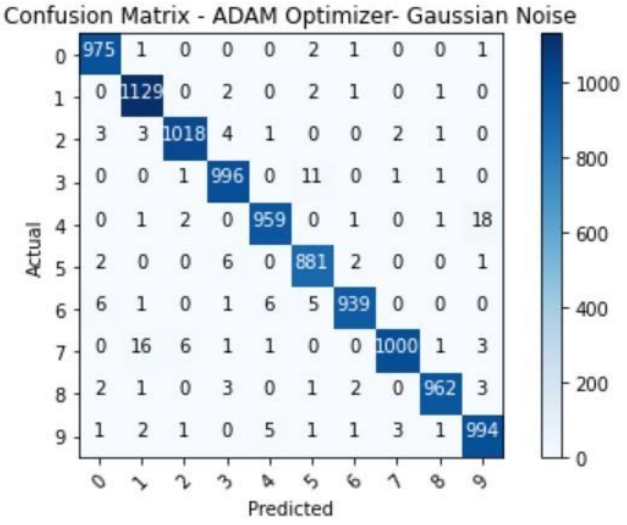
$$\text{F1 Score} = \frac{2TP}{2TP+FP+FN} \quad (5)$$

Table VII, shows the notation used by formulas to calculate Accuracy, Precision, Recall, Specificity andF1 Score, respectively.

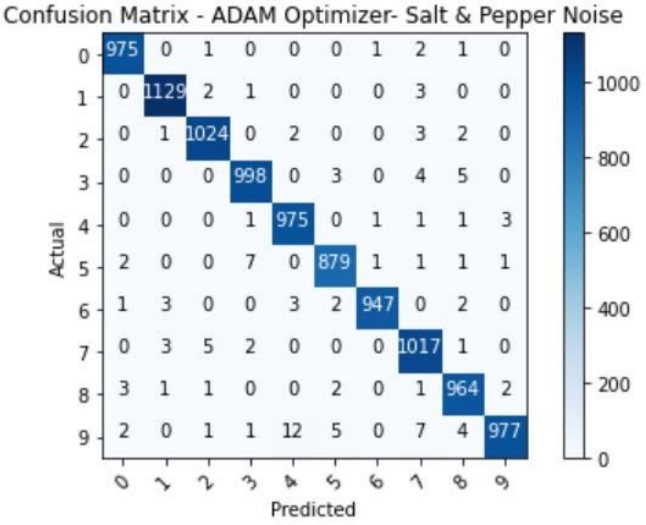
Table IV : TP FP FN TN

	Event = Positive	Event = Negative
Prediction = Positive	**True Positive**	**False Positive**
Prediction = Negative	**False Negative**	**True Negative**

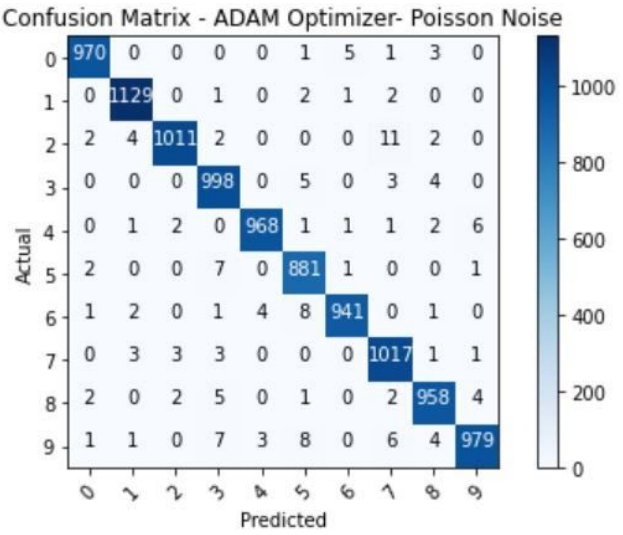
Figure 7.8 shows the Confusion matrix for all the 6 cases



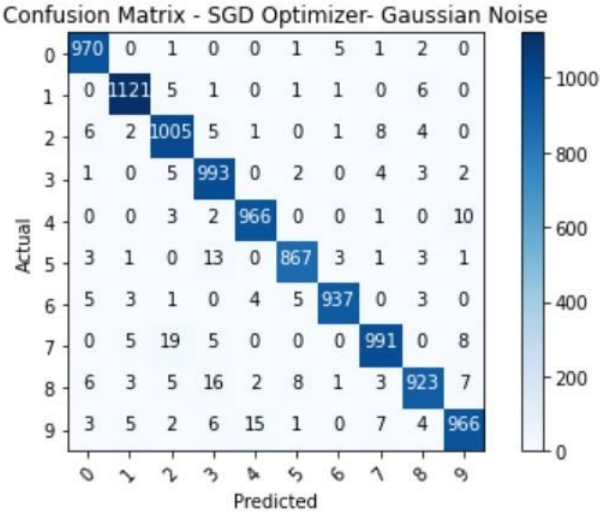
7.8 (a)



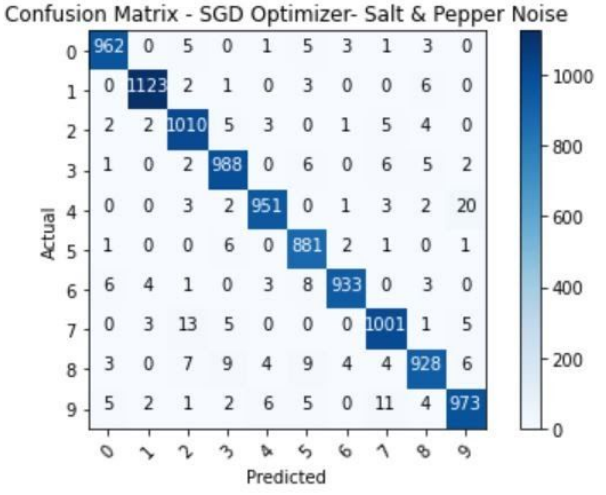
7.8 (b)


Confusion Matrix - ADAM Optimizer- Poisson Noise
Actual
Predicted
0 970 0 0 0 0 1 5 1 3 0
1 0 1129 0 1 0 2 1 2 0 0
2 2 4 1011 2 0 0 0 11 2 0
3 0 0 0 998 0 5 0 3 4 0
4 0 1 2 0 968 1 1 1 2 6
5 2 0 0 7 0 881 1 0 0 1
6 1 2 0 1 4 8 941 0 1 0
7 0 3 3 3 0 0 0 1017 1 1
8 2 0 2 5 0 1 0 2 958 4
9 1 1 0 7 3 8 0 6 4 979
0 1 2 3 4 5 6 7 8 9
1000
800
600
400
200
0

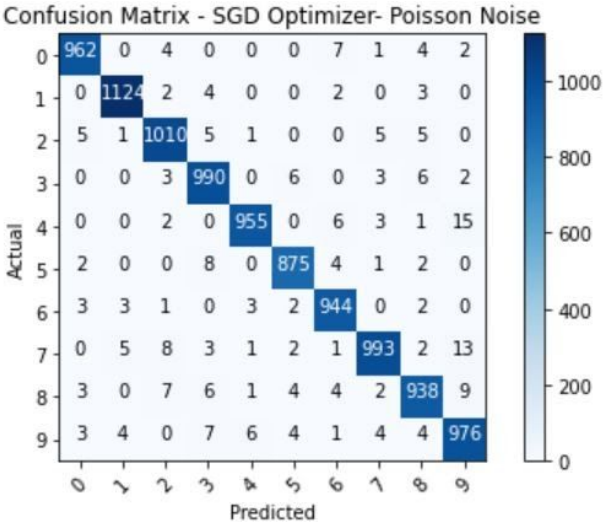
7.8(c)


Confusion Matrix - SGD Optimizer- Gaussian Noise
Actual
Predicted
0 970 0 1 0 0 1 5 1 2 0
1 0 1121 5 1 0 1 1 0 6 0
2 6 2 1005 5 1 0 1 8 4 0
3 1 0 5 993 0 2 0 4 3 2
4 0 0 3 2 966 0 0 1 0 10
5 3 1 0 13 0 867 3 1 3 1
6 5 3 1 0 4 5 937 0 3 0
7 0 5 19 5 0 0 0 991 0 8
8 6 3 5 16 2 8 1 3 923 7
9 3 5 2 6 15 1 0 7 4 966
0 1 2 3 4 5 6 7 8 9
1000
800
600
400
200
0

7.8 (d)


Confusion Matrix - SGD Optimizer- Salt & Pepper Noise
Actual
Predicted
0 962 0 5 0 1 5 3 1 3 0
1 0 1123 2 1 0 3 0 0 6 0
2 2 2 1010 5 3 0 1 5 4 0
3 1 0 2 988 0 6 0 6 5 2
4 0 0 3 2 951 0 1 3 2 20
5 1 0 0 6 0 881 2 1 0 1
6 6 4 1 0 3 8 933 0 3 0
7 0 3 13 5 0 0 0 1001 1 5
8 3 0 7 9 4 9 4 4 928 6
9 5 2 1 2 6 5 0 11 4 973
0 1 2 3 4 5 6 7 8 9
1000
800
600
400
200
0

7.8(e)



7.8 (f)

Accuracy for the models are shown the Table

Table IV : Accuracy of all the models

Noise	Adam Optimizer	SGD Optimizer
Gaussian Noise	0.9739	0.9853
Salt and Pepper Noise	0.9750	0.9884
Poisson Noise	0.9767	0.9851

CHAPTER 8

CONCLUSION

This project proposes a generalized Deep Convolutional Neural Network model to classify and recognize any type of noisy images. Two models were compared, the first model uses an Adam optimizer and the second one uses SGD optimizer. Three types of noise viz. Gaussian Noise, Salt and Pepper Noise and Poisson Noise were added to the input image and tested for the two models. From the training, test and validation results model 2, with SGD optimizer gives more accuracy when compared to the model Adam optimizer.

ADVANTAGES

- Far Less Parameters to train.
- Minimum computation costs.
- Small Model size
- Useful for Real-Time applications

DISADVANTAGES

- Less accurate with very high amount of noise added

CHALLENGES

The major challenges in training such a model are

1. Limited training Data : Gathering enough data to train a machine-learning model is the biggest challenge of all. It is very challenging to collect video dataset manually.
2. Misclassifications

APPLICATION

1. Image denoising application in Medical Images like computer tomography (CT) scan.
2. Image denoising of Satellite Images.

CHAPTER 9
FUTURE SCOPE

The deep convolutional neural network model achieved good performance and is suitable for other types of classification also. As this is a generalized model, this can be used to recognize and classify any type of image with any type of noise. Future work includes the improvement of the same model with high rate noise. Denoising video dataset using 3DCNN or LSTM(Long Short Term Memory Units) are also enhancements. Also this system can be implemented for denoising images in real time applications in medical field, space and research, etc.

REFERENCE

[1] O. Sheremet, K. Sheremet, O. Sadovoi and Y. Sokhina, "Convolutional Neural Networks for Image Denoising in Infocommunication Systems," 2018 International Scientific-Practical Conference Problems of Infocommunications. Science and Technology (PIC S&T), Kharkiv, Ukraine, 2018, pp. 429-432, doi: 10.1109/INFOCOMMST.2018.8632109.

[2] Q. Xiang and X. Pang, "Improved Denoising Auto-Encoders for Image Denoising," 2018 11th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI), Beijing, China, 2018, pp. 1-9, doi: 10.1109/CISP-BMEI.2018.8633143.

[3] S. Suresh, H. T. P. Mithun and M. H. Supriya, "Sign Language Recognition System Using Deep Neural Network," 2019 5th International Conference on Advanced Computing & Communication Systems (ICACCS), Coimbatore, India, 2019, pp. 614-618, doi: 10.1109/ICACCS.2019.8728411.

[4] Z. Liu, W. Q. Yan and M. L. Yang, "Image denoising based on a CNN model," 2018 4th International Conference on Control, Automation and Robotics (ICCAR), Auckland, 2018, pp. 389-393, doi: 10.1109/ICCAR.2018.8384706.

[5] Zarshenas, Amin & Suzuki, Kenji. (2018). Deep Neural Network Convolution for Natural Image Denoising. 2534-2539. 10.1109/SMC.2018.00434.

[6] Ç. P. Dautov and M. S. Özerdem, "Wavelet transform and signal denoising using Wavelet method," 2018 26th Signal Processing and Communications Applications Conference (SIU), Izmir, 2018, pp. 1-4, doi: 10.1109/SIU.2018.8404418.

[7] E. Pranav, S. Kamal, C. Satheesh Chandran and M. H. Supriya, "Facial Emotion Recognition Using Deep Convolutional Neural Network," 2020 6th International Conference on Advanced Computing and Communication Systems (ICACCS), Coimbatore, India, 2020, pp. 317-320, doi: 10.1109/ICACCS48705.2020.9074302.