

# **IMAGE CLASSIFICATION USING NEURAL NETWORKS**

Project report submitted in partial fulfilment of the requirement for the  
degree of Bachelor of Technology in  
**Computer Science and Engineering/Information Technology**

By

Pradyut Lohani (161313)

Under the supervision of

Dr. Rakesh Kanji



Department of Computer Science & Engineering and Information Technology

**Jaypee University of Information Technology, Waknaghat**

## Candidate's Declaration

I hereby declare that the work presented in this report entitled “ **Hand Written Digit classification using Neural Networks** ” in partial fulfilment of the requirements for the award of the degree of **Bachelor of Technology in Computer Science and Engineering** submitted in the department of Computer Science & Engineering and Information Technology, Jaypee University of Information Technology, Warknaghat is an authentic record of my own work carried out over a period from July 2019 to May 2020 under the supervision of Dr. Rakesh Kanji and Prof. Dr. Satya Prakash Ghreera.

The matter embodied in the report has not been submitted for the award of any other degree or diploma.



Pradyut Lohani (161313)

This is to certify that the above statement made by the candidate is true to the best of my knowledge.



Dr. Rakesh Kanji

(Assistant Professor – Senior Grade)

## **ACKNOWLEDGEMENT**

I take this opportunity to express our profound gratitude and deep regards to my project guide Dr. Rakesh Kanji for his exemplary guidance, monitoring and constant encouragement throughout the course of this project. The blessing, help and guidance given by him, time to time shall carry me a long way in the journey of life on which I am about to embark.

I am also obliged to staff members of JUIT College, for the valuable information provided by them in their respective fields. I am grateful for their cooperation during the period of my assignment.

Lastly, I thank almighty, our parents and our classmates for their constant encouragement without which this assignment would not have been possible.

# Contents

## **1. Introduction**

- 1.1 Introduction
- 1.2 Problem Statement
- 1.3 Aim
- 1.4 Methodology
- 1.5 Organization

## **2. Literature Survey**

- 2.1 Books and publication

## **3. System Development**

- 3.1 System Architecture
- 3.2 Perceptron
- 3.3 Prediction and loss function
- 3.4 Gradient descent and logistic regression algorithm
- 3.5 Non-Linear models
- 3.6 Neural Networks
- 3.7 Training a neural network
- 3.8 Measures to increase efficiency
- 3.9 Neural Networks using pytorch
- 3.10 Algorithms

## **4. Performance Analysis**

- 4.1 Command Line Testing
- 4.2 Dataset Testing
- 4.3 Performance

## **5. Conclusion**

## **References**

## List of Figures & Tables

1. Layers of Deep Learning [4]
2. Deep learning Vs Machine Learning [5]
3. Advantages of Deep Learning [7]
4. Image Recognition Roadmap [8]
5. Input Image [9]
6. Top Classes of Classification [9]
7. Output Image [10]
8. Organization of project [10]
9. Layers of Application [12]
10. Perceptron [13]
11. Discrete and continuous prediction [14]
12. Maximum Likelihood probability [15]
13. Gradient descent w.r.t weight and bias [17]
14. Weight manipulation w.r.t gradient [17]
15. Non-linear models [18]
16. Probability in non-linear models [19]
17. Weight to linear models [19]
18. Structure of neural networks [20]
19. Feed-forwarding process [21]
20. Algorithm for back-propagation [22]
21. Over-fitting [23]
22. Early-stopping [23]
23. Classifier model using class [24]
24. Classifier model without class [25]
25. Feed forwarding and back propagation in pytorch [26]
26. Dropout in pytorch model [26]
27. Loading Image datasets [27]
28. Transfer Learning [28]
29. Libraries imported [31]
30. Dataset details [32]
31. Input [33]
32. Model [34]
33. Model Training [34]
34. Epochs Run [35]
35. Accuracy [35]
36. Output [36]
  
37. Table 1: Effective model in terms of probability and loss [15]

## **ABSTRACT**

The world is currently engaged with Internet for all its work and will continue to do so in future. Now the mundane works of the users are seeking the help of Artificial Intelligence for its effectiveness and for its learning. Going ahead, AI algorithms will be fused into an ever-increasing number of ordinary applications. For instance, you may need to incorporate a picture classifier in a Smartphone application. To do this, you'd utilize a deep learning model prepared on countless pictures as a feature of the general application architecture. A huge piece of programming improvement later on will utilize these sorts of models as common parts of uses.

In this project, we'll train an image classifier to classify distinctive types of handwritten digits.

With this undertaking, I have designed a program that can be trained on any arrangement of marked digits.

# **CHAPTER 1: INTRODUCTION**

## **1.1 Introduction**

When we talk about classifying images, we usually observe and spot distinguishing features in items available to us, scattered in our surroundings. This is based on various different reasons and the primary reason behind this is the fact that our brain/mind is trained and is always training unconsciously with the same arrangement of images that brings about the ability in humans to differentiate among various different things easily. When we explore the real world, we usually do not keep taking notes, we don't really realize it and our mind keeps taking in the information automatically. Experiencing and discovering different aspects of the world is not a test for us, our subconscious mind does this without any problems.

Unlike the human mind, the computer sees visuals with different features and finds patterns in the image, whether it is still or video, to understand and distinguish the key elements of the image. The way a computer/any other system perceives a picture is not the same as that of other living beings. Computer Vision uses photographic computations to analyse the graphics from an image/picture or an assortment of many different pictures. One of the most commonly occurring instances of computer vision is to separate the population which is on foot and the other which is on vehicles, setting up multiple user-given/uploaded images with high precision.

Keeping in mind the emerging capabilities of ML and computer vision, many companies are investing more and more in image recognition to analyse the data for various uses from many different kinds of graphic sources, for example; Graphic testing, classification of objects, face recognition and many more.

Image recognition is the ability of the system to detect objects in images. It uses computer vision advancement with artificial intelligence and computed calculations to detect images through the camera system. With the ongoing development of machine learning and the

computational power of machines, image recognition has shocked the world. Cars, Internet Commerce, Retail, Security, Probation, Social Insurance, Agriculture, Image Recognition are some of the most widely used areas.

### **Applications-**

- Traffic Management System: Digit recognizer can be used to identify the number plates of violators. Sometimes at some busy juncture a rule violator may be missed by the authorities patrolling there, but image recognizer systems can be used to identify the offenders.
- Banks: Several cheques are processed every day at the bank. A digit recognizer can be used to automate the procedure. It can be used to identify the account numbers and the transaction can be carried out rapidly without any manual intervention.
- Drones: Drones armed with image recognition can be used to provide surveillance, image-based automatic monitoring, and to control different situations in distant locations.
- Industrial: Examining formation lines, evaluating basic objects/settings at every moment of time inside the premises. Detecting the success/failure of the previous items which went through, to bring reduction in the inadequacies of the object. Assessing each and everyone of their employees can help organizations to have total control of different processes in the framework.
- Auto Driving Vehicles: The future vehicles being made with object recognition systems can differentiate objects and take relevant decisions regarding the next step for the vehicle. Smaller androids can be used to help organizations to find and exchange articles starting from one location then moving onto the next one using these autonomous vehicles to save time and manpower.
- Military Surveillance: Identification of suspicious actions in different regions can help counter an assault and save innocent lives. Image recognition systems can do this automatically and then inform the authorities of any illegal activities, so they can plan the next course of action.



## **CONVOLUTIONAL NEURAL NETWORKS (CNN)**

Convolutional neural networks are deep neural networks used to organize images, group them by similarity, and provide object recognition. These algorithms can detect objects, people, and various other types of information.

CNNscan do optical character recognition (OCR) on simple, written images that digitize content and enable natural-language processing by hand-reporting and interpreting images. The same is true with the sound when speaking externally as a spectrogram. Recently, CNNs have been particularly concerned with content testing and graph messaging with graph CNNs.

The software industry is moving towards machine intelligence in the contemporary era. Machine learning is essential in every field as machines work beyond their capabilities. In short, machine learning is made up of algorithms that interpret data, learn from them, and apply them to make intelligent decisions to tackle real life problems with a simple solution.

Examples of machine learning are everywhere. HBO GO knows what you want to see or how snapchat recognizes a digital photo. Or how the Customer Benefit Agent will know if you can be happy with their help before attempting a CSAT study. All these examples show us the importance and the growing need of convolutional neural networks.

The thing about traditional machine learning algorithms is that however great they are and however great their performance is, they remain a machine and give machine like results to the user. They need space, skill and human intervention. For AI inventors, researchers and the rest of the world, deep learning holds more guarantee to advance the world to a more promising technological land.

Deep Learning(DL) is a subset(and one of the most important one right now) of Machine Learning that achievesexemplaryresults and flexibility by gaining knowledge to interact with the world as established order of ideas, with every single idea characterized in connection to fewer complex ideas, and more exclusive and exceptional portrayals.

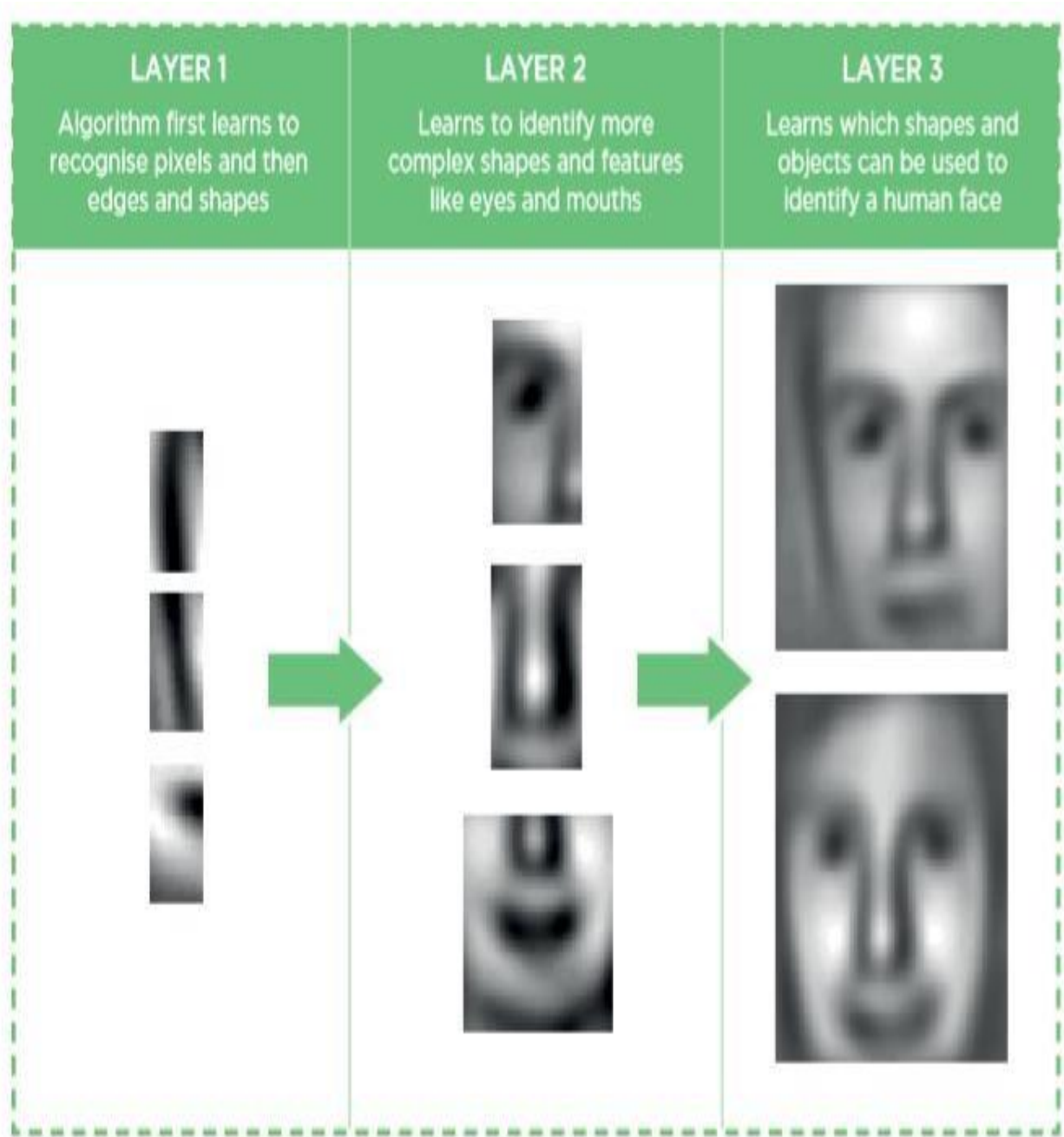


Fig. 1

## 1.2 Problem Statement

As previously specified, Image recognition has many different uses, from military to image captcha for digital security, and all the above-mentioned uses will continue to grow as organizing images is becoming a basic requirement for an organization. If the dataset is small, classification can be done manually by hand, but for classifying large datasets of images manual doing the classification becomes very time consuming and wastes resources and becomes next to impossible. Also, for manually classifying the images, complete knowledge of class sets and its nuances is required otherwise estimates can be unfitting. These difficulties in manual classification led to the rise of idea of image classification applications. Numerous machine learning models are used for image classification and all of them have their own difficulties. In traditional Machine learning models, most of the applied parameters need to be recognized in order to reduce the complexity of the data and make patterns more noticeable. The main advantage of Deep Learning algorithms is that it attempts to study high level features from data in an incremental way.

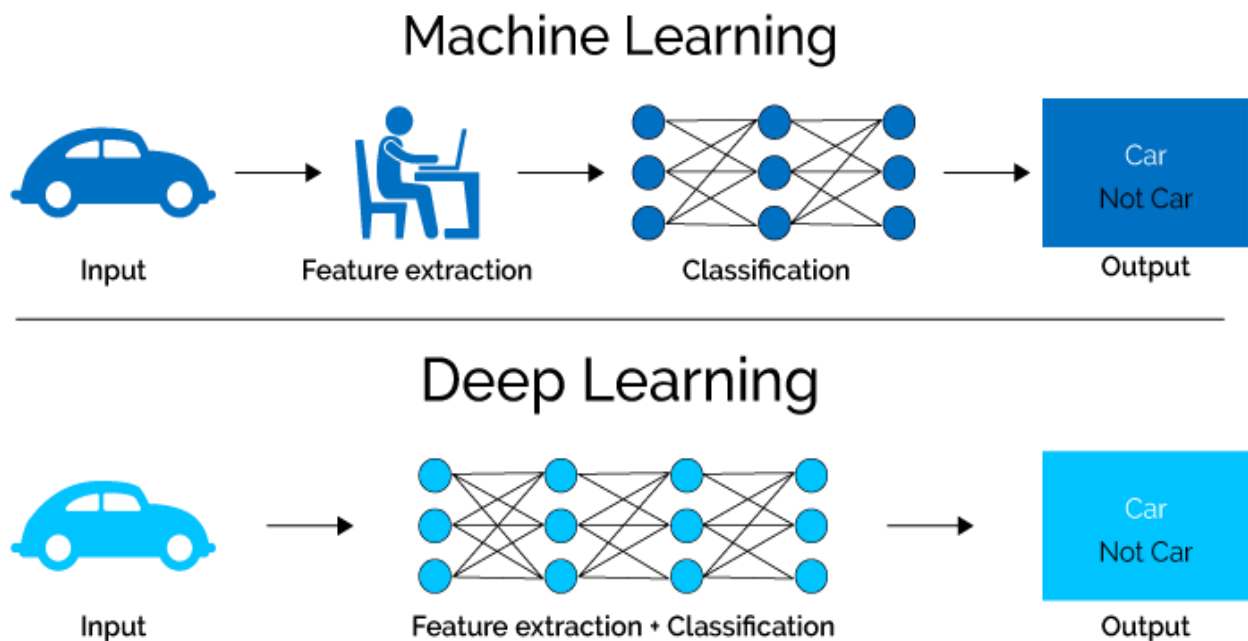


Fig. 2

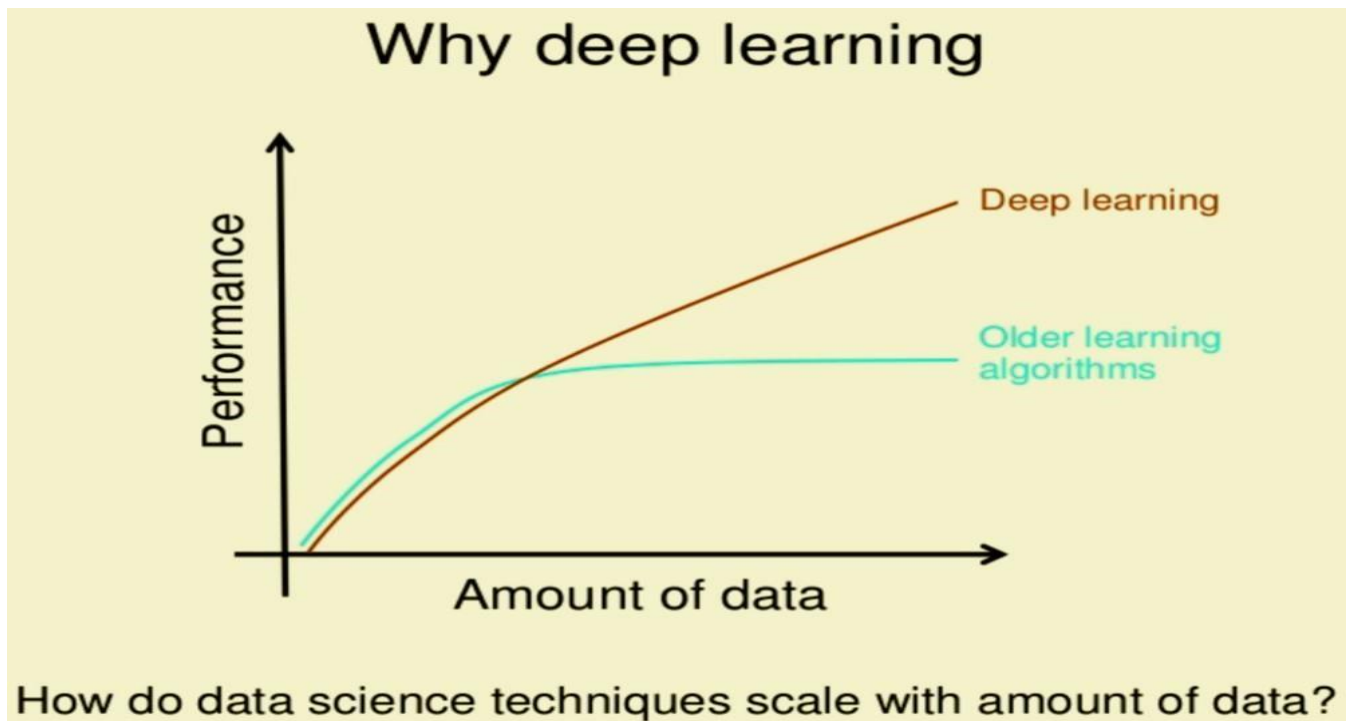


Fig. 3

### 1.3 Aim

The aim of this project is to implement image classification to classify hand written numbers. The image classifying model is built and trained on the data from a dataset which will classify images of each number into its respective label.

### 1.4 Methodology

The first step is to import all the necessary packages/libraries with appropriate epithets.

We break down the data into three parts and load each part for a different use:

- **Training:** To design and train, various transformations like image scaling, image cropping, etc. will be done, and this will give us a higher accuracy in our model. Input data, given in the

dataset, will be resized to a specific size as mandated by the pre-trained networks (generally 224x224).

- **Testing and Validation:** The usage of these datasets is in computing the accuracy of the designed model on new and never seen before data. Transformation is to be done before using the data.

### **Building the classifier after training:**

We use the pre-trained model from MNIST dataset to get all the different features found in the image.

- The pre-trained network is loaded.
- A new, untrained model is built by building a network as a classifier to classify images.
- Apply forward propagation to train the weights and biases for the neurons.
- Use backpropagation to fine tune the neural net layers to minimize the loss function using techniques like gradient descent to find the optimal value of the weight and bias term.
- Track correctness and the loss and cost to land on the best hyperparameter values for the model.

### **Class Prediction:**

The class of the image is determined by finding the probability of each class in the model.

The highest probability is the class to which the object belongs giving us the result in the form of probabilities.

# Image Recognition Roadmap

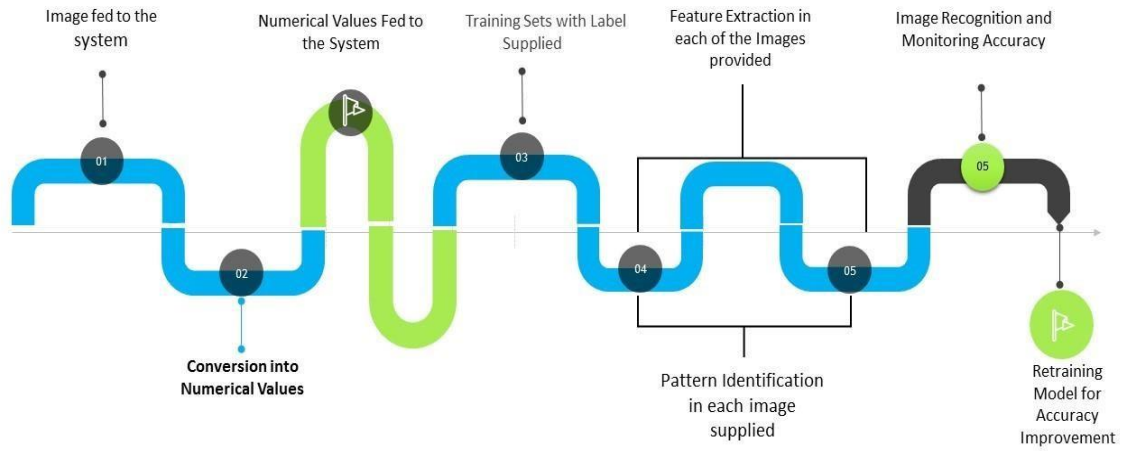


Fig. 4

## Input Example:

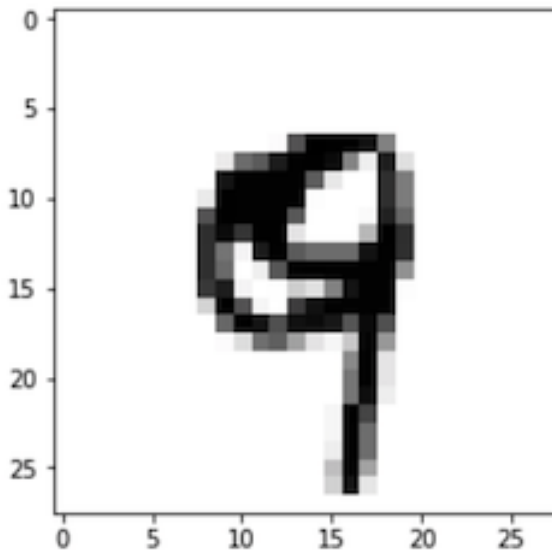


Fig. 5

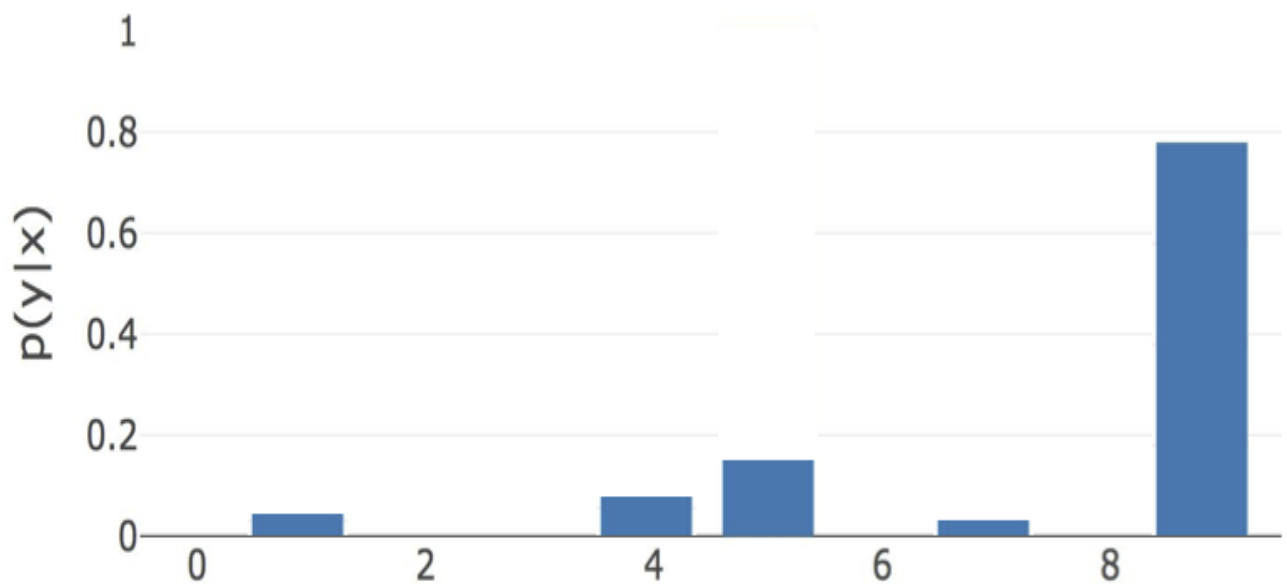


Fig. 6

Output Example:

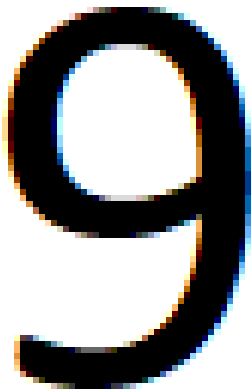


Fig. 7

## 1.5SET-UP

The project/model follows the following steps to obtain the best result:

1. Image dataset is loaded, and then pre-process the data to clean the image and make the size suitable for processing.
2. Train the classifier on the dataset.
3. Predict images using the classifier trained in the previous step.
4. Analyse the results provided by the model.

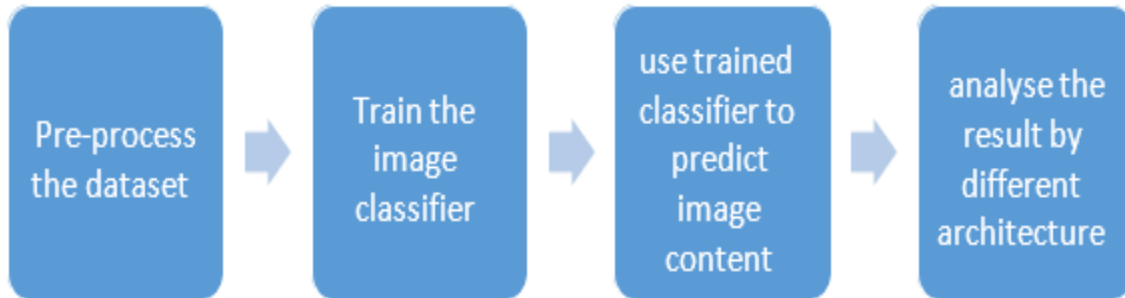


Fig. 8

## **CHAPTER 2: LITERATURE SURVEY**

### **2.1 Books and Publications**

I arrived at the decision to use a specific model for different classification technique used in the project required a thorough study of all the probable models and techniques with their advantages and disadvantages.

To classify the images the steps involved are:

- Defining the classes for result  
Various different properties of the images help in determining the classes of the images.
- Selection of Features  
To correct the differences between the classes and arrive at the correct result, feature selection is required. It is the most important aspect of ML as features of each class varies.
- Sampling the training data



It is necessary to sample the training data. Learning approaches like Supervised learning or Unsupervised learning will be selected according to the data.

- Training and testing the model

Applying the most suitable model to achieve the correct result.

The books that helped me in my research on the steps involved to achieve the result is:

- Deep Learning (Josh Patterson & Adam Gibson)
- Machine Learning for Image Classification by Yu-Jin Zhang

An IEEE paper titled ‘Simple convolutional neural network on image classification’ provided us insight and knowledge on topic of image recognition using neural networks.

### **CHAPTER 3: SYSTEM DEVELOPMENT**

### 3.1 System Architecture

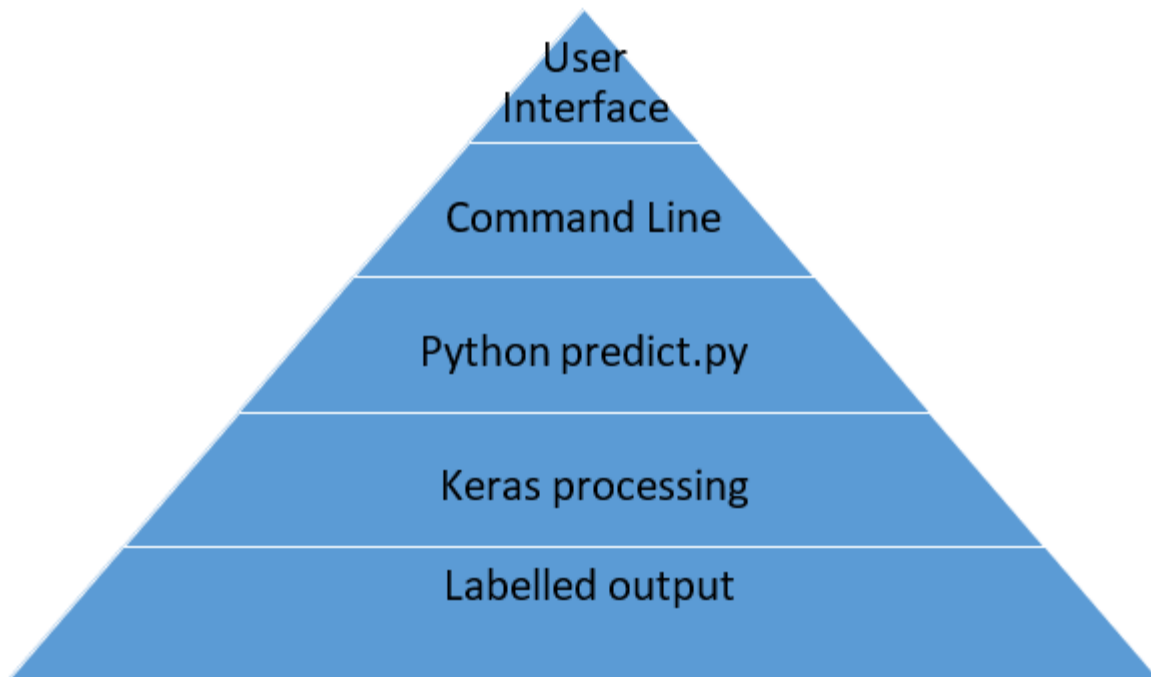


Fig. 9

### 3.2 Perceptron-

Perceptron is the building block of a neural network, similar to a neuron in the body. Perceptron takes the inputs and these inputs are multiplied by the weight value and this result is then processed by the perceptron and the binary output is given as a simple YES or NO. Perceptron is the simplest neural network, with a single hidden layer.

The inputs are fed into the hidden layer, where it is computed and then outputted through the output layer.

Here, the perceptron is given the job to predict if the student will be eligible for admission in a college on the basis of his/her examination grades and scores.

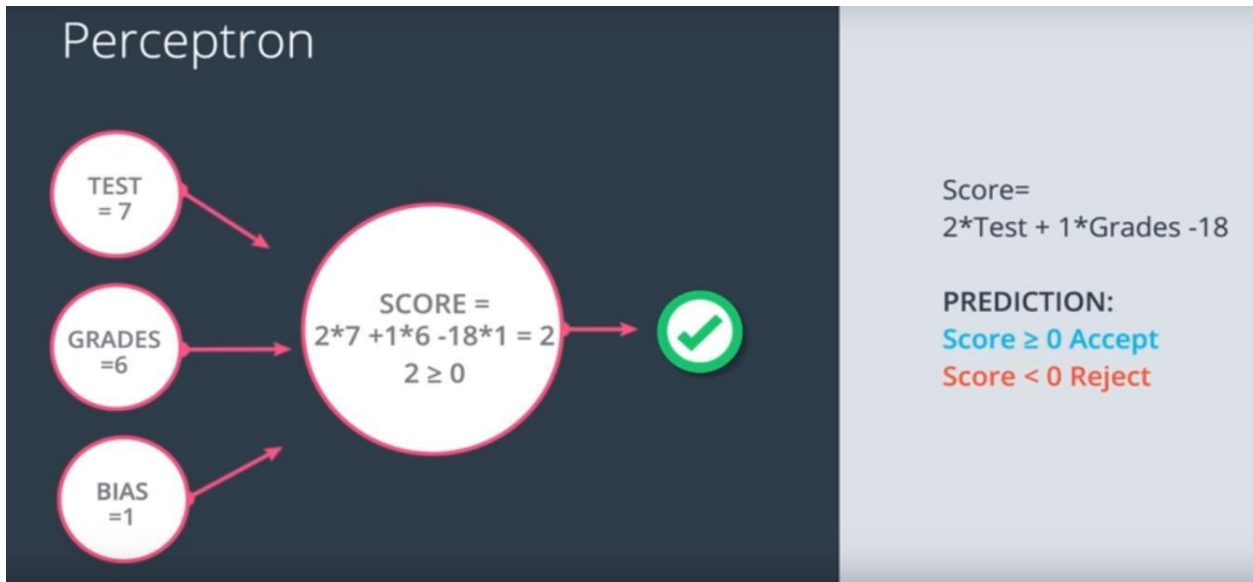


Fig. 10

Here, the inputs are the applicant's scores and grades, and the perceptron is fed these inputs, and taking into account the previous data, the linear equation is drawn by the perceptron, which gives the applicant's score. Where  $score \geq 0$  means acceptance and  $score < 0$  means rejection.

The linear equation determines the perceptron result.

This linear equation is likely to predict the wrong output. To avoid this, the equation must be changed with the use of weights.

Types of False Assessments: -

- If the result is negative, but the object label is positive.
- If the result is positive, but the object label is negative.

And these two types of errors have very distinct solutions.

### Perceptron Algorithm:

For every misclassified point ( $x_1, x_2, \dots, x_n$ ):

1. if prediction=0:
  - for  $i=1$  to  $n$ :
    - change  $w(i)$  to  $w(i)+ax(i)$
    - change  $b$  to  $b+a$
2. if prediction=1:
  - for  $i=1$  to  $n$ :
    - change  $w(i)$  to  $w(i)-ax(i)$
    - change  $b$  to  $b-a$

### 3.3 Loss Function and prediction of the model:

We get only two outputs from the perceptron as it uses a step function, but in the physical world in reality the number of outputs required may be far more than two. To overcome this error, we use the sigmoid function, which returns a value ranging from 0 to 1.

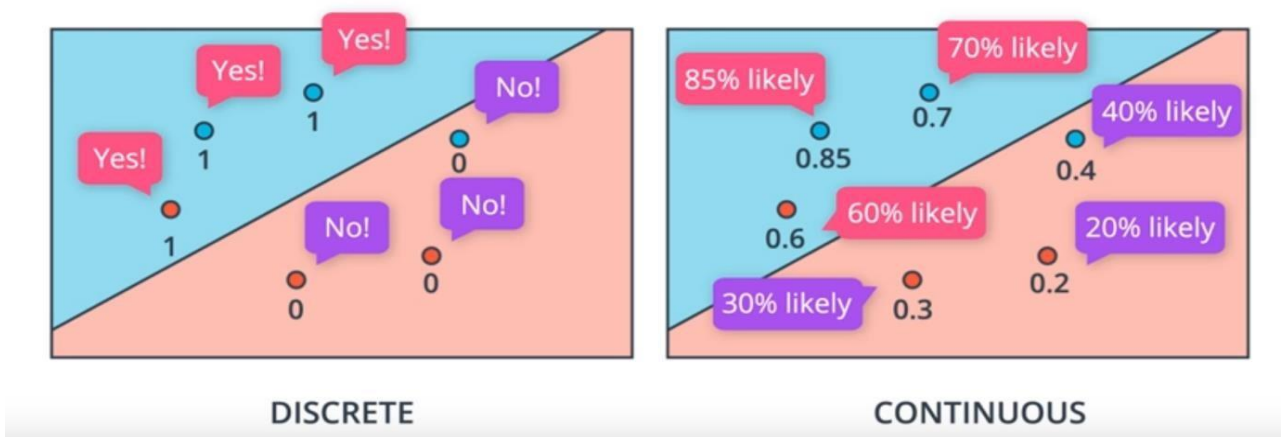


Fig. 11

The points on the line has value 0.5 and the value of the points vary as they move away from the line. If the point is below the line it decreases and if the point is above the line it increases.

Here, the points above the line have the probability 0.7, 0.6 and 0.85 in the above figure. So, we will use Softmax function if we need to classify the points into three or more classes.

Softmax function for class A:  $\exp(a)/(\exp(a)+\exp(b)+\exp(c))$

Softmax function for class B:  $\exp(b)/(\exp(a)+\exp(b)+\exp(c))$

Softmax function for class C:  $\exp(c)/(\exp(a)+\exp(b)+\exp(c))$

*Maximum Likelihood Probability*: It is the multiplied result of all the calculated probabilities of each colour as predicted by the above-mentioned perceptron. For eg;

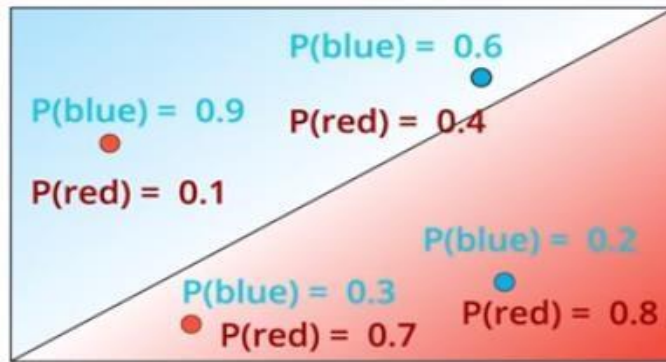


Fig. 12

$$MLP = 0.1 * 0.6 * 0.7 * 0.2 = 0.0084$$

If the number of digits is so large that the product of the probability is too small and of little or no importance, we add the individual probabilities, which is the concept of cross entropy.

In *CROSS ENTROPY*, we take a log of all the probabilities and add them.

In the above-mentioned example, cross entropy is given by computing the log values of the probabilities:

$$\ln(0.6) + \ln(0.2) + \ln(0.1) + \ln(0.7) = 4.8$$

To sum up:

	Product of probabilities	Cross-Entropy loss
More effective model	more	less

Table 1

### 3.4 Gradient descent and logistic regression algorithm:

Logistic regression follows the following steps to achieve the result:

- i. Importing the dataset to use it as our input data.
- ii. Design and Implement the model on the data.
- iii. Calculate the loss function.
- iv. Minimizing the loss/error by minimizing the cost function.

We can define an error function with the help of cross entropy loss

$$\text{Error Function} = - \frac{1}{m} \sum (1-y) * (\ln(1-\text{ycap})) + y * \ln(\text{ycap})$$

where, ycap: predicted output

y: label of the point.

**To decrease the loss/error**, gradient descent is the easiest and most widely used procedure. The main purpose and use of Gradient Descent(GD) is to decrease the output value of a function by iteratively moving in the direction of steepest descent as defined by the negative direction of the gradient. In ML, to find the most suitable value of the parameters of our model we use gradient descent. Suppose that, we stand on a hill and proceed towards

the ground to get down from the hill. Many different options are there to choose from, but we select the onewhere we have to move the lease and we move to a position lower in height than the previous position. Now we repeat this step until we reach the foot of the hill, the same thing happens during minimization of loss, we compute the gradient of loss w.r.t weights and bias and update the respective weights and bias to get a better result.

Now, we can go ahead and calculate the derivative of the error  $E$  at a point  $x$ , with respect to the weight  $w_j$ .

$$\begin{aligned}
 \frac{\partial}{\partial w_j} E &= \frac{\partial}{\partial w_j} [-y \log(\hat{y}) - (1 - y) \log(1 - \hat{y})] \\
 &= -y \frac{\partial}{\partial w_j} \log(\hat{y}) - (1 - y) \frac{\partial}{\partial w_j} \log(1 - \hat{y}) \\
 &= -y \cdot \frac{1}{\hat{y}} \cdot \frac{\partial}{\partial w_j} \hat{y} - (1 - y) \cdot \frac{1}{1 - \hat{y}} \cdot \frac{\partial}{\partial w_j} (1 - \hat{y}) \\
 &= -y \cdot \frac{1}{\hat{y}} \cdot \hat{y}(1 - \hat{y})x_j - (1 - y) \cdot \frac{1}{1 - \hat{y}} \cdot (-1)\hat{y}(1 - \hat{y})x_j \\
 &= -y(1 - \hat{y}) \cdot x_j + (1 - y)\hat{y} \cdot x_j \\
 &= -(y - \hat{y})x_j
 \end{aligned}$$

A similar calculation will show us that

$$\frac{\partial}{\partial b} E = -(y - \hat{y})$$

Fig. 13

## Gradient Descent Step

Therefore, since the gradient descent step simply consists in subtracting a multiple of the gradient of the error function at every point, then this updates the weights in the following way:

$$w'_i \leftarrow w_i - \alpha[-(y - \hat{y})x_i],$$

which is equivalent to

$$w'_i \leftarrow w_i + \alpha(y - \hat{y})x_i.$$

Similarly, it updates the bias in the following way:

$$b' \leftarrow b + \alpha(y - \hat{y}),$$

Fig. 14

## 3.5 Models with a Non-Linear design

So far, we have just spoken about and applied our algorithms to form a linear model, but NN can be used to perform even difficult tasks, and these complex and difficult tasks generally need a non-linear model. Ironically, we still require a linear model to obtain the non-linear model.

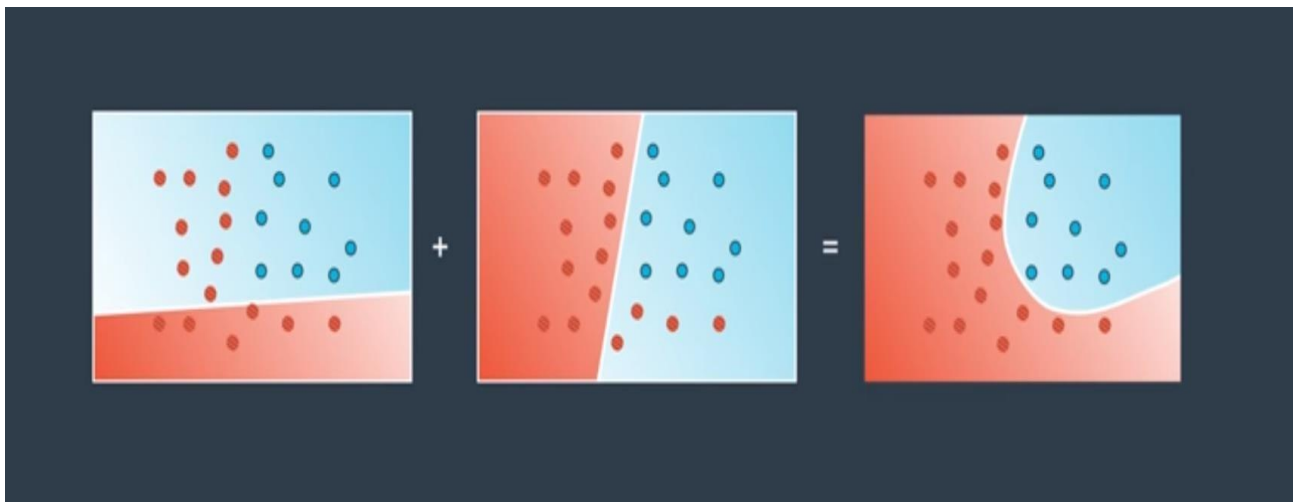


Fig. 15



In the figure given above, the first model which is linear, categorizes the red and blue points but the categorization is not very accurate, then in the second model which is also linear, the categorization of the blue and red points is still not very efficient but still gives a better result than the previous model, but on joining both the previous models we get the best result which is obtained by using a non-linear model.

Each point's probability is added from both the models and this sum is passed onto a sigmoid function which gives the best probability of that point using a non-linear model which was made by the combination of the two previous models.

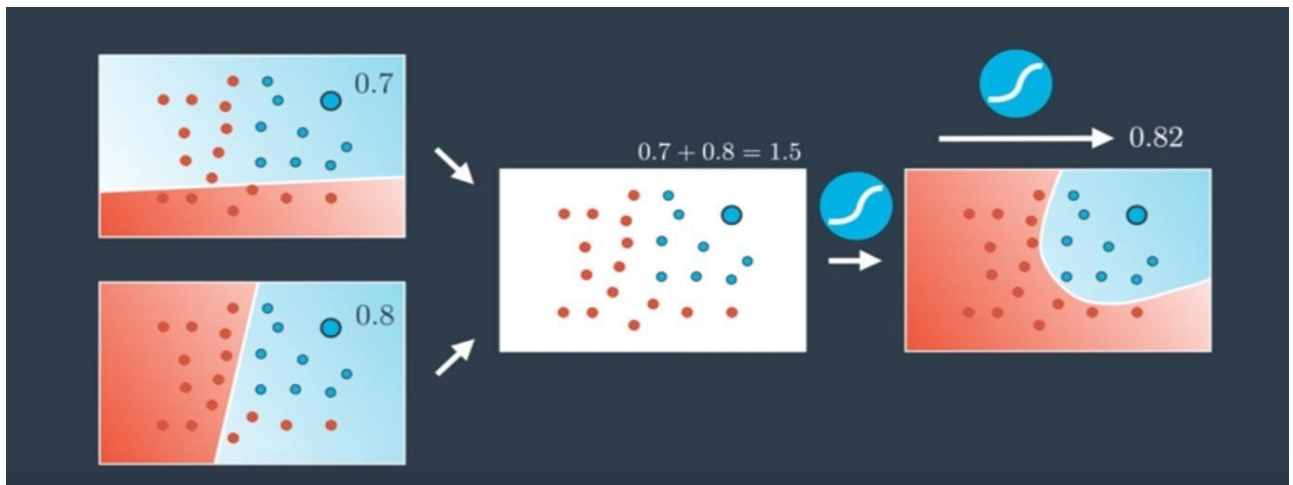


Fig. 16

For example; The probability of a point in the given image is 0.7 in the first model and 0.8 in the second model, and we add the two probabilities to 1.5, and then the sigmoid function is applied, which gives us the probability of 0.82 for the same point in the non-linear model.

We add weight and bias to the model to improve our result. In the image above we add a weight of 7 to Model-1 and 5 to Model-2, and then for this the output is calculated.

$$7 * (0.7) + 5 * (0.8) - 6 = 2.9 \rightarrow \text{sigmoid} \rightarrow 0.95$$

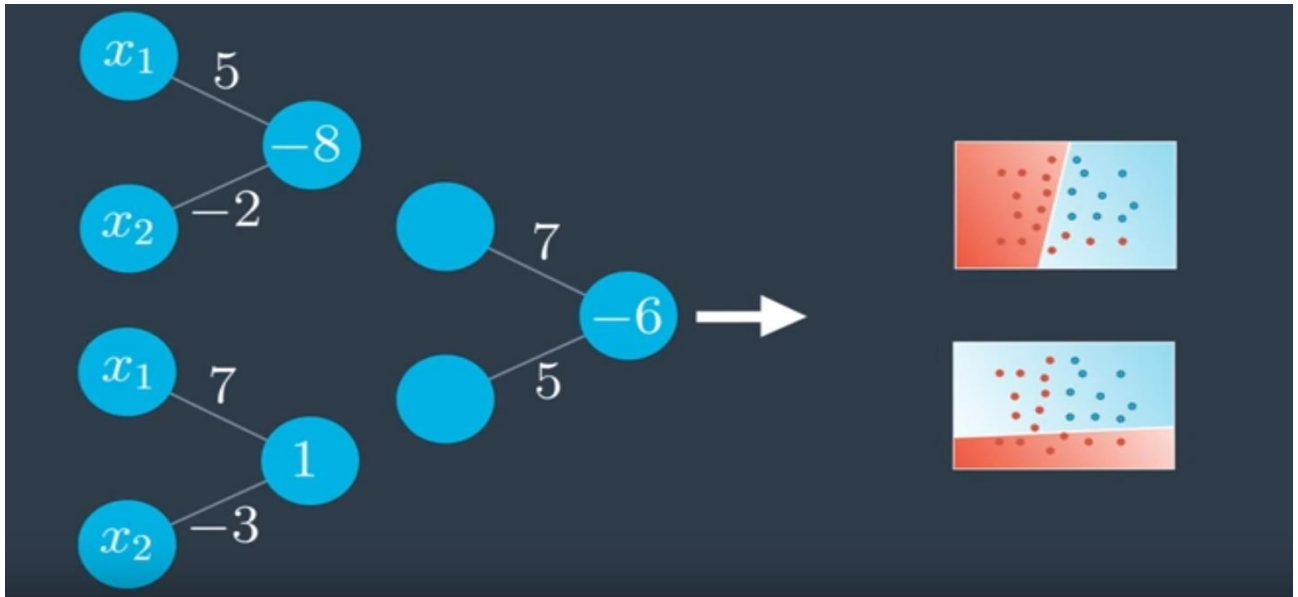


Fig. 17

All these concepts form the most basic conceptual intricacies of Deep Neural Networks.

### 3.6 Neural Networks(NN):

We can design a neural network using the above-mentioned concepts of perceptron and non-linear models (which we obtained by using 2 linear models). The following three layers form the building block of a neural network (NN):

- Input Layer
- Hidden Layer
- Output Layer

The figure below shows the working of these layers in a neural network as the input layer gives an input which is passed onto the hidden layer where the product and sum of weights nodes and biases respectively give an output.

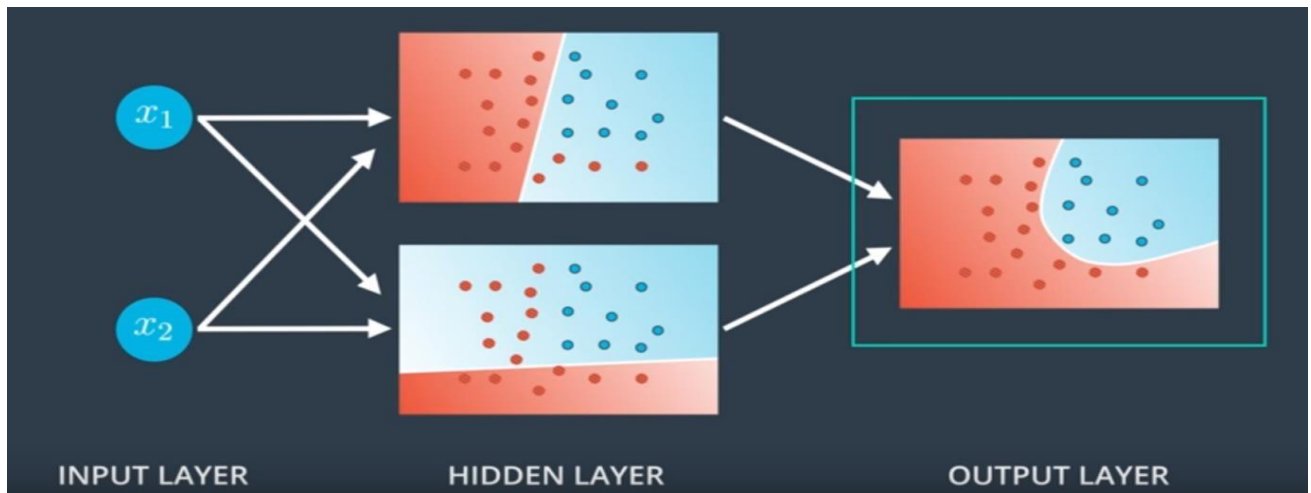


Fig. 18

### 3.7 Neural Network Training:

Before correctly categorizing images the first step is to train the neural network properly.

Training a Neural network is typically done in 2 phases:

- Feed-Forward Propagation
- Back Propagation

#### 1. Feed-Forward Algorithm: -

In feedforward propagation the flow of information is in the forward direction,  $x$  is used to calculate some intermediate function in the hidden layer which is then used to compute  $y$ . These networks are characterized by the construction of many different functions. Each model is related to a graph that describes how the functions are constructed.

In general, the feedforward process in a neural network is used to convert inputs into outputs. The inputs are first multiplied by their weights and then the activation function is applied to these dot products. The output of the activation function is input to the hidden layer and then these inputs undergo dot-product with their weights again and so the output is the input for the hidden layer if any more hidden layers are there, and if not, then these outputs go directly to the output layer.

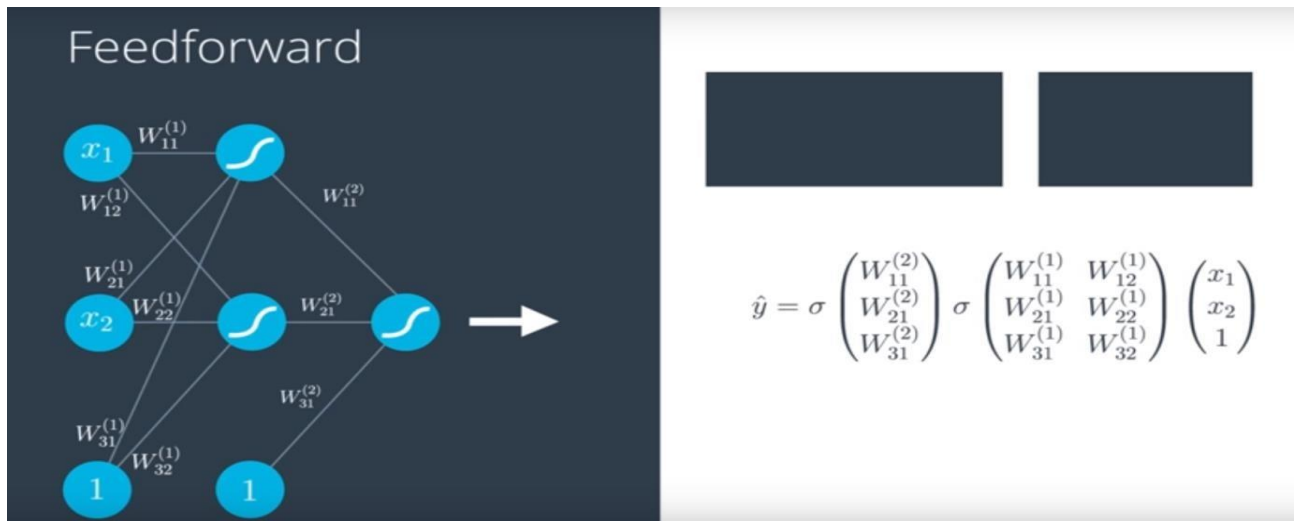


Fig. 19

## 2. Backpropagation:

For now we'll only consider a simple network with one hidden layer and one output unit. Here's the general algorithm for updating the weights with backpropagation:

- Set the weight steps for each layer to zero
  - The input to hidden weights  $\Delta w_{ij} = 0$
  - The hidden to output weights  $\Delta W_j = 0$
- For each record in the training data:
  - Make a forward pass through the network, calculating the output  $\hat{y}$
  - Calculate the error gradient in the output unit,  $\delta^o = (y - \hat{y})f'(z)$  where  $z = \sum_j W_j a_j$ , the input to the output unit.
  - Propagate the errors to the hidden layer  $\delta_j^h = \delta^o W_j f'(h_j)$
  - Update the weight steps:
    - $\Delta W_j = \Delta W_j + \delta^o a_j$
    - $\Delta w_{ij} = \Delta w_{ij} + \delta_j^h a_i$
- Update the weights, where  $\eta$  is the learning rate and  $m$  is the number of records:
  - $W_j = W_j + \eta \Delta W_j / m$
  - $w_{ij} = w_{ij} + \eta \Delta w_{ij} / m$
- Repeat for  $e$  epochs.

Fig. 20

### 3.8 Measures to increase efficiency:

- **Early stopping:** With each training epoch, loss decreases but on running validated data to check the performance of the trained model, loss decreases for a while but then the loss again starts increasing. Because the data has been trained over and over multiple times it has been overfitted, which gives good result on training data but on validation data the performance is really bad.

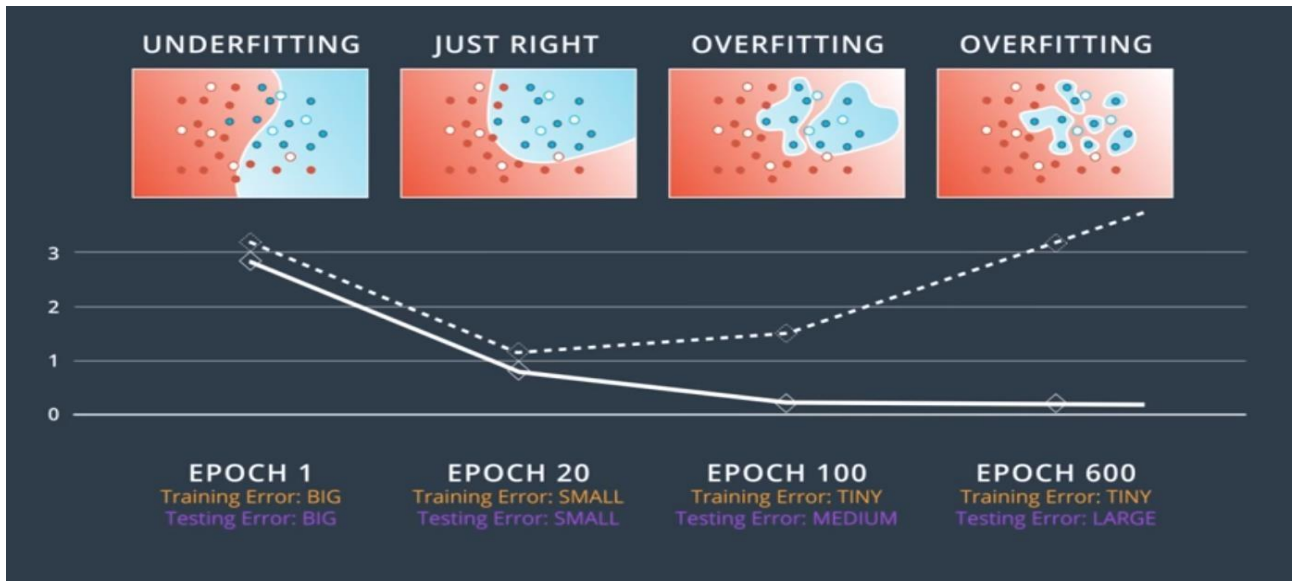


Fig. 21

To avoid overfitting, the epochs count is decreased and is stopped as soon as the loss of validation data from the dataset starts to increase.

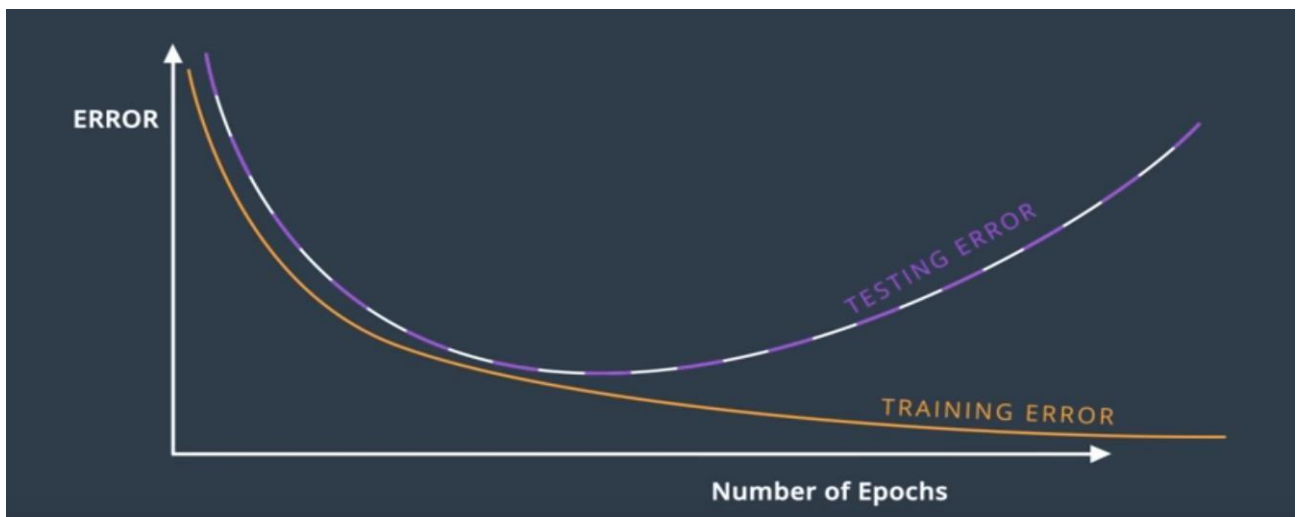


Fig. 22

- **Dropout Method:**In this method, some nodes chosen randomly are not trained on each epoch, to allow some of the other different nodes to become extra involved and get stronger. This node selection is completely random and involves regulating the probabilities, such as to 1/5, 1/3.
- **Local Minima Method:**If we calculate the gradient descent, we may not be able to set the global minima every time, we can sometimes encounter the local minima. To avoid this, one can start getting the from random places to calculate the gradient descent.

### 3.9 Pytorch implementation of Neural Networks(NN)

```

class Network(nn.Module):
    def __init__(self):
        super().__init__()
        # Defining the layers, 128, 64, 10 units each
        self.fc1 = nn.Linear(784, 128)
        self.fc2 = nn.Linear(128, 64)
        # Output layer, 10 units - one for each digit
        self.fc3 = nn.Linear(64, 10)

    def forward(self, x):
        ''' Forward pass through the network, returns the output logits '''

        x = self.fc1(x)
        x = F.relu(x)
        x = self.fc2(x)
        x = F.relu(x)
        x = self.fc3(x)
        x = F.softmax(x, dim=1)

        return x

model = Network()
model

```

Fig. 23

In nn.Sequential, defining a class is not necessary.

```
model = nn.Sequential(nn.Linear(784, 128),  
                      nn.ReLU(),  
                      |  
                      nn.Linear(128, 64),  
                      nn.ReLU(),  
                      nn.Linear(64, 10))
```

Fig. 24

### **Loss Calculation in Pytorch Library:**

nn module provides us a mechanism to calculate the loss in PyTorch. Losses like cross-entropy are assigned to a criterion, and to calculate the loss, we define and pass in a criterion as the o/p of the network with the correctly defined labels to calculate the loss. The output of this whole procedure (the criterion) is the calculated loss, which is to be minimized.

### **Autograd Module:**

The tensor gradient is computed using a module called Autograd. It is used to compute the gradient values of our weight w.r.t loss. It keeps the record of all the operations that are performed on the tensors, then backpropagates over these operations to calculate the gradient along each path. *'requires\_grad'* is set as True for the tensor, so as to allow PyTorch to keep track and calculate the gradients.

Calculating loss and applying autograd:



```

epochs = 5
for e in range(epochs):
    running_loss = 0
    for images, labels in trainloader:
        images = images.view(images.shape[0], -1)
        optimizer.zero_grad()
        output = model.forward(images)
        loss = criterion(output, labels)
        loss.backward()
        optimizer.step()

    running_loss += loss.item()
else:
    print(f"Training loss: {running_loss/len(trainloader)}")

```

Fig. 25

In the above mentioned code, the loss is computed by criterion, the gradient is computed by `loss.backward()` and the weights are updated using `optimizer.step()`.

### Validation and Inference for better results:

`nn.dropout` allows the nodes to learn better, which allows us to increase the accuracy of our model in PyTorch.

```

class Classifier(nn.Module):
    def __init__(self):
        super().__init__()
        self.fc1 = nn.Linear(784, 256)
        self.fc2 = nn.Linear(256, 128)
        self.fc3 = nn.Linear(128, 64)
        self.fc4 = nn.Linear(64, 10)

        # Dropout module with 0.2 drop probability
        self.dropout = nn.Dropout(p=0.2)

```

Fig. 26

For validation, the model is checked with unseen images and with gradient off because the aim here is to find the accuracy of the model and not training. `model.eval()` directs the model for evaluation purposes. Images in model are loaded in same way as of training and the



validation loss is calculated. Then probability for each output class is calculated and finds the class with maximum probability and compares it with the label. This way accuracy and validation loss is calculated.

### Loading Input from the Image Datasets:

```
data_dir = 'Cat_Dog_data'

train_transforms = transforms.Compose([transforms.RandomRotation(30),
                                       transforms.RandomResizedCrop(224),
                                       transforms.RandomHorizontalFlip(),
                                       transforms.ToTensor()])

test_transforms = transforms.Compose([transforms.Resize(255),
                                       transforms.CenterCrop(224),
                                       transforms.ToTensor()])

train_data = datasets.ImageFolder(data_dir + '/train', transform=train_transforms)
test_data = datasets.ImageFolder(data_dir + '/test', transform=test_transforms)

trainloader = torch.utils.data.DataLoader(train_data, batch_size=32)
testloader = torch.utils.data.DataLoader(test_data, batch_size=32)
```

Fig. 27

The datasets module in PyTorch in torchvision is used to load the training and testing data separately. The directory is imported using the method ImageFolder and the data is pushed onto the Dataloader method in torch.utils after loading the whole data, while also giving the batch size as an argument.

## Transfer Learning :

ImageNet a huge dataset with more than 10 lakhs correctly labelled pictures in thousand categories. A neural network model using convolutional layers can be trained using ImageNet. The model works very efficiently as features detector on unknown images. Transfer learning is using pre-trained models on our own model to get results with high accuracy. It is useful for datasets where the size of the dataset is low. In this case transfer learning comes in to train our model so as to get highly efficient results.

Pretrained models can be imported using torchvision.models and pretrained=True has to be given in the argument. When these pre-trained models are loaded, we take the parameters and halt it at those values. The input layer, hidden layer and the output layer all need to be defined before getting the parameters, as getting the layers is not an automatic fare and has to be done manually.

```
model = models.densenet121(pretrained=True)
|
for param in model.parameters():
    param.requires_grad = False

model.classifier = nn.Sequential(nn.Linear(1024, 256),
                                nn.ReLU(),
                                nn.Dropout(0.2),
                                nn.Linear(256, 2),
                                nn.LogSoftmax(dim=1))
```

Fig. 28

### 3.10 ALGORITHMS :

#### 1<sup>st</sup> Algorithm: **Model Training**

def model\_train (input data, model, hyperparameters)

-> Describe Data by means of image sets

-> Compute dataset sizes

-> define the appropriate model

-> hardware = GPU (preferred)/CPU

-> train the model

-> stats

#### Algorithm 2: **Image Processing**

To make the specifications of all the images same.

def process\_pic(pic):

-> pic open

-> pic size changed

-> pic cropped (centre)

-> tensor

-> normalize

-> tensor\_image = adjust(pic)

-> return image\_new

## CHAPTER 4: PERFORMANCE ANALYSIS

First, all the libraries are to be imported which we be using to design our model, so as to avoid errors of function not found.

Before compiling, import each and every library which will be used to make the work easier:

```
In [2]: import numpy as np
import matplotlib
matplotlib.use('agg')
import matplotlib.pyplot as plt

import os
os.environ['TF_CPP_MIN_LOG_LEVEL']='3'
# for testing on CPU
#os.environ['CUDA_VISIBLE_DEVICES'] = ''

from keras.datasets import mnist
from keras.models import Sequential, load_model
from keras.layers.core import Dense, Dropout, Activation
from keras.utils import np_utils
```

Fig. 29

### 4.1 Command line testing

Command line can be used for compiling, executing and also for sending the parameters to be used for the model manually to the program.

- >Using the file name directory can be forwarded to the program.
- >model to be used is also passable from the command line.
- > hyperparameter values can be passed on for the model training.
- >GPU's can be selected
- >The k best cases can be run and applied

## 4.2 Dataset testing

The training and testing data needs to be divided adequately to build an efficient model with high accuracy. The dataset will be split into the following sets:

1. Train Dataset
2. Validate Dataset
3. Test Dataset

To train the model we use the Training dataset which will work the best if the amount of data for training is high. Higher amount of training data gives the best result and the best accuracy for the model.

Train and testing data need to be split into a ratio so as to test for high accuracy after training the model and before applying never seen before real world data.

Usually we use the 80:20 ratio for training and testing data respectively. This gives us a good result as 80% data is quite enough for training our model with high precision.

```
In [6]: # Shape
print("X_train shape", X_train.shape)
print("y_train shape", y_train.shape)
print("X_test shape", X_test.shape)
print("y_test shape", y_test.shape)

# building the input vector from the 28x28 pixels
X_train = X_train.reshape(60000, 784)
X_test = X_test.reshape(10000, 784)
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')

# normalizing
X_train /= 255
X_test /= 255

# Shape after preprocessing
print("Train matrix shape", X_train.shape)
print("Test matrix shape", X_test.shape)

X_train shape (60000, 28, 28)
y_train shape (60000,)
X_test shape (10000, 28, 28)
y_test shape (10000,)
Train matrix shape (60000, 784)
Test matrix shape (10000, 784)
```

Fig. 30

## 4.3 Code Performance

- Importing and inputting the data

```
In [3]: (X_train, y_train), (X_test, y_test) = mnist.load_data()
```

```
In [4]: fig = plt.figure()
for i in range(9):
    plt.subplot(3,3,i+1)
    plt.tight_layout()
    plt.imshow(X_train[i], cmap='gray', interpolation='none')
    plt.title("Digit: {}".format(y_train[i]))
    plt.xticks([])
    plt.yticks([])
fig
```

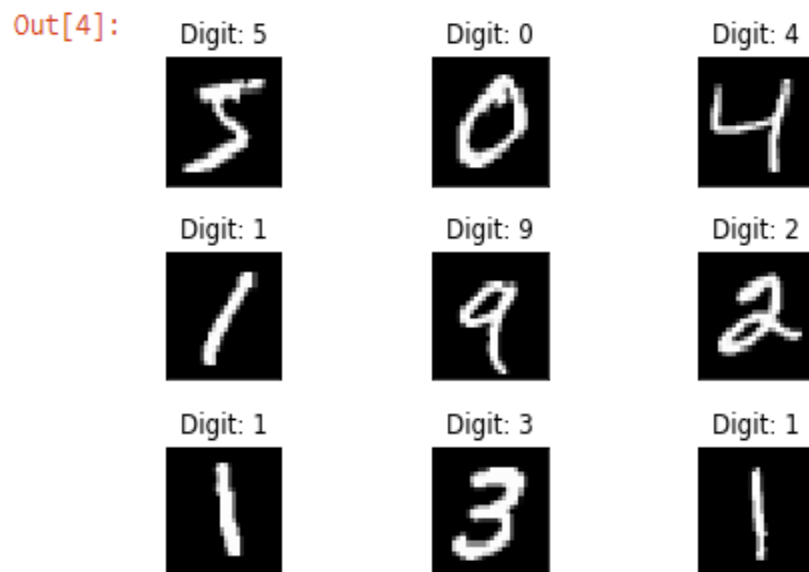


Fig. 31

- Creating a sequential model in the following order:

Dense convolutional layer -> activation layer (relu) -> dropout->Dense convolutional layer -> activation layer(relu) -> dropout layer -> dense convolutional layer -> activation layer (softmax)

```
In [9]: # building a linear stack of layers with the sequential model
model = Sequential()
model.add(Dense(512, input_shape=(784,)))
model.add(Activation('relu'))
model.add(Dropout(0.2))

model.add(Dense(512))
model.add(Activation('relu'))
model.add(Dropout(0.2))

model.add(Dense(10))
model.add(Activation('softmax'))
```

Fig. 32

- Running the sequential model for 20 epochs to get the optimal value for the weights and biases of the model and using them to calculate the optimal loss value and cost function.

```
In [11]: # training the model and saving metrics in history
history = model.fit(X_train, Y_train,
                    batch_size=128, epochs=20,
                    verbose=2,
                    validation_data=(X_test, Y_test))

model.save('mnist_prev.h5')
```

Fig. 33

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
- 13s - loss: 0.2458 - acc: 0.9260 - val_loss: 0.1079 - val_acc: 0.9638
Epoch 2/20
- 12s - loss: 0.1016 - acc: 0.9681 - val_loss: 0.0882 - val_acc: 0.9731
Epoch 3/20
- 11s - loss: 0.0710 - acc: 0.9773 - val_loss: 0.0708 - val_acc: 0.9771
Epoch 4/20
- 11s - loss: 0.0548 - acc: 0.9823 - val_loss: 0.0687 - val_acc: 0.9785
Epoch 5/20
- 12s - loss: 0.0475 - acc: 0.9850 - val_loss: 0.0688 - val_acc: 0.9796
Epoch 6/20
- 12s - loss: 0.0376 - acc: 0.9878 - val_loss: 0.0639 - val_acc: 0.9806
Epoch 7/20
- 13s - loss: 0.0329 - acc: 0.9890 - val_loss: 0.0611 - val_acc: 0.9832
Epoch 8/20
- 12s - loss: 0.0287 - acc: 0.9906 - val_loss: 0.0587 - val_acc: 0.9835
Epoch 9/20
- 11s - loss: 0.0283 - acc: 0.9903 - val_loss: 0.0704 - val_acc: 0.9815
Epoch 10/20
- 11s - loss: 0.0240 - acc: 0.9921 - val_loss: 0.0624 - val_acc: 0.9827
Epoch 11/20
- 11s - loss: 0.0247 - acc: 0.9919 - val_loss: 0.0724 - val_acc: 0.9810
Epoch 12/20
- 11s - loss: 0.0235 - acc: 0.9923 - val_loss: 0.0762 - val_acc: 0.9818
```

Fig. 34

- Calculating the accuracy of the model

```
In [12]: mnist_model = load_model('mnist_prev.h5')
loss_and_metrics = mnist_model.evaluate(X_test, Y_test, verbose=2)

print("Test Loss", loss_and_metrics[0])
print("Test Accuracy", loss_and_metrics[1])
```

```
Test Loss 0.06776916432169709
Test Accuracy 0.9837
```

Fig. 35



- Output

9837 classified correctly  
163 classified incorrectly

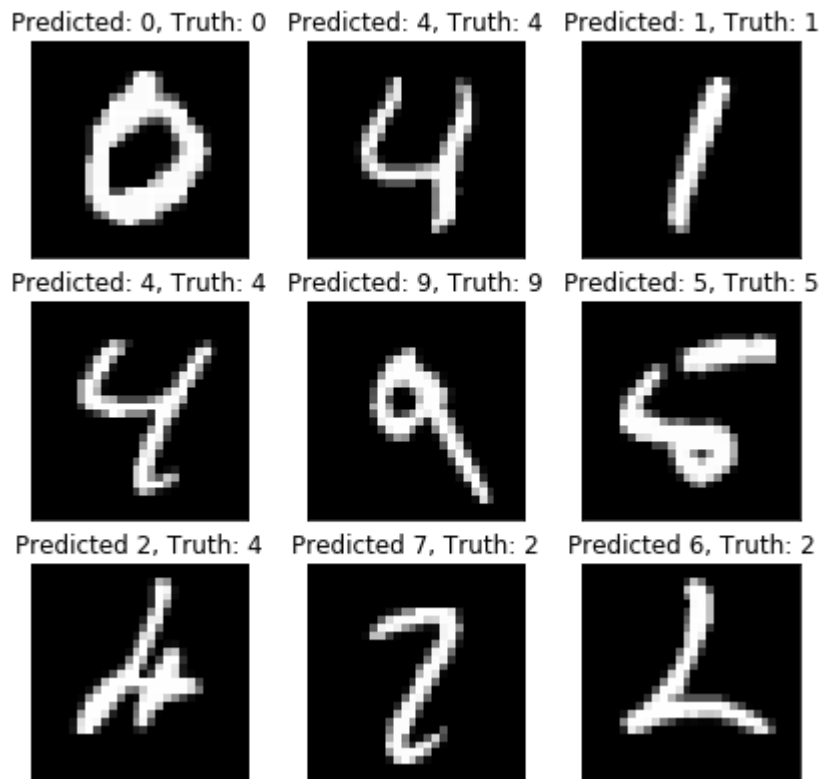


Fig. 36

**My model gives a test accuracy of 98.37 % with a loss value of 0.0677**

## **CHAPTER 5: CONCLUSION**

During the duration of this project, I have modelled and applied various different sorts of machine learning models, some of them were self-modelled from scratch while others were pre-trained (already) models and each model gives me a different accuracy measure for the same data, for eg; these self-designed models gave an accuracy ranging between 70 and 80 percent for the same data, then applying some methods to increase the accuracy by improving my model using controlled epochs, validation data, dropout modules I was able to increase the accuracy range between 80 to 85 percent. To further increase the accuracy of my model, I used a pre-trained neural network, to train the same dataset using deep learning concepts to increase the efficiency. Using neural networks, I increased my accuracy to >90%.

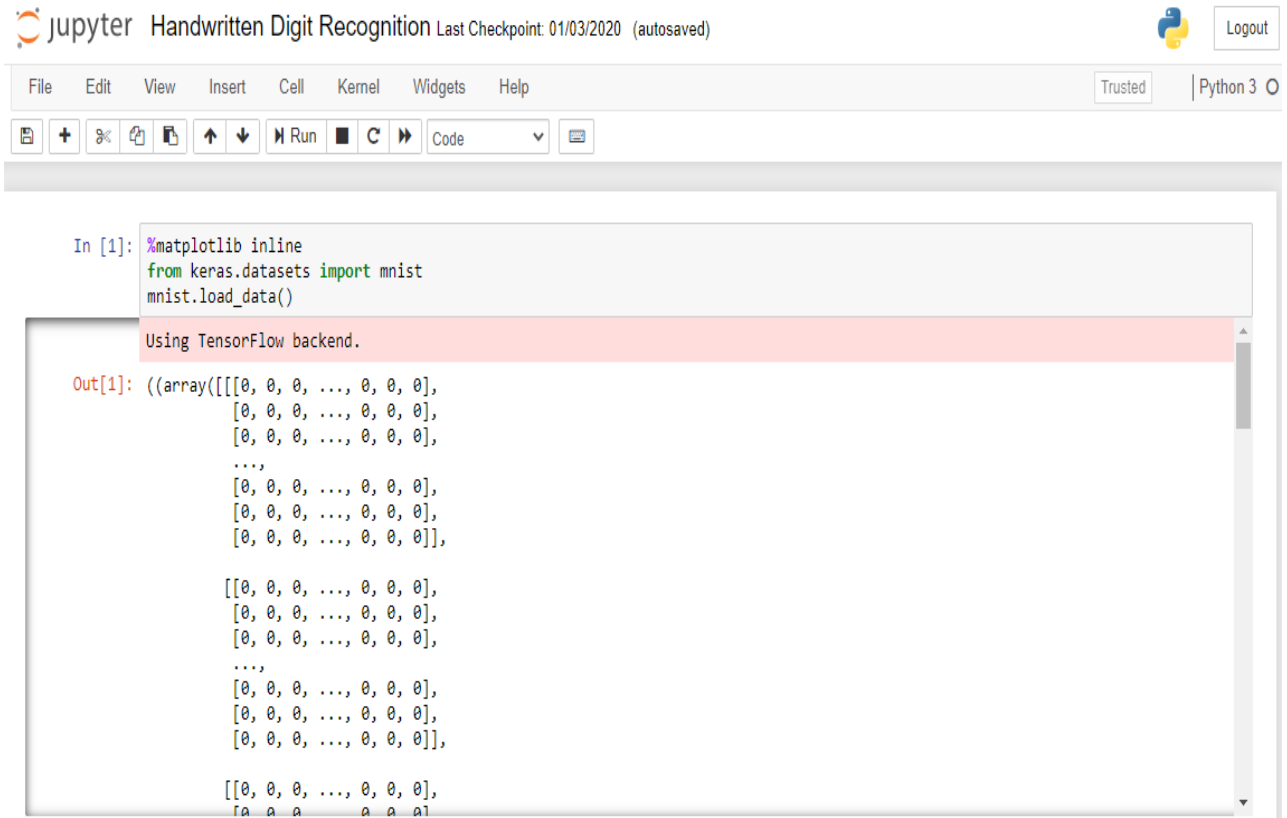
Finally, I used the Keras and TensorFlow Deep Learning libraries to implement a sequential model of dense neural network layers, activated the output using ReLU and then passed it through a dropout layer to avoid overfitting, for a few cycles, and finally applied the softmax function to get the class of the image. Using this model, I finally attained an accuracy of 98.37%.

To conclude, for classification of images, if the input data is less for the model to train properly we can use transfer learning to get the parameters from various already trained models, and if the input data is sufficient then we can use that data to train our own model, whether building it from scratch or using pre-defined ML or DL libraries is up to us. But using the multiple deep learning and machine learning libraries available to us saves our time and help us in getting better accuracy for our model.

## References

- [1] IEEE paper: <https://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=6973086>
  
- [2] (Online Resource)<https://www.einfochips.com/blog/understanding-image-recognition-and-its-uses/>
  
- [3] (Online Resource).<https://medium.com/intro-to-artificial-intelligence/simple-image-classification-using-deep-learning-deep-learning-series-2-5e5b89e97926>
  
- [4] <https://medium.com/@sidereal/cnns-architectures-lenet-alexnet-vgg-googlenet-resnet-and-more-666091488df5>
  
- [5] (Online Resources) <https://skymind.ai/wiki/convolutional-network>
  
- [6] (Online Resources) <https://udacity.com>
  
- [7] (Online Resources) Khan Academy

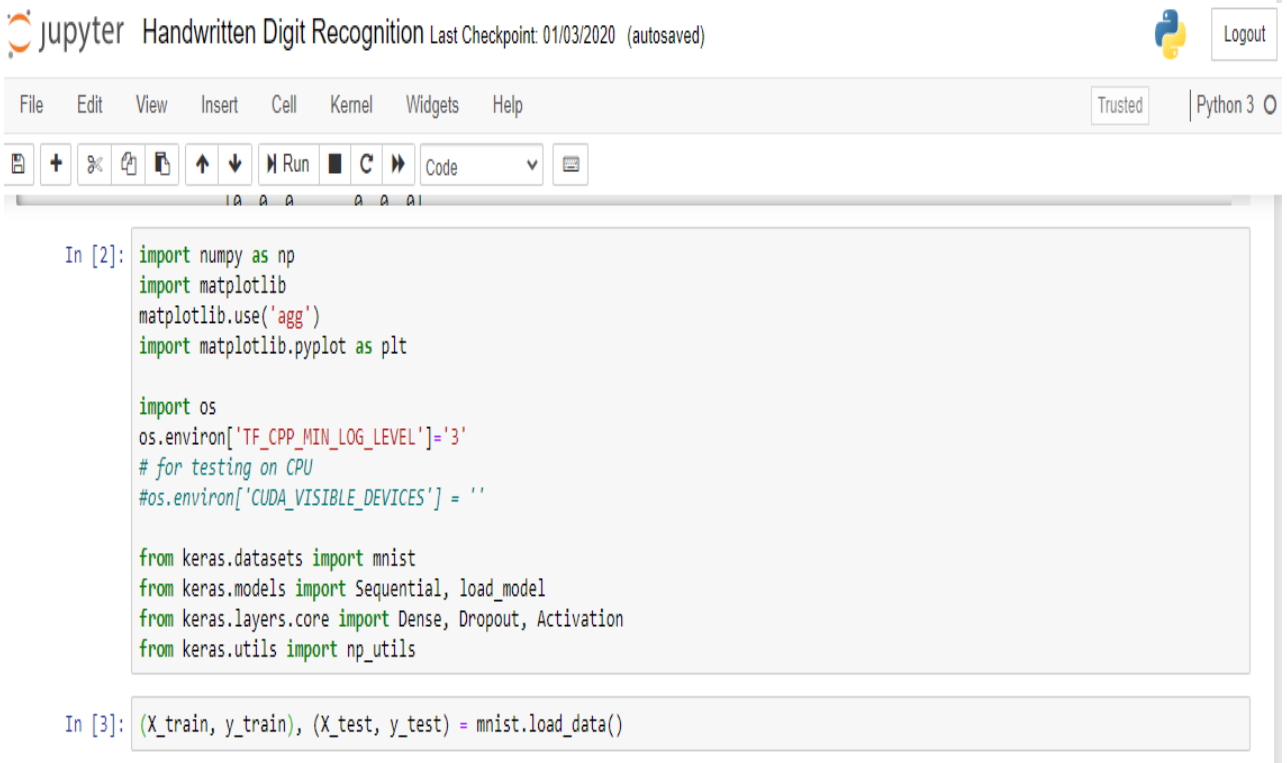
## Program Screenshots:



```
In [1]: %matplotlib inline
from keras.datasets import mnist
mnist.load_data()

Using TensorFlow backend.

Out[1]: ((array([[0, 0, 0, ..., 0, 0, 0],
                [0, 0, 0, ..., 0, 0, 0],
                [0, 0, 0, ..., 0, 0, 0],
                ...,
                [0, 0, 0, ..., 0, 0, 0],
                [0, 0, 0, ..., 0, 0, 0],
                [0, 0, 0, ..., 0, 0, 0]],
               [[0, 0, 0, ..., 0, 0, 0],
                [0, 0, 0, ..., 0, 0, 0],
                [0, 0, 0, ..., 0, 0, 0],
                ...,
                [0, 0, 0, ..., 0, 0, 0],
                [0, 0, 0, ..., 0, 0, 0],
                [0, 0, 0, ..., 0, 0, 0]],
               [[0, 0, 0, ..., 0, 0, 0],
                [0, 0, 0, ..., 0, 0, 0],
                [0, 0, 0, ..., 0, 0, 0]]))
```

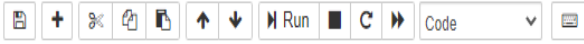


```
In [2]: import numpy as np
import matplotlib
matplotlib.use('agg')
import matplotlib.pyplot as plt

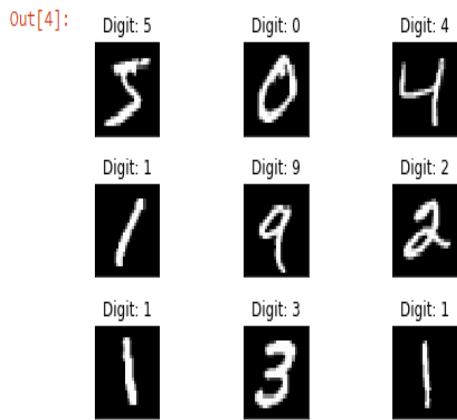
import os
os.environ['TF_CPP_MIN_LOG_LEVEL']='3'
# for testing on CPU
#os.environ['CUDA_VISIBLE_DEVICES'] = ''

from keras.datasets import mnist
from keras.models import Sequential, load_model
from keras.layers.core import Dense, Dropout, Activation
from keras.utils import np_utils

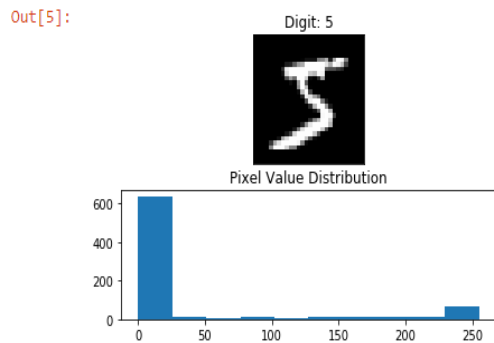
In [3]: (X_train, y_train), (X_test, y_test) = mnist.load_data()
```

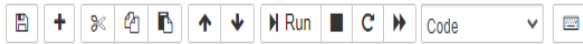


```
In [4]: fig = plt.figure()
for i in range(9):
    plt.subplot(3,3,i+1)
    plt.tight_layout()
    plt.imshow(X_train[i], cmap='gray', interpolation='none')
    plt.title("Digit: {}".format(y_train[i]))
    plt.xticks([])
    plt.yticks([])
fig
```



```
In [5]: fig = plt.figure()
plt.subplot(2,1,1)
plt.imshow(X_train[0], cmap='gray', interpolation='none')
plt.title("Digit: {}".format(y_train[0]))
plt.xticks([])
plt.yticks([])
plt.subplot(2,1,2)
plt.hist(X_train[0].reshape(784))
plt.title("Pixel Value Distribution")
fig
```





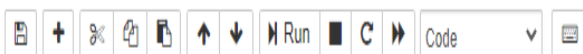
```
In [6]: # Shape
print("X_train shape", X_train.shape)
print("y_train shape", y_train.shape)
print("X_test shape", X_test.shape)
print("y_test shape", y_test.shape)

# building the input vector from the 28x28 pixels
X_train = X_train.reshape(60000, 784)
X_test = X_test.reshape(10000, 784)
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')

# normalizing
X_train /= 255
X_test /= 255

# Shape after preprocessing
print("Train matrix shape", X_train.shape)
print("Test matrix shape", X_test.shape)
```

```
X_train shape (60000, 28, 28)
y_train shape (60000,)
X_test shape (10000, 28, 28)
y_test shape (10000,)
Train matrix shape (60000, 784)
Test matrix shape (10000, 784)
```



```
In [7]: print(np.unique(y_train, return_counts=True))

(array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9], dtype=uint8), array([5923, 6742, 5958, 6131, 5842, 5421, 5918, 6265, 5851, 5949],
dtype=int64))
```

```
In [8]: n_classes = 10
print("Shape before one-hot encoding: ", y_train.shape)
Y_train = np_utils.to_categorical(y_train, n_classes)
Y_test = np_utils.to_categorical(y_test, n_classes)
print("Shape after one-hot encoding: ", Y_train.shape)
```

```
Shape before one-hot encoding: (60000,)
Shape after one-hot encoding: (60000, 10)
```



```
In [9]: # building a linear stack of layers with the sequential model
model = Sequential()
model.add(Dense(512, input_shape=(784,)))
model.add(Activation('relu'))
model.add(Dropout(0.2))

model.add(Dense(512))
model.add(Activation('relu'))
model.add(Dropout(0.2))

model.add(Dense(10))
model.add(Activation('softmax'))
```

WARNING:tensorflow:From C:\Users\DELL\Anaconda3\envs\tensorflow\lib\site-packages\tensorflow\python\framework\op\_def\_library.py:263: colocate\_with (from tensorflow.python.framework.ops) is deprecated and will be removed in a future version.  
 Instructions for updating:  
 Colocations handled automatically by placer.  
 WARNING:tensorflow:From C:\Users\DELL\Anaconda3\envs\tensorflow\lib\site-packages\keras\backend\tensorflow\_backend.py:3445: calling dropout (from tensorflow.python.ops.nn\_ops) with keep\_prob is deprecated and will be removed in a future version.  
 Instructions for updating:  
 Please use `rate` instead of `keep\_prob`. Rate should be set to `rate = 1 - keep\_prob`.

```
In [10]: model.compile(loss='categorical_crossentropy', metrics=['accuracy'], optimizer='adam')
```



```
In [11]: # training the model and saving metrics in history
history = model.fit(X_train, Y_train,
                    batch_size=128, epochs=20,
                    verbose=2,
                    validation_data=(X_test, Y_test))

model.save('mnist_prev.h5')

# plotting the metrics
fig = plt.figure()
plt.subplot(2,1,1)
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='lower right')

plt.subplot(2,1,2)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper right')

plt.tight_layout()

fig
```



File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

```
In [12]: mnist_model = load_model('mnist_prev.h5')
loss_and_metrics = mnist_model.evaluate(X_test, Y_test, verbose=2)

print("Test Loss", loss_and_metrics[0])
print("Test Accuracy", loss_and_metrics[1])

Test Loss 0.06776916432169709
Test Accuracy 0.9837
```

```
In [13]: # Load the model and create predictions
mnist_model = load_model('mnist_prev.h5')
predicted_classes = mnist_model.predict_classes(X_test)

correct_indices = np.nonzero(predicted_classes == y_test)[0]
incorrect_indices = np.nonzero(predicted_classes != y_test)[0]
print()
print(len(correct_indices), " classified correctly")
print(len(incorrect_indices), " classified incorrectly")

# adapt figure size to accomodate 18 subplots
plt.rcParams['figure.figsize'] = (7,14)
figure_evaluation = plt.figure()

# plotting 9 correct predictions
for i, correct in enumerate(correct_indices[:9]):
    plt.subplot(6,3,i+1)
    plt.imshow(X_test[correct].reshape(28,28), cmap='gray', interpolation='none')
    plt.title(
        "Predicted: {}, Truth: {}".format(predicted_classes[correct],
        y_test[correct]))

plt.xticks([])
plt.yticks([])

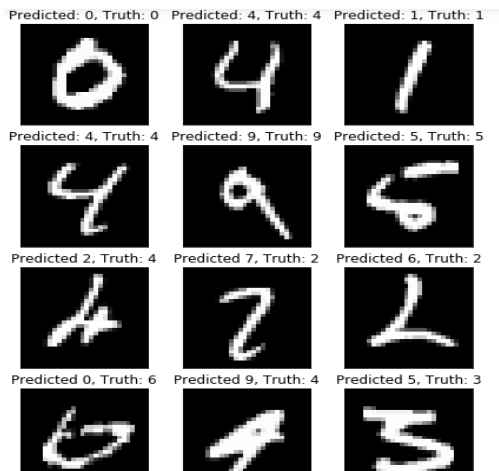
# plotting 9 incorrect predictions
for i, incorrect in enumerate(incorrect_indices[:9]):
    plt.subplot(6,3,i+10)
    plt.imshow(X_test[incorrect].reshape(28,28), cmap='gray', interpolation='none')
    plt.title(
        "Predicted {}, Truth: {}".format(predicted_classes[incorrect],
        y_test[incorrect]))

plt.xticks([])
plt.yticks([])

figure_evaluation
```

9837 classified correctly  
163 classified incorrectly

Out[13]: Predicted: 7, Truth: 7 Predicted: 2, Truth: 2 Predicted: 1, Truth: 1



# IMAGE CLASSIFICATION USING NEURAL NETWORKS

---

## ORIGINALITY REPORT

---

<b>14%</b>	<b>0%</b>	<b>2%</b>	<b>14%</b>
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

---

## PRIMARY SOURCES

---

<b>1</b>	Submitted to Jaypee University of Information Technology Student Paper	<b>12%</b>
<b>2</b>	Submitted to University of Surrey Student Paper	<b>1%</b>
<b>3</b>	Submitted to University of Ulster Student Paper	<b>1%</b>
<b>4</b>	Submitted to Harrisburg University of Science and Technology Student Paper	<b>&lt;1%</b>
<b>5</b>	Submitted to Thammasat University Student Paper	<b>&lt;1%</b>
<b>6</b>	Yu, L.. "Credit risk assessment with a multistage neural network ensemble learning approach", Expert Systems With Applications, 200802 Publication	<b>&lt;1%</b>
<b>7</b>	Hisham El-Amir, Mahmoud Hamdy. "Deep Learning Pipeline", Springer Science and Business Media LLC, 2020 Publication	<b>&lt;1%</b>

---

8

Gefei Yang, Utsav B. Gewali, Emmett Ientilucci, Micheal Gartley, Sildomar T. Monteiro. "Dual-Channel Densenet for Hyperspectral Image Classification", IGARSS 2018 - 2018 IEEE International Geoscience and Remote Sensing Symposium, 2018

Publication

---

<1%

---

Exclude quotes Off

Exclude matches Off

Exclude bibliography Off

**JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY, WAKNAGHAT**

**PLAGIARISM VERIFICATION REPORT**

Date: 15/07/2020

Type of Document (Tick):  PhD Thesis  M.Tech Dissertation/ Report  B.Tech Project Report  Paper

Name: Pradyut Lohani Department: Computer Science Enrolment No 161313

Contact No. 9582465255 E-mail. pradyut.lohani@gmail.com

Name of the Supervisor: Dr. Rakesh Kanji

Title of the Thesis/Dissertation/Project Report/Paper (In Capital letters): \_\_\_\_\_  
IMAGE CLASSIFICATION USING NEURAL NETWORKS

**UNDERTAKING**

I undertake that I am aware of the plagiarism related norms/ regulations, if I found guilty of any plagiarism and copyright violations in the above thesis/report even after award of degree, the University reserves the rights to withdraw/ revoke my degree/report. Kindly allow me to avail Plagiarism verification report for the document mentioned above.

**Complete Thesis/Report Pages Detail:**

- Total No. of Pages = 51
- Total No. of Preliminary pages = 5
- Total No. of pages accommodate bibliography/references = 1

*Pradyut*  
(Signature of Student)

**FOR DEPARTMENT USE**

We have checked the thesis/report as per norms and found **Similarity Index** at .....14.....(%). Therefore, we are forwarding the complete thesis/report for final plagiarism check. The plagiarism verification report may be handed over to the candidate.

*Rakesh Kanji*  
(Signature of Guide/Supervisor)

Signature of HOD

**FOR LRC USE**

The above document was scanned for plagiarism check. The outcome of the same is reported below:

Copy Received on	Excluded	Similarity Index (%)	Generated Plagiarism Report Details (Title, Abstract & Chapters)	
	<ul style="list-style-type: none"> <li>• All Preliminary Pages</li> <li>• Bibliography/Images/Quotes</li> <li>• 14 Words String</li> </ul>		Word Counts	
<b>Report Generated on</b>			Character Counts	
		<b>Submission ID</b>	Total Pages Scanned	
			File Size	

Checked by  
Name & Signature

Librarian

.....

**Please send your complete thesis/report in (PDF) with Title Page, Abstract and Chapters in (Word File) through the supervisor at [plagcheck.juit@gmail.com](mailto:plagcheck.juit@gmail.com)**