IMAGE CLASSIFICATION USING NEURAL NETWORKS

Project report submitted in partial fulfillment of the requirement for the degree of Bachelor of Technology

in

Computer Science and Engineering/Information Technology

By

Vishal Duhan (151398)

Under the supervision of

Dr.Geetanjali (Assistant Professor-Senior Grade)

to

Department of Computer Science & Engineering and Information Technology

Jaypee University of Information Technology Waknaghat, Solan-173234, Himachal Pradesh

Candidate's Declaration

We hereby declare that the work presented in this report entitled **" Image classification using Neural Networks"** in partial fulfillment of the requirements for the award of the degree of **Bachelor of Technology** in **Computer Science and Engineering/Information Technology** submitted in the department of Computer Science & Engineering and Information Technology**,** Jaypee University of Information Technology Waknaghat is an authentic record of my own work carried out over a period from August 2018 to May 2019 under the supervision of **Dr. Geetanjali** (**Assistant Professor - Senior Grade(**CSE)).

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

Vishal Duhan(151398)

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

Dr. Geetanjali

Assistant Professor

CSE

Dated: 10 May 2019

ACKNOWLEDGEMENT

We take this opportunity to express our profound gratitude and deep regards to our guide Dr. Geetanjali(Assistant Professor , CSE) for her exemplary guidance, monitoring and constant encouragement throughout the course of this project. The blessing, help and guidance given by her time to time shall carry us a long way in the journey of life on which we are about to embark.

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

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

Contents

List of Figures

List of Tables

List of Abbreviations

1. ANN – Artificial Neural Networks
2. CNN – Convolutional Neural Networks
3. CVPR - Computer Vision & Pattern Recognition
4. OCR - Optical Character Recognition
5. SVM – Support Vector Machine

ABSTRACT

As 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 should 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 a image classifier to classify distinctive types of flowers. You can imagine utilizing something like this in a phone application that reveals to you the name of the flower your camera is taking a look at. In practice, you'd train this classifier, at that point send out it for use in your application. We'll be utilizing this dataset of around 100 flower categories.

When we will finish this undertaking, we'll have an application that can be trained on any arrangement of marked pictures. Here your system will find out about flowers and end up as an command line application. In any case, the use of the task is altogether subject to the client.

CHAPTER 1: INTRODUCTION

1.1 Introduction

When it comes to identifying pictures, we people can obviously perceive and recognize distinctive features of items. This is on the grounds that our brains have been trained unconsciously with a similar arrangement of pictures that has brought about the advancement of capabilities to separate between things easily. We are not really conscious when we interpret the real world. Experiencing distinctive elements of the visual world and recognizing effortlessly is a no test to us. Our subconscious mind does every one of the process with no issue.

In opposition to human brains, computer sees visuals as a variety of numerical qualities and searches for patterns in the advanced picture, be it a still, video, realistic, or even live, to perceive and distinguish key highlights of the picture. The way in which a system interprets a picture is totally not quite the same as humans. Computer vision utilizes picture handling calculations to analyze and understand visuals from a solitary picture or an arrangement of pictures. A case of computer vision is distinguishing people on foot and vehicles out and about by, arranging and sifting a great many user uploaded pictures with accuracy.

Thinking about the developing capability of computer vision, numerous organisation are investing into image recognition to analyze and investigate information coming essentially from visual sources for various uses, for example, medicinal image examination, identifying objects in independent vehicles, face detection for security reason, and so forth.

Image recognition is the ability of a software or system to identify objects, individuals, places, and activities in images. It utilizes machine vision advances with artificial intelligence and prepared calculations to recognize images through a camera system. Much fuelled by the ongoing progressions in machine learning and an expansion in the computational intensity of the machines, image recognition has surprised the world. Car, internet business, retail, producing ventures, security, observation, social insurance, cultivating and so on., can have a wide utilization of image recognition.

Uses-

Drones:

Image recognition capabilities equipped drones can provide inspection, vision-based automatic monitoring, and control in remote area.

Manufacturing:

Investigating creation lines, assessing basic focuses all the time inside the premises. Observing the nature of the last items to lessen the imperfections. Evaluating the state of the laborers can help fabricating enterprises to have a total control of various exercises in the frameworks.

Autonomous Vehicles:

Self-sufficient vehicles with image recognition can distinguish exercises out and about and take important activities. Smaller than expected robots can help coordinations ventures to find and exchange the articles starting with one place then onto the next. It additionally keeps up the database of the item development history to keep the item from being lost or stolen.

Military Surveillance:

Identification of irregular exercises in the fringe regions and programmed basic leadership abilities can help counteract invasion and bring about sparing the lives of officers.

CNN- Convolutional neural networks are deep neural networks that are utilized to arrange pictures , bunch them with similarity , and perform object recognition. They are algorithms that can recognize things, people, road signs, platypuses and numerous different parts of information.

Convolutional networks perform optical character recognition (OCR) to digitize content and make natural-language processing possible on simple and written by hand reports, where the pictures are images to be interpreted. CNNs can likewise be connected to sound when it is spoken to outwardly as a spectrogram. All the more as of late, convolutional networks have been connected specifically to content examination and graph information with graph convolutional networks.
The software industry in contemporary times are marching towards machine intelligence. Machine Learning has become necessary in every sector as a way of making machines works beyond their capabilities. In short, Machine Learning is set of algorithms that parse data, learn from them, and then apply what they've learned to make intelligent decisions.

Examples of Machine Learning are everywhere. It's how Hulu knows which show you'll want to watch or how Instagram recognizes face in a digital photo. Or then again how a client benefit agent will know whether you'll be happy with their help before you even take a CSAT study.

The thing about conventional Machine Learning algorithms is that as mind boggling as they may appear, regardless they're machine like. They require part of space skill, human mediation just equipped for what they're intended for; not all that much, not all that much. For AI originators and whatever remains of the world, that is the place deep learning holds more guarantee.

Deep Learning is a subset of Machine Learning that accomplishes extraordinary power and adaptability by learning to speak to the world as settled pecking order of ideas, with every idea characterized in connection to less complex ideas, and more unique portrayals figured as far as less conceptualones.

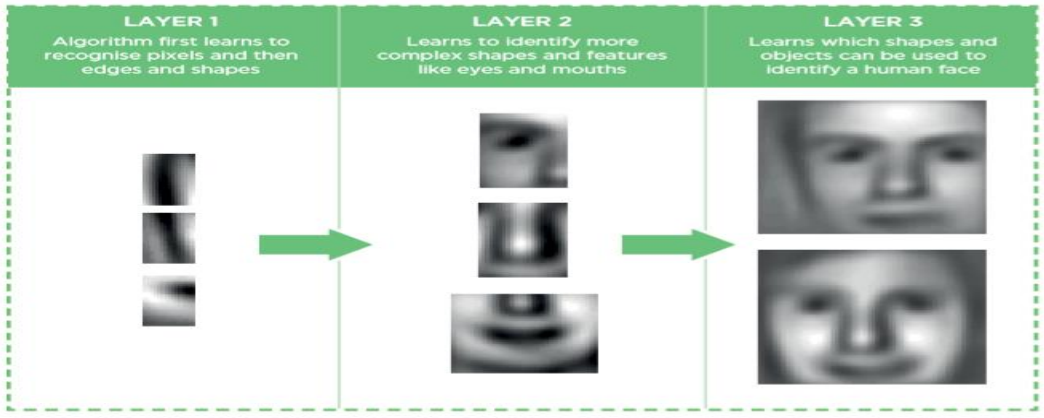

LAYER 1
Algorithm first learns to recognise pixels and then edges and shapes
LAYER 2
Learns to identify more complex shapes and features like eyes and mouths
LAYER 3
Learns which shapes and objects can be used to identify a human face

Fig. 1 - Layers of Deep Learning

1.2 Problem Statement

As earlier stated , Image recognition has many different applications from military to image captcha for security and these application will continue to flourish as classification of images is becoming a necessity for a organization. Usage of image classifiers instead of manually classifying is almost similar when the images to be classified is very meager in number but when we talk about classifying large datasets of images, manually classification becomes very tedious and the work becomes next to impossible. Also for manually classification of images, complete knowledge of class sets and its nuances is compulsory otherwise prediction can be incorrect. With the emergence of following problems in manual classification led to the rise of image classification application. There are various machine learning models that can be used for image classification and all of them has their respective disadvantages. In traditional Machine learning techniques, most of the applied features need to be identified by an domain expert in order to reduce the complexity of the data and make patterns more visible to learning algorithms to work. The biggest advantage Deep Learning algorithms as discussed before are that they try to learn high-level features from data in an incremental manner. This eliminates the need of domain expertise and hard core feature extraction.

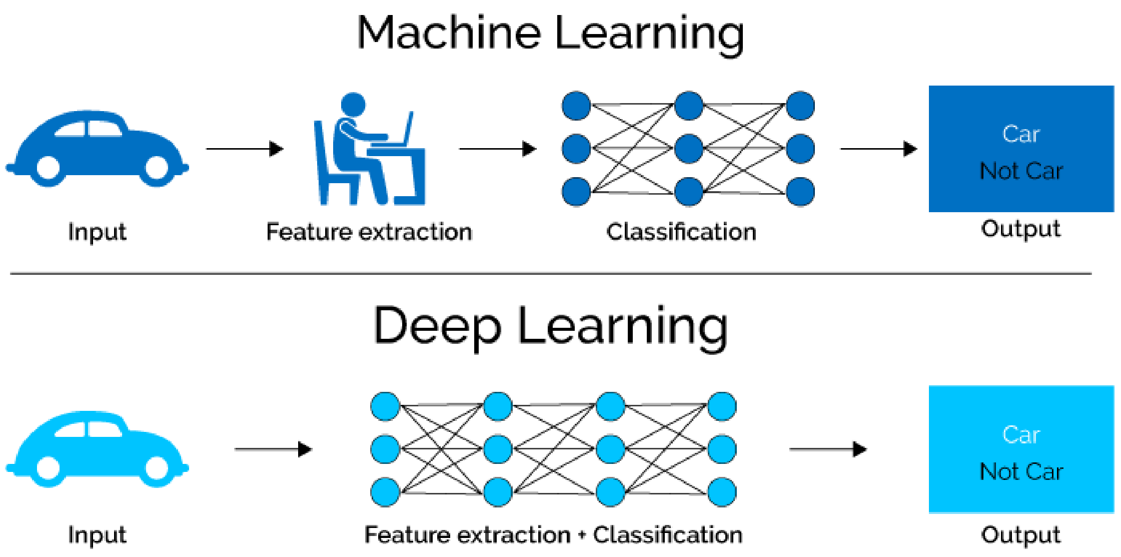


Fig 2. Deep Learning vs Machine Learning

Also deep learning provides an higher accuracy in the results as it got trained large number of data sets and provide better efficiency when amount of data is large.

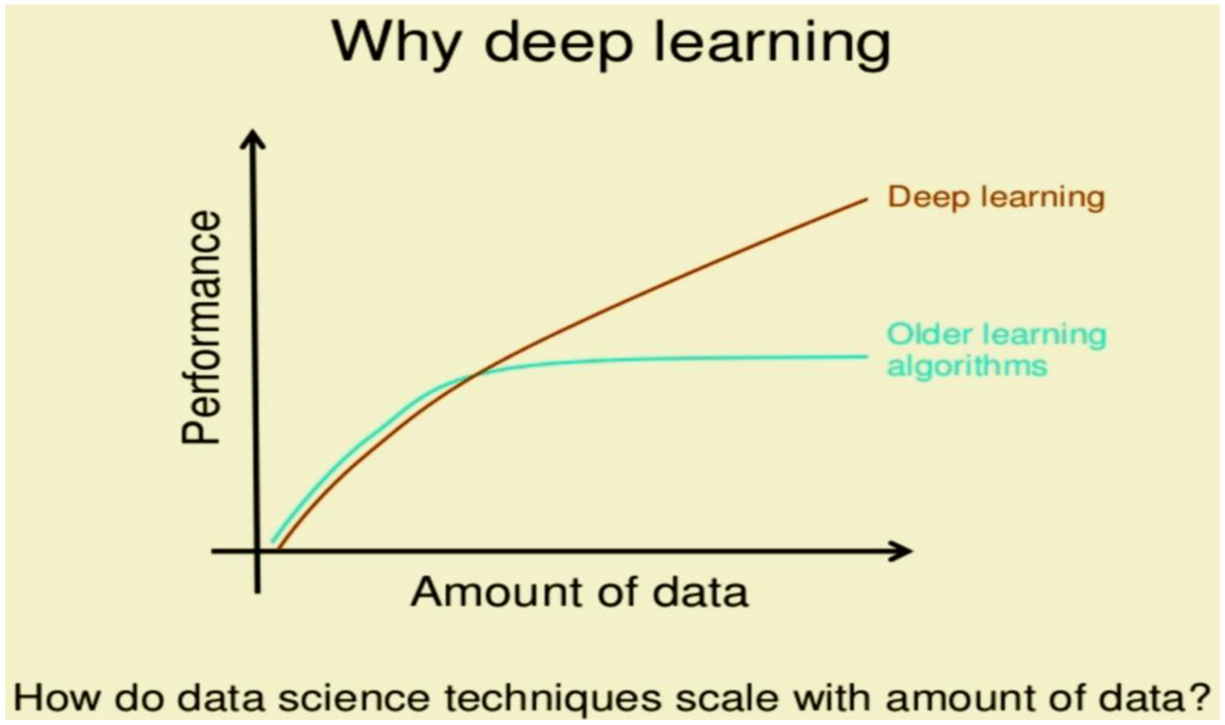


Fig - 3 advantages of deep learning

1.3 Objectives

The main objective of this project is to implement a web application/mobile application of a image classifier, and that classifier will classify images into their respective labels; also the classifier is trained on a large dataset.

1.4 **Methodology**

First of all, imports all the required packages and sub packages with suitable epithats. Load the required data in 3 parts:

- Training: for training, transformation like scaling, rotation and cropping etc. are to be performed as it will lead to better performance. Input data should be resized to a particular length and width as required by pre-trained networks(generally 224x224).

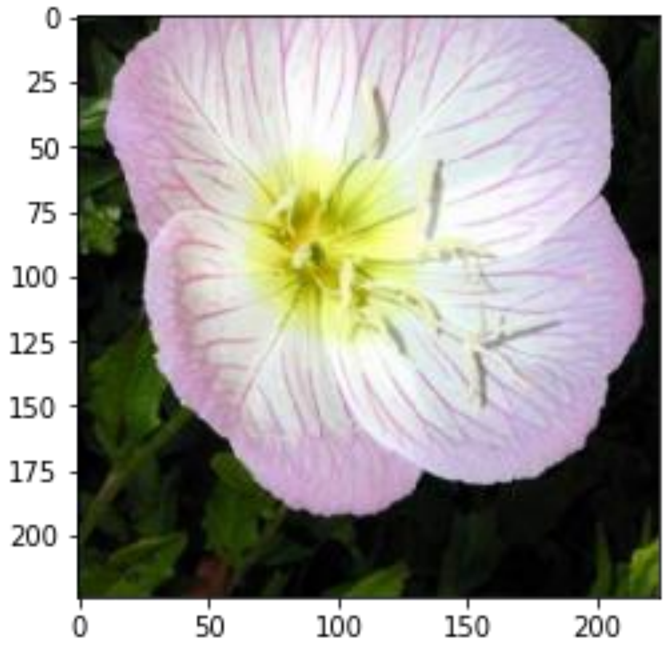


Fig. 4 - 224x224 size input image

- Validation and testing: these sets can be used to measure the performance of model on unseen data. resizing and cropping is required for this data.

ImageNet is pre-trained dataset

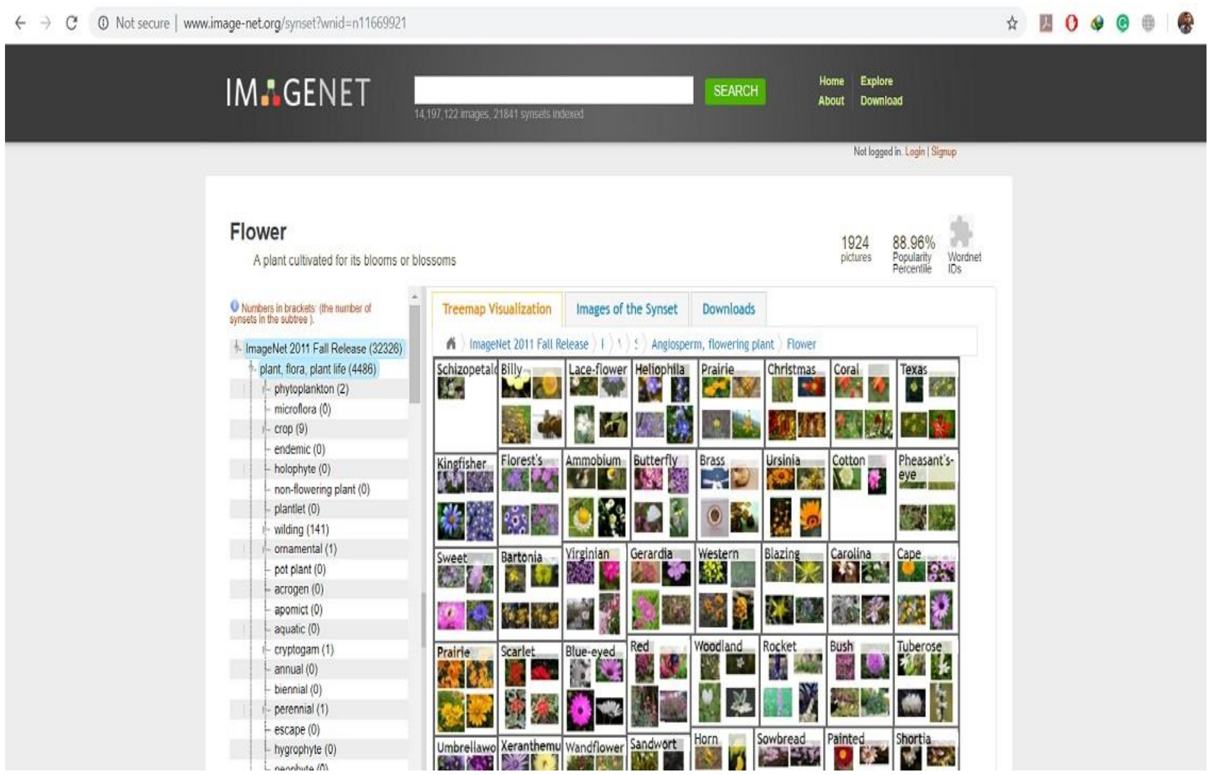


Fig. 5 - Imagenet

By using a .json file the data can be sorted into folders with numbers and those numbers will correspond to specific names specified in the .json file.

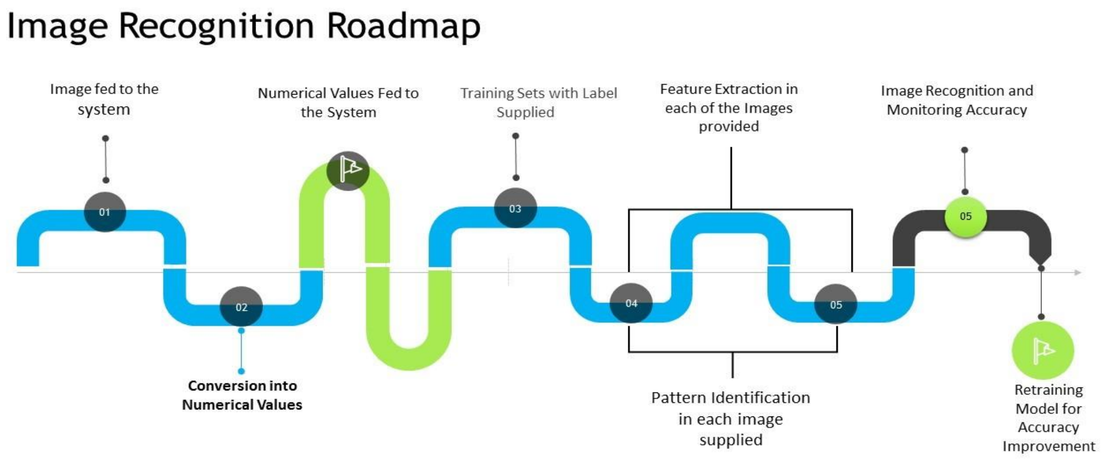


Image Recognition Roadmap
Image fed to the system
Numerical Values Fed to the System
Training Sets with Label Supplied
Feature Extraction in each of the Images provided
Image Recognition and Monitoring Accuracy
01
02
03
04
05
05
Conversion into Numerical Values
Pattern Identification in each image supplied
Retraining Model for Accuracy Improvement

Fig. 6 - Path of classification

Training and building the classifier:

We will use pre-trained model from torchvision.models to get features of image.

(for simple representation VGG can be used)

- Load a pre-trained network.
- Make a new, untrained network as a classifier
- use backpropagation to train the classifier layers using the pre-trained network to get features.
- track accuracy and loss to determine the best hyperparameters and maintain statistics.

Prediction of classes:

Now, probabilities of the different is calculated to find the class of the image.

Number of classes can use k-top function.

Where k is an integer, 5-top defines top 5 classes and 4-top defines 4 top classes which is similar to the image.

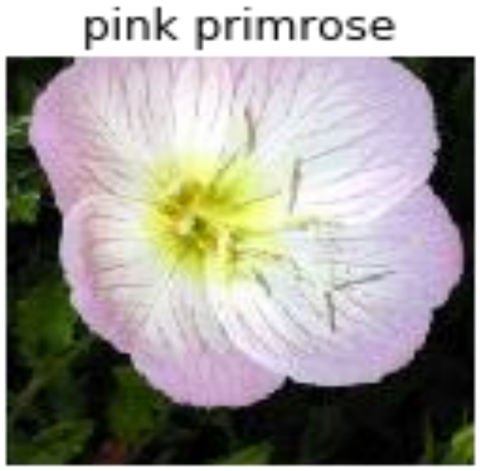


Fig. 7 - Output image

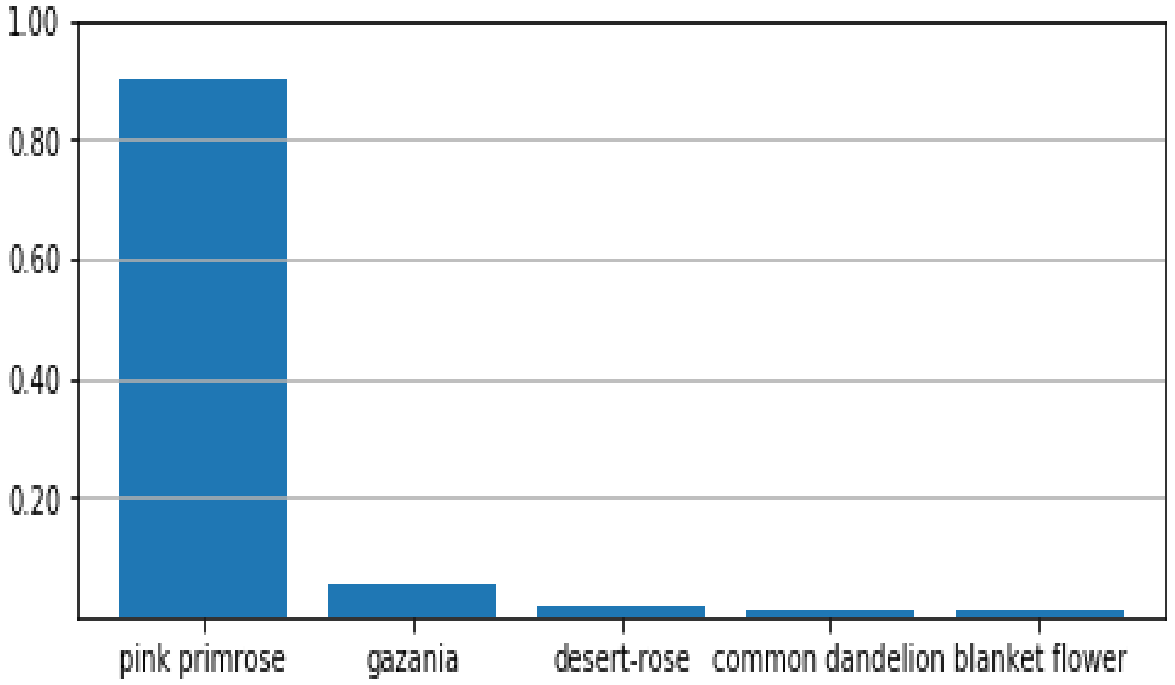


Fig. 8 - 5-top classes of output

1.5 **Organization**

The project is broken down into multiple steps:

1. preprocess and load the image dataset
2. On the dataset, train the classifier
3. predict image content using classifier trained in 2.
4. Analyze the results provided by architecture

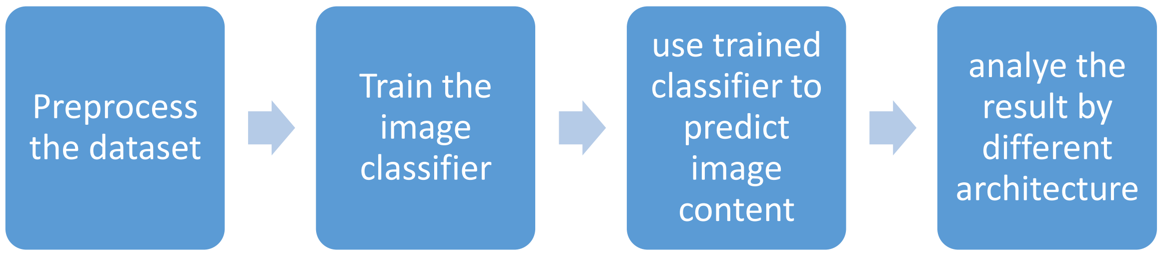

Preprocess the dataset
Train the image classifier
use trained classifier to predict image content
analye the result by different architecture

Fig. 9 - Organization of project

CHAPTER 2: LITERATURE SURVEY

2.1 Books and Publications

To arrive at a conclusion to use the particular type of classification technique in the project required a arduous study of all the probable techniques with their advantages and disadvantages along with their motivation.

As to classify any image various steps is involved:

- Define Classification Classes
  Classification classes depends on property and objective of image.
- Feature Selection
  Multi-spectral and multi-temporal properties is used to correct the differences between the classes. Features of classes varies.
- Training data' sampling
  It is necessary to sample the training data. Supervised classification, Unsupervised learning, and Semi supervised learning will be used according to the data.
- Estimate the Universal Statistics
  Appropriate method will be selected with the help of compare and contrast.
- Method of Classification
  Some methods are- SVM, Linear, ANN, Fuzzy Tree.

The books that shed lights on this topic is:

Image Recognition and Classification: Algorithms, Systems, and Applications (Optical Science and Engineering); 1st edition; by Bahram Javidi.

Digital Image Processing 3rd Edition by Richard E Woods and Rafael C Gonzalez

Machine Learning for Image Classification by Yu-Jin Zhang (Department of Electronic Engineering, Tsinghua University, China)

Also, a IEEE paper also covered this topic " we review the current activity of image classification methodologies and techniques. Image classification is a complex process which depends upon various factors. Here, we discuss about the current techniques, problems as well as prospects of image classification. The main focus will be on advanced classification techniques which are used for improving classification accuracy."

2.2 **Talks, Conferences and Meetups**

International Conference on Signal Image Technology and Internet Based Systems addressed this issue in their 14th International conference held on 26 Nov 2018 - 29 Nov 2018 in Las Palmas de Gran Canaria, Spain.

ICCVBIC-2018 — International Conference On Computational Vision and Bio Inspired Computing's main aim is to give a universal discussion to the trading of thoughts among intrigued scientists, understudies, engineers, and specialists in the zones of Computational Vision and Bio Inspired Computing. It is being composed on 29-30 November, 2018 by the Inventive Research Organization. ICCVBIC will give an outstanding international discussion to sharing learning and results in all fields of science, engineering and Technology. ICCVBIC gives quality key specialists who give an open door in bringing up innovative thoughts. Ongoing updates in the in the field of innovation will be a stage for the upcoming researchers.
IEEE's CVPR 2019 : IEEE Conference on Computer Vision and Pattern Recognition, CVPR addressing the issues such as 3D from Multiview and Sensors, 3D from Single Image, Big Data, Large Scale Methods, Computer Vision Theory, Datasets and Evaluation, Deep Learning Techniques, Document Analysis, RGBD sensors and analytics, Face, Gesture, and Body Pose, Image and Video Synthesis, Low-level Vision,

Machine Learning, General, Medical, Biological and Cell Microscopy, Motion and Tracking, Optimization Methods

CHAPTER 3: SYSTEM DEVELOPMENT

3.1 **System Architecture**

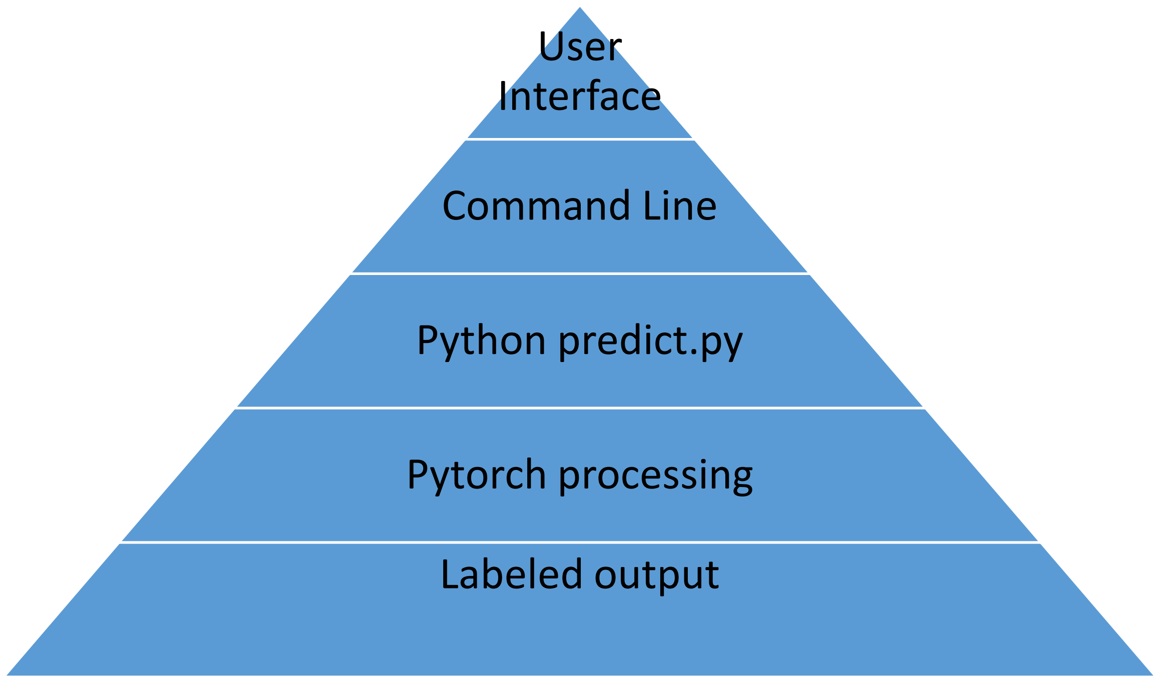


Fig. 10- The layers of the application

3.2 Perceptron-

A perceptron which looks like a neuron in the body is the building block of the Artificial Intelligence. Perceptron takes inputs (numbers of inputs are not limited by number) and these inputs are multiplied by some weights and these combination of inputs and weights are then processed by the perceptron and output in the terms of YES or NO is provided.

for eg- Suppose a perceptron model predicts whether a student will get admission in a university based on his test score and grade score.

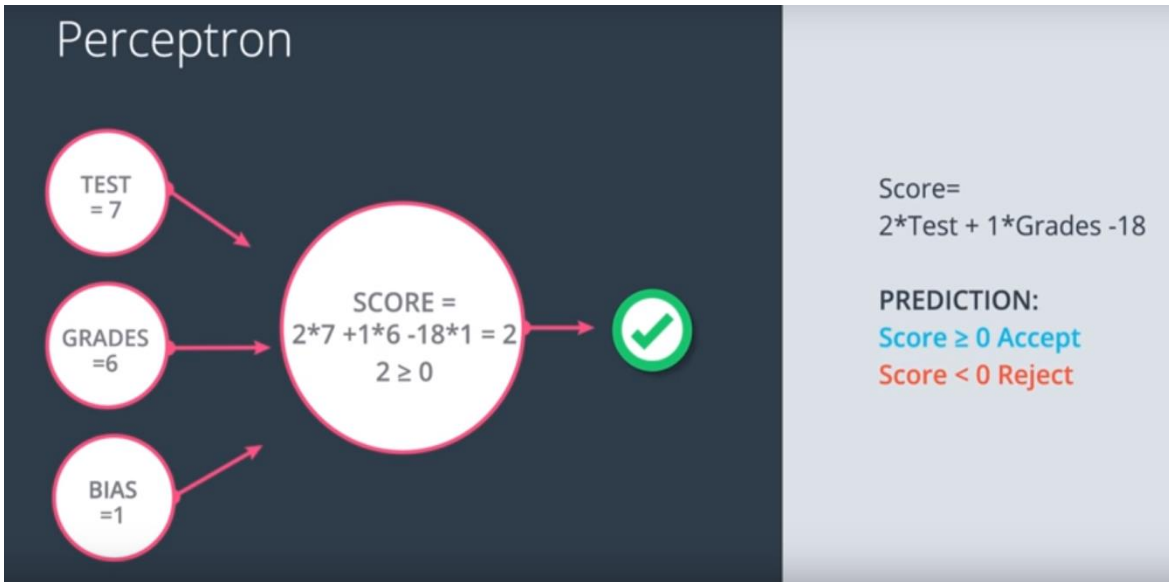


Fig 11: Perceptron

In this example, inputs are the test and grade scores of the candidate, and these inputs are fed to the perceptron and according to the previous data a linear equation is formed by the perceptron which will judge the score of the candidate. In this figure score equal to or above 0 means Acceptance from the university and score below 0 means Denial from the university.

As you can see the linear equation will decide the outcome of the perceptron.
So, there is very much probability that this linear equation may predict the output wrong. To avoid this wrong output the equation needs to be altered with the help of weights.
There are 2 types of wrong prediction:-
1. If the point is classified positive, but it has a negative label.
2. If the point is classified negative, but it has a positive label.
And these 2 types of error has different solution.

ALGORITHM FOR PERCEPTRON:

```
For every misclassified points(x1,x2.....xn):
    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 Prediction and loss function:

Till now we are using a step function in our perceptron which gives us only 2 outputs but we know that in real life we need a whole lot of outputs so we will use a sigmoid function which gives a continuous value ranging between 0 and 1.

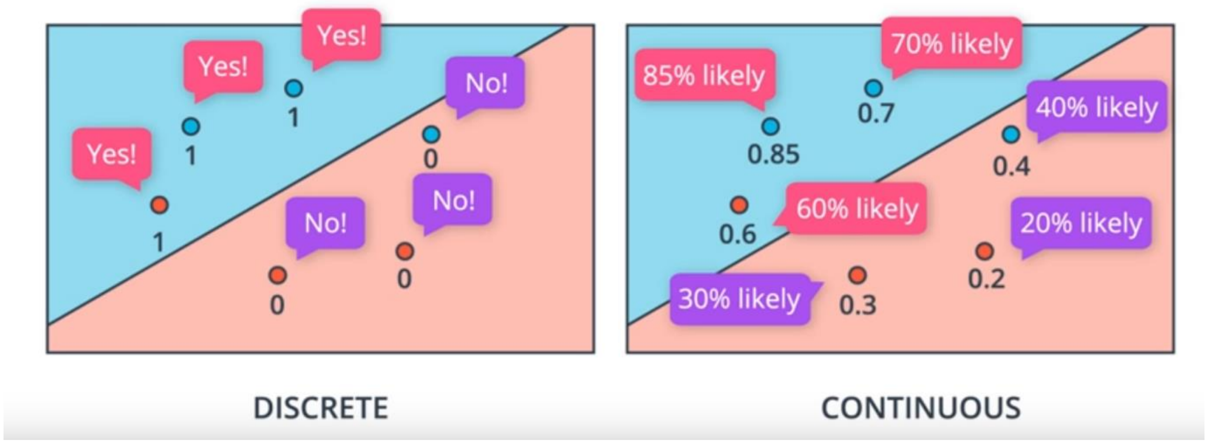


Fig 12: Discrete and continuous prediction

The points on the line has the value 0.5 and as the points move away from the line then the value for points vary as it increases is the point is above the line and decreases as the point is below the line.

For eg. In the figure above, those points above the line has the probability 0.6 , 0.7 and 0.85.

Till now, we have only 2 output classes but what if we needs to classify the points into 3 or more classes? We will use softmax function.

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- This is the product of the probabilities of their respective color as perdicted by the perceptron. for example:

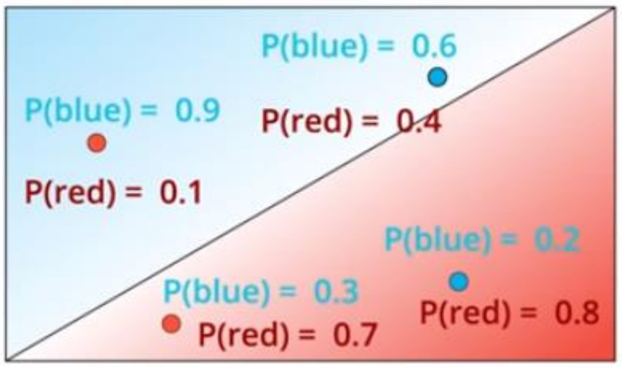


Fig 13: Maximum Likelihood Probability

MLP= 0.1*0.6*07*0.2=0.0084

If the number of points is very large then the product of probability can be very small and will be of no significance , to avoid this situation we need to sum the individual probabilities, this give rise to the concept of cross entropy.

In CROSS ENTROPY, we take the log of all the probability and adds them.
In the example above, cross entropy can be calculated by:
-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
Less effective model	less	more

Table 1: Effective model in terms of probability and loss

3.4 Gradient descent and logistic regression algorithm:

The basic algorithm for logistic regression is:

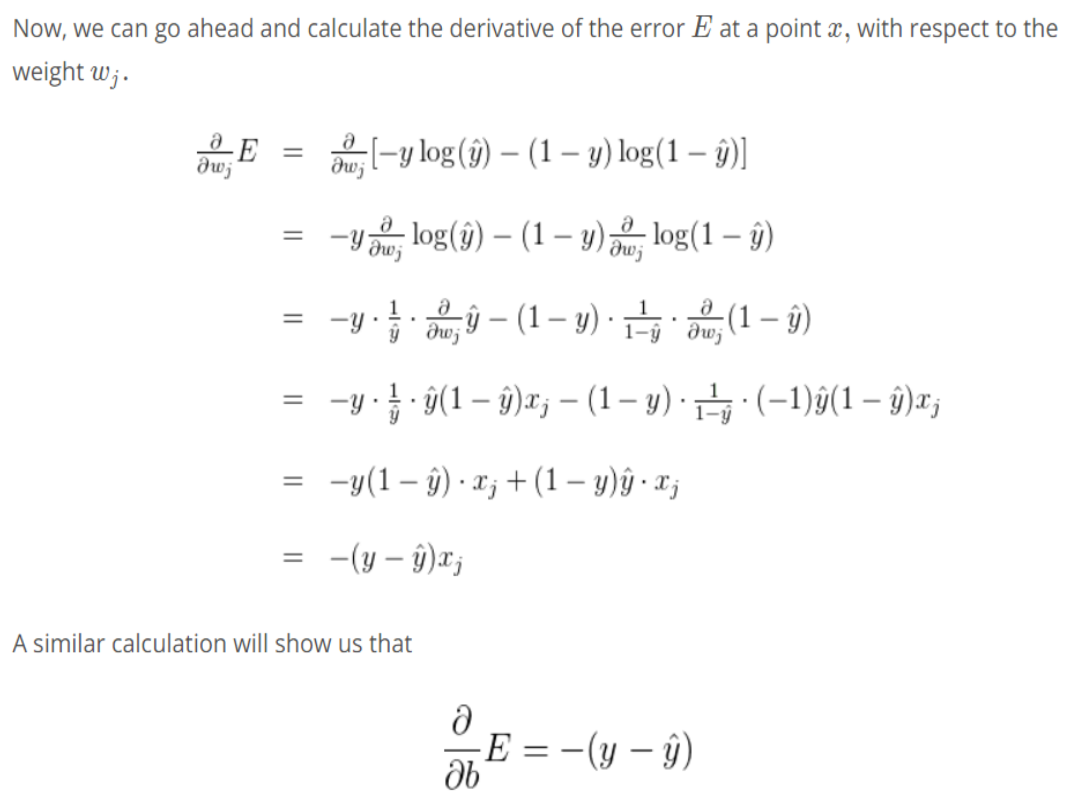
1. Take the data.
2. Pick a random model.
3. Calculate the error.
4. Minimize the error.

We can define a error function with the help of cross entropy loss such as

Error Function= - 1/m ∑(1-y)*(ln(1-ycap))+y*ln(ycap)

where ycap is the predicted output and y is label of the point.

To minimize the error we have to use the concept of gradient descent. Gradient descent is an algorithm used to lower the output of some function by iteratively moving in the direction of steepest descent as defined by the negative direction of the gradient. In machine learning, we use gradient descent to update the parameters of our model. Think as you are standing on top of mountain and you want to come down. There are various options and directions for you to come down but you chooses the steepest step and comes to a position lower than the earlier position but still very much on top. Now you repeat the steps of steepest descent until you reach to the bottom , the same applies to the process of minimization of loss you calculate the gradient of loss with respect to weights and bias and updates the respective weights and bias to get the better model.



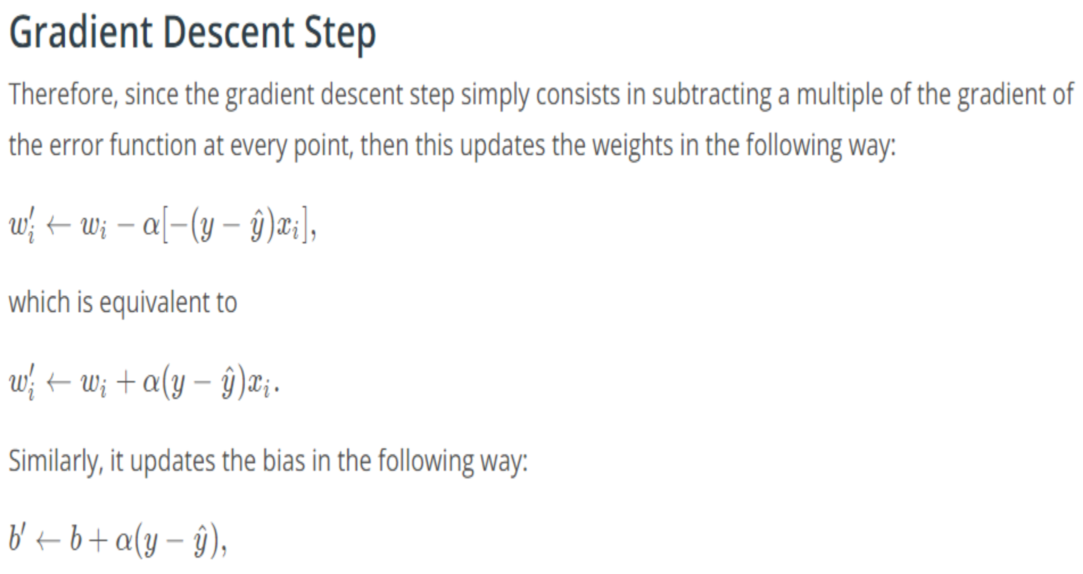
Now, we can go ahead and calculate the derivative of the error E at a point x, with respect to the weight w_j.

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

A similar calculation will show us that

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

Fig 14: Gradient descent wrt weight and bias



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 15: weight manipulation wrt gradient

3.5 Non-Linear Models

Till now we have only talked about keeping in mind a linear model but Neural Networks are used to accomplish a more complex task and these task are generally non-linear. To achieve a non-linear model ironically a linear model is used.

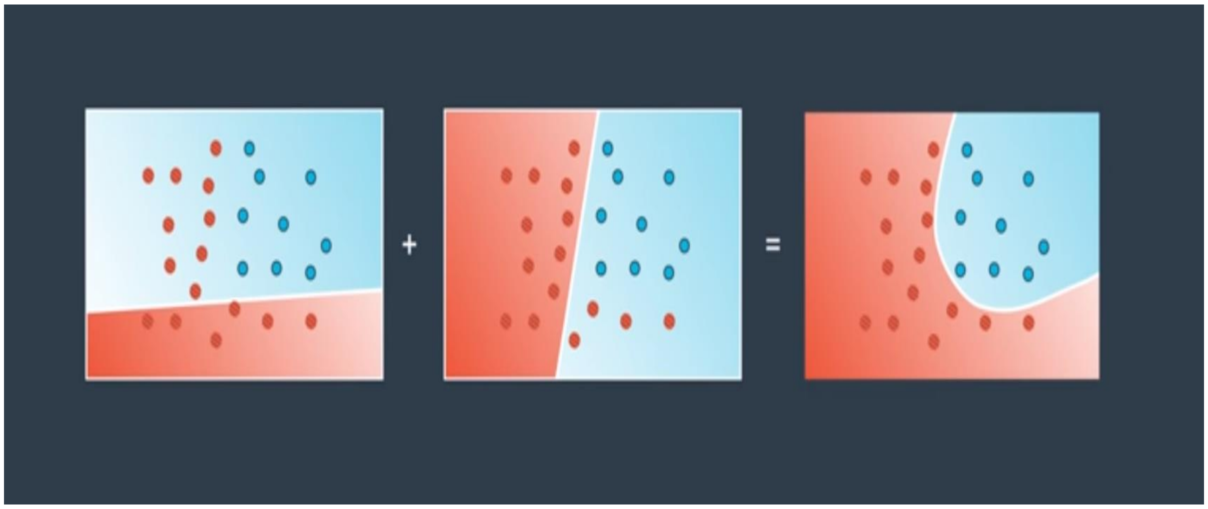


Fig 16: Non-Linear models

In the above figure, first linear model classifies the blue and red points but not very efficiently and also in the second linear model, the model classifies the red and blue points better than first linear model and if we add both the model than we get a non linear model with a very high efficiency.

Also the probability of a point in both the model is added and then passes into sigmoid function which gives us the probability of the point in non linear model.

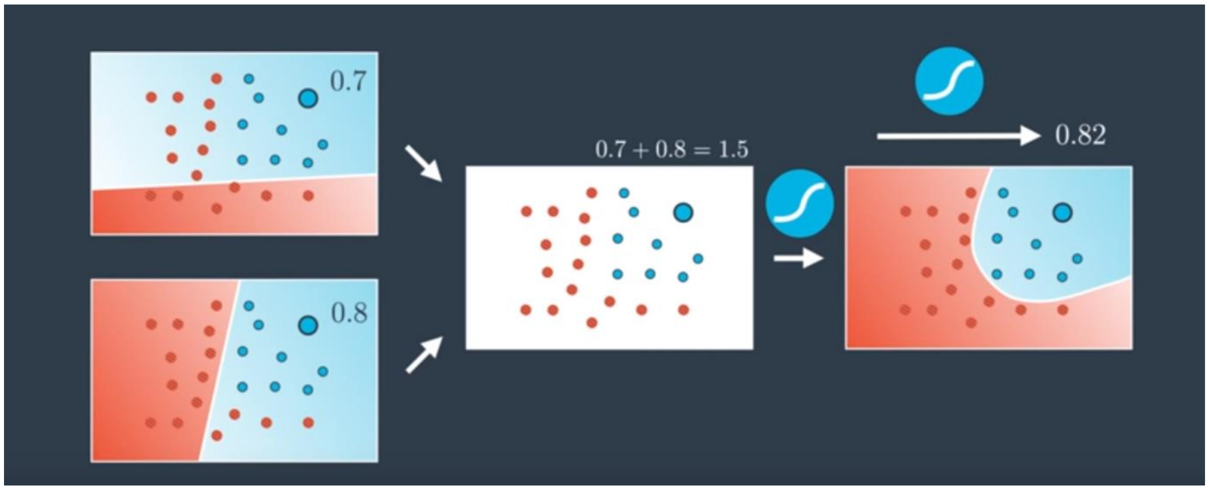


Fig 17: Probability in non linear models

for example in this figure the probability of the point is 0.7 in first linear model and 0.8 in second linear model and we have added both probability to 1.5 and then sigmoid function is applied which gives us the probability of the same point in non-linear model equals to 0.82.

We can also add weights and bias to the models like in the above diagram we can add a weight of 7 to first model and a weight of 5 to second model and the output is calculated accordingly.

7*(0.7)+5*(0.8)-6=2.9 ->Sigmoid-> 0.95

We can define the concepts of non-linear model and weights to linear model as:

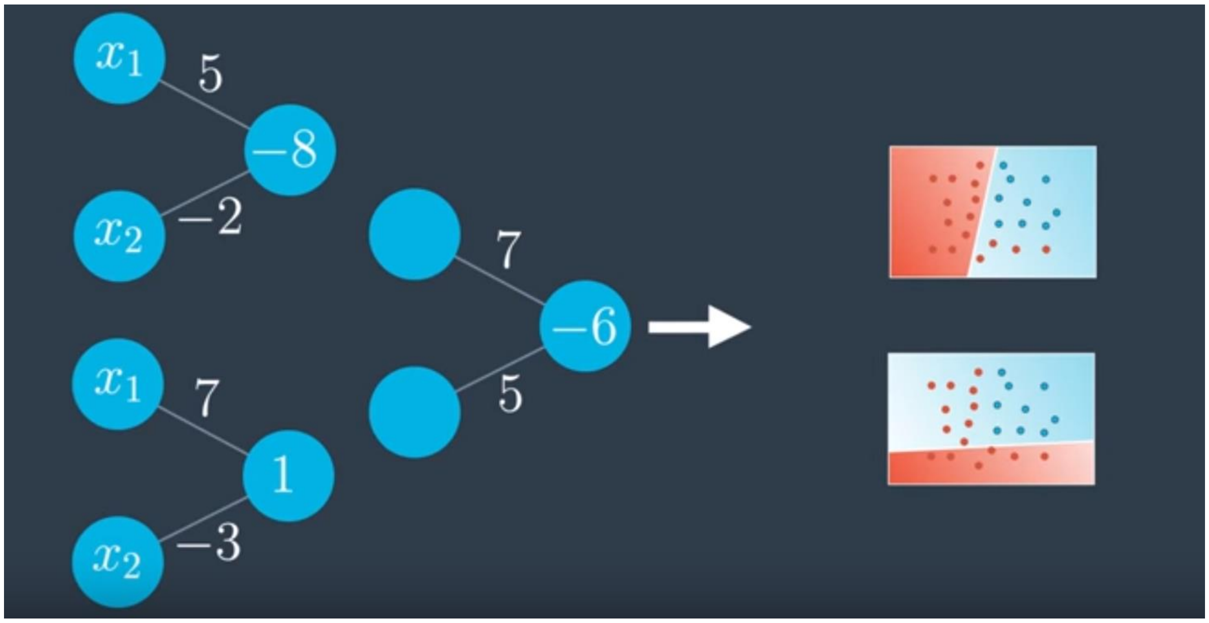


Fig 18: weights to linear models

All these concepts are a rudimentary and base concepts of Deep Neural Networks.

3.6 Neural Networks:

Using all these concepts of perceptrons and non-linear models we can define a neural network. Neural Networks comprises of 3 Layers:

1. INPUT LAYER
2. HIDDEN LAYER
3. OUTPUT LAYER

Input layer contains the input provided by the model which in turn are multiplied by the weights and this amalgam becomes the input for Hidden layers. There can be many hidden layers of different shapes. These hidden layers are used to form very complex models. The output of hidden layers becomes the input for output layer. The number of outputs in output layers is generally equals to the classes of input data.

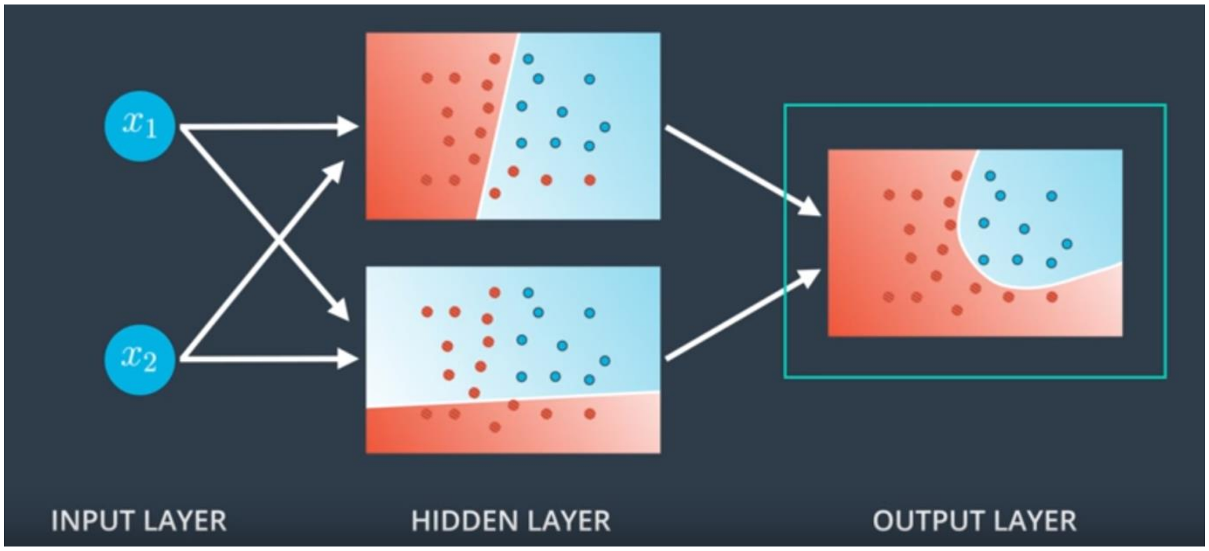


Fig 19: structure of Neural Network

3.7 Training a Neural Network:

A neural network needs to be trained before correctly classifying images. Training a Neural networks typically consist of 2 phases:

1. Feed-Forwarding
2. Backpropagation

1. Feed-Forwarding- The reason these networks are called feedforward is that the flow of information takes place in the forward direction, as x is used to calculate some intermediate function in the hidden layer which in turn is used to calculate y.

These networks are represented by a composition of many different functions. Each model is associated with a graph describing how the functions are composed together.

Basically, Feedforward is the process neural networks use to turn the input into an output. Inputs are first multiplied with their respective weights and then activation function is applied to these dot product. The output of the activation function becomes the input for hidden layer and then these input are again dot producted with their respective weights and hence the output becomes input for hidden layer if any and if not then these output goes straight to output layer.

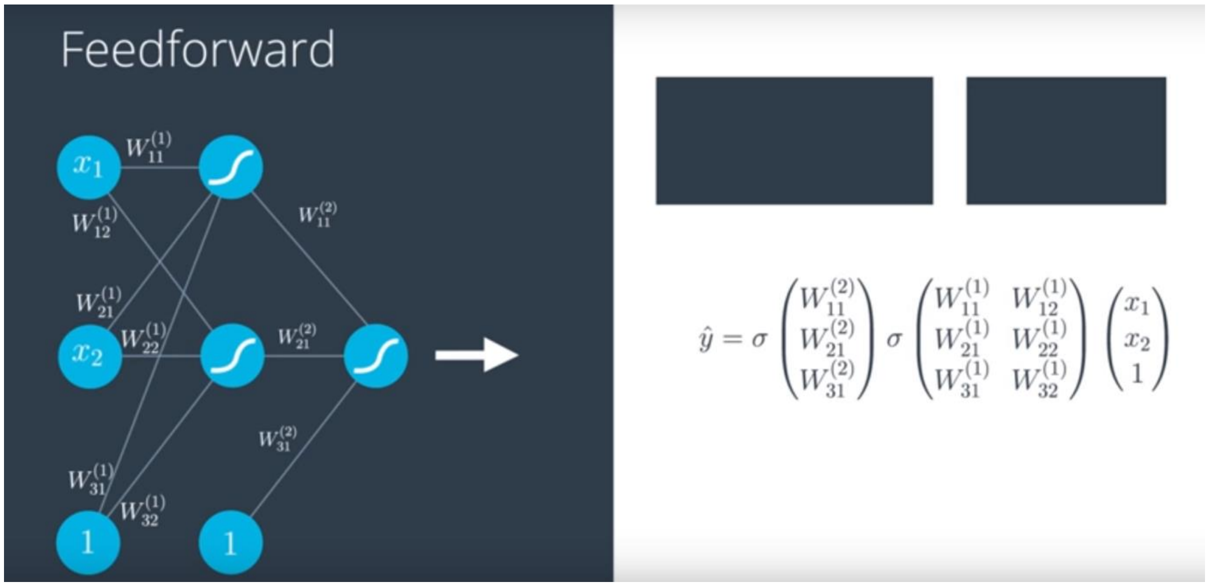
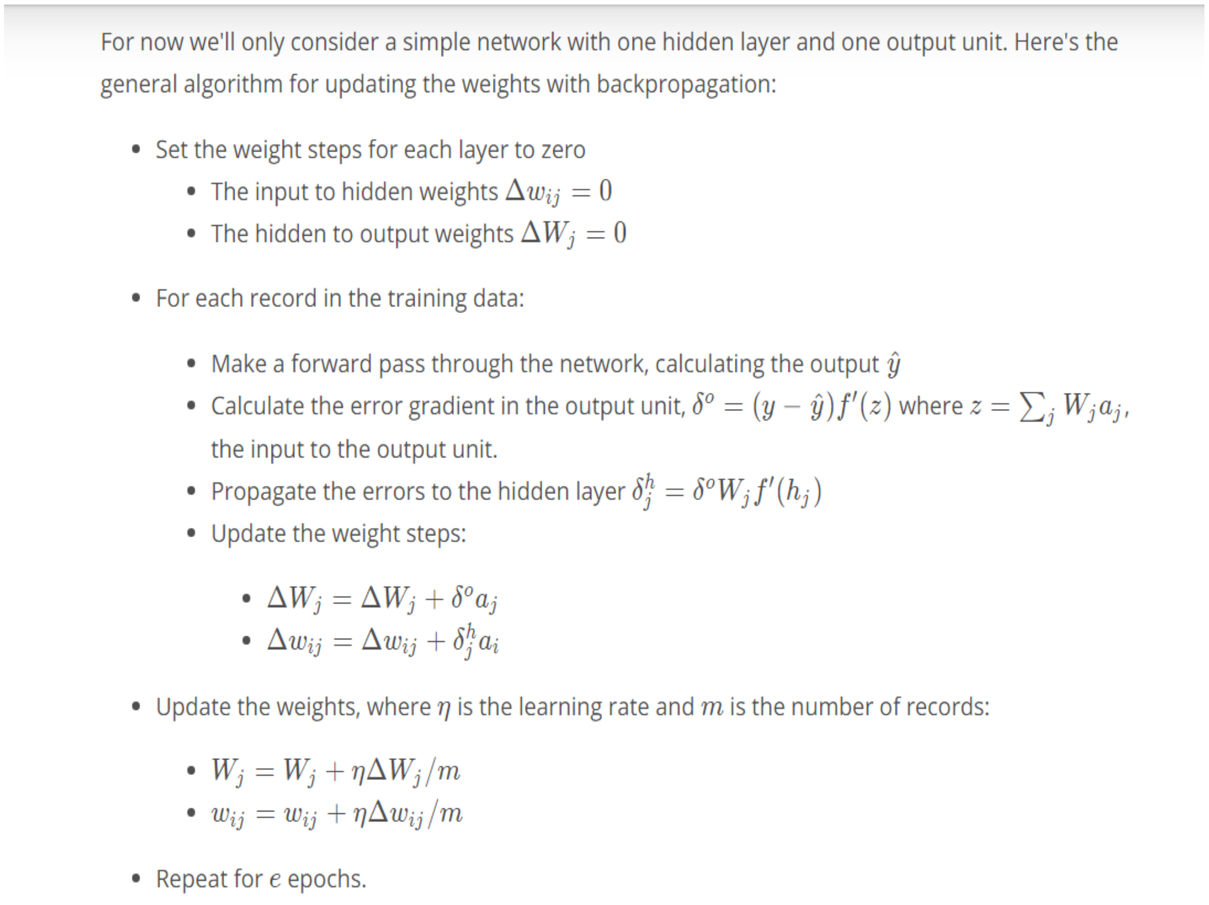

Feedforward
x_1
x_2
1
1
$W_{11}^{(1)}$
$W_{12}^{(1)}$
$W_{21}^{(1)}$
$W_{22}^{(1)}$
$W_{31}^{(1)}$
$W_{32}^{(1)}$
$W_{11}^{(2)}$
$W_{21}^{(2)}$
$W_{31}^{(2)}$
$\hat{y} = \sigma \begin{pmatrix} W_{11}^{(2)} \\ W_{21}^{(2)} \\ W_{31}^{(2)} \end{pmatrix} \sigma \begin{pmatrix} W_{11}^{(1)} & W_{12}^{(1)} \\ W_{21}^{(1)} & W_{22}^{(1)} \\ W_{31}^{(1)} & W_{32}^{(1)} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ 1 \end{pmatrix}$

Fig 20: Feed forward process

2. Backpropagation - Backpropagation algorithms are methods used to train neural networks following a gradient descent approach that uses the chain rule. The main feature of backpropagation is its iterative, recursive and efficient way of calculating the weights updates which helps to improve the network until it is able to perform the task for which it is being designed.

In backpropagation, error are calculated by comparing the output of feed-forwarding and labels. Gradient of the error with respect to the weights are then calculated. Gradient with respect to the weight is itself a chain derivative as error is connected to output , output is connected to activation function and activation function is connected to the weights. This gradient is used to minimize the error by updating the weights of the network. This process of feed forwarding and backpropagation is run several times. To avoid any drastic weights changes , we use learnign rate so that weights do not change on extremes and integrity as well as efficiency of the model is maintained.

ALGORITHM for back propagation:


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 η 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 21: Algorithm for back propagation

3.8 Measures to increase effieciency:

Early stopping: When we train our data a large number of times, the loss begins to decrease but when we try to run validate data on trained model, loss decreases up to some extent and then it begins to increase. This is because we have trained data with a large number of epochs and model has become a overfitted model which will not perform efficiently on validation data.

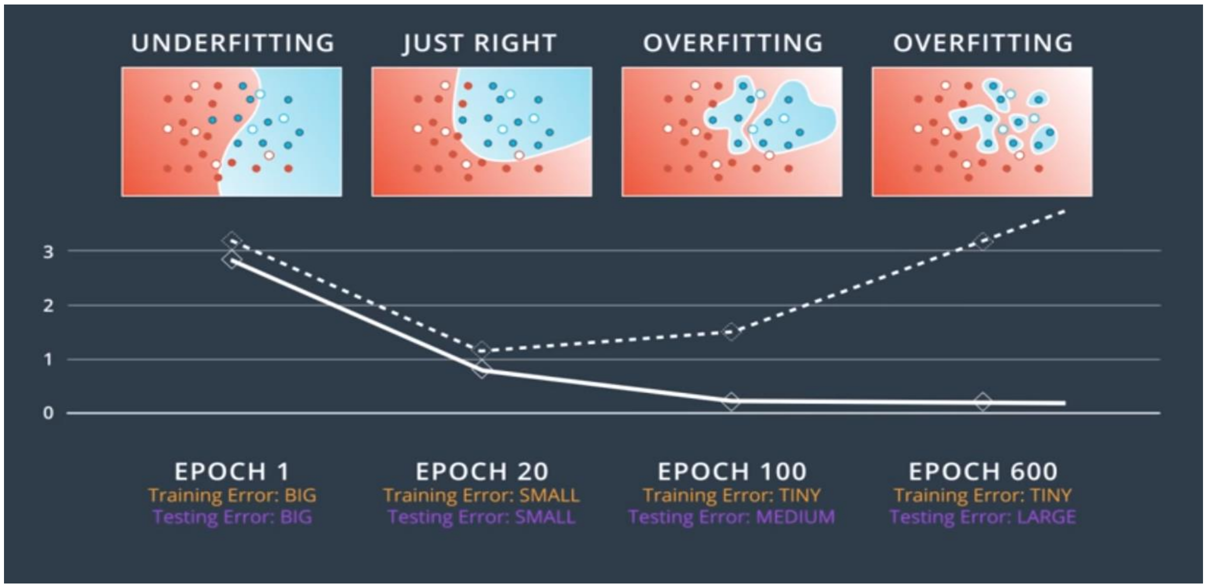


Fig 22: Over-fitting

To avoid over fitting we need to stop our epochs count where loss for valid data starts increasing. This is called early stopping.

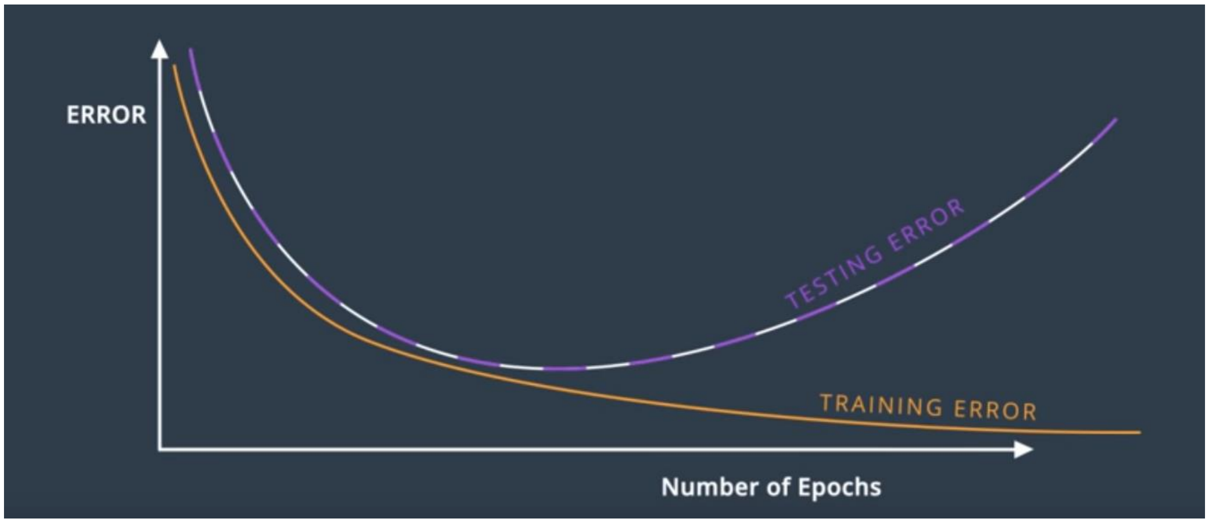


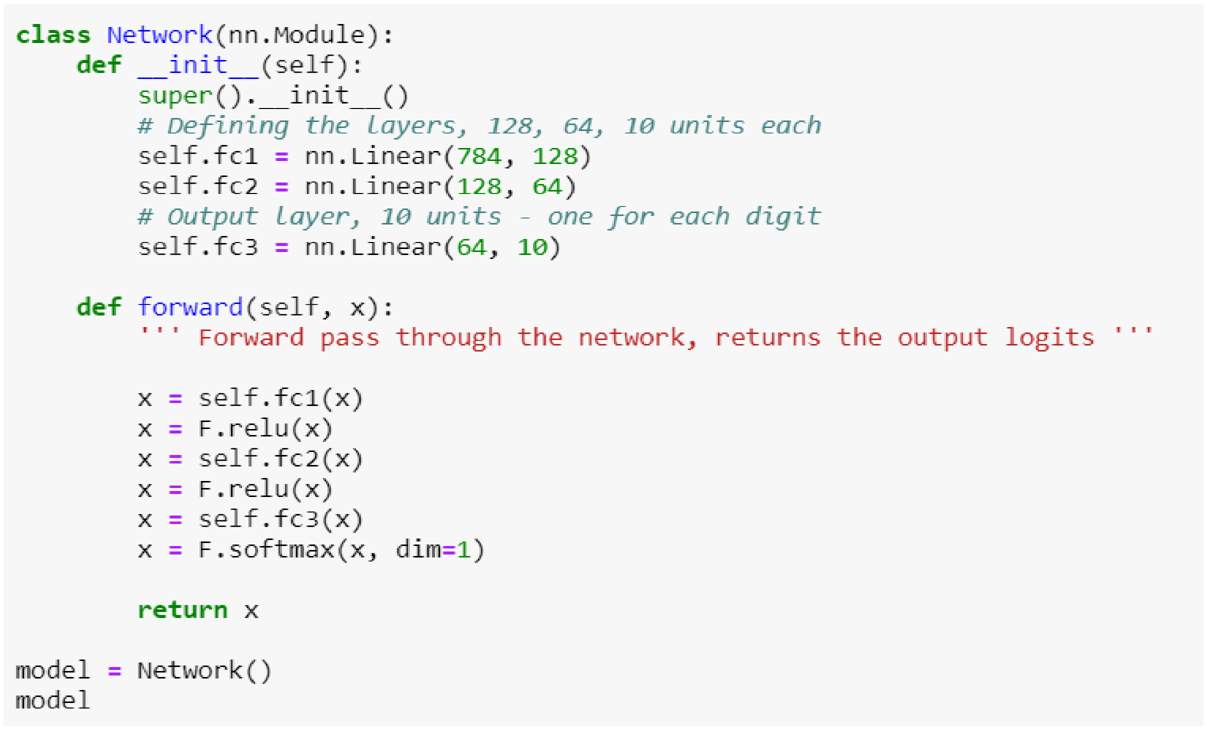
Fig 23: Early-stopping

Dropout: In dropout, random nodes in each epochs are freezed and not allowed to train so that other nodes takes more part in the process and becomes strong, The process of selecting the nodes for freezing is completely random and is completed by adjusting a probability like 0.2,0.3.

Local Minima: While calculating the gradient descent, you will always not able to set a global minima , you may someday encounter a local minima. To avoid getting a local minima, you can restart getting gradient descent from random places.

3.9 Neural Networks using Pytorch

Building network: Using pytorch, we can design our own network which comprises of the number of inputs, hidden layers, number of hidden layers, output layers, and activation functions.

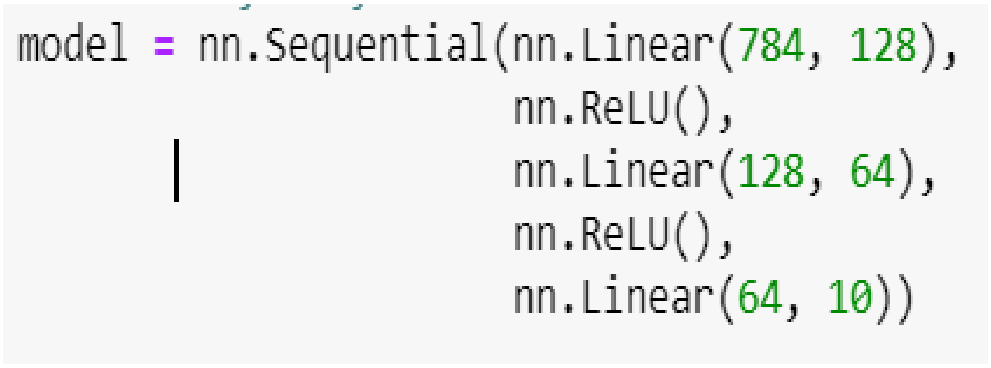


```python
class Network(nn.Module):
    def __init__(self):
        super().__init__()
        # Defining the layers, 128, 64, 10 units each
        self.fc1 = nn.Linear(784, 128)
        self.fc2 = nn.Linear(128, 64)
        # Output layer, 10 units - one for each digit
        self.fc3 = nn.Linear(64, 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 24: Classifier model using class

Here, we have defined a class Network which takes nn.Module as argument. Here we have defined 2 hidden layers with inputs 128 and 64 respectively. And after each dot product of weights and inputs, activation function is performed. The output consist of 10 classes and can be obtained using Softmax.

A network can also be created using nn.Sequential in which we don't have to define a class.



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

Fig 25: Classifier model without class

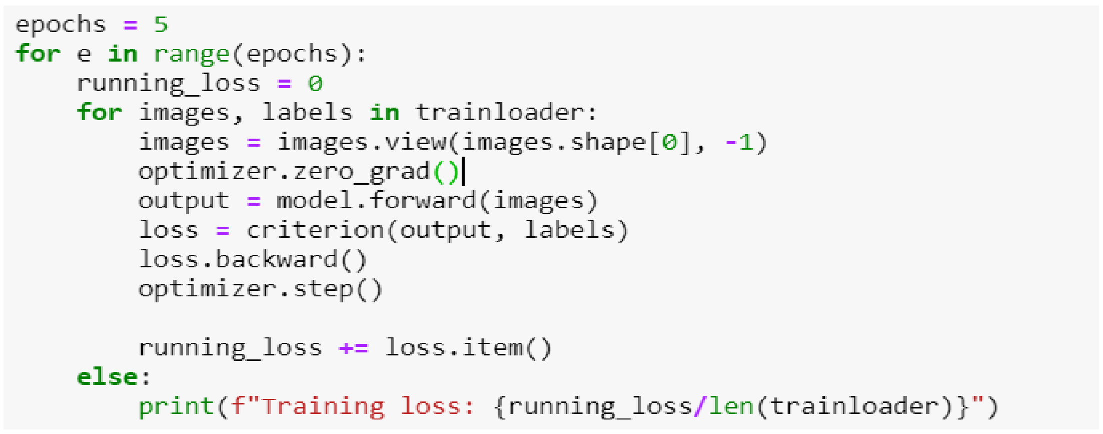
LOSSES in Pytorch:

We can calculate the loss with PyTorch using nn module. PyTorch provides losses for example cross-entropy loss (nn. CrossEntropyLoss).In general, loss is assigned to the criterion. To actually calculate the loss, you first define the criterion then pass in the output of your network and the correct labels.The output of the criterion is the loss for the network.

Autograd:

Torch provides a module called autograd which calculates the gradients of tensors. We use autograd to calculate the gradients of all our weights parameters with respect to the loss. Autograd works by keeping record of operations performed on tensors, then going backwards through those operations, calculating gradients along the path. Set requires_grad = True on a tensor, to make sure PyTorch keeps track of operations on a tensor and calculates the gradients.

Loss and Autograd together:



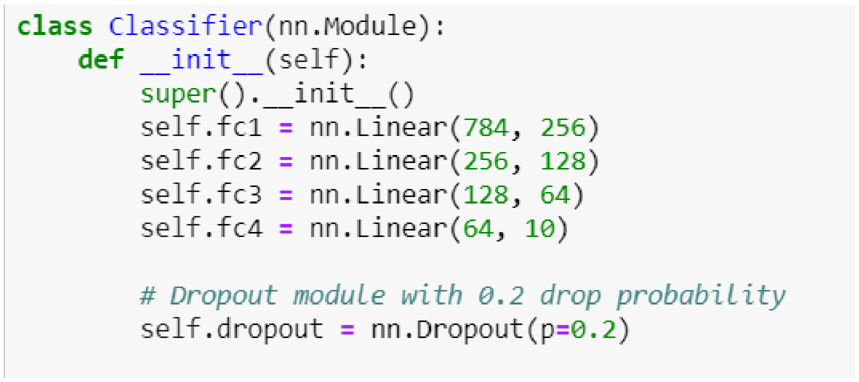
```python
epochs = 5
for e in range(epochs):
    running_loss = 0
    for images, labels in trainloader:
        images = images.view(images.shape[0], -1)
        optimizer.zero_grad()
        output = model.forward(images)
        loss = criterion(output, labels)
        loss.backward()
        optimizer.step()

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

Fig 26: feed forwarding and back propagation in pytorch

Here criterion calculates the loss and loss.backward() calculates the gradient and optimizer.step() updates the weights of the model.

Inference and validation:

To increase the efficiency of the Pytorch model we can use the nn.dropout function whichenables the nodes to learn more accurately.

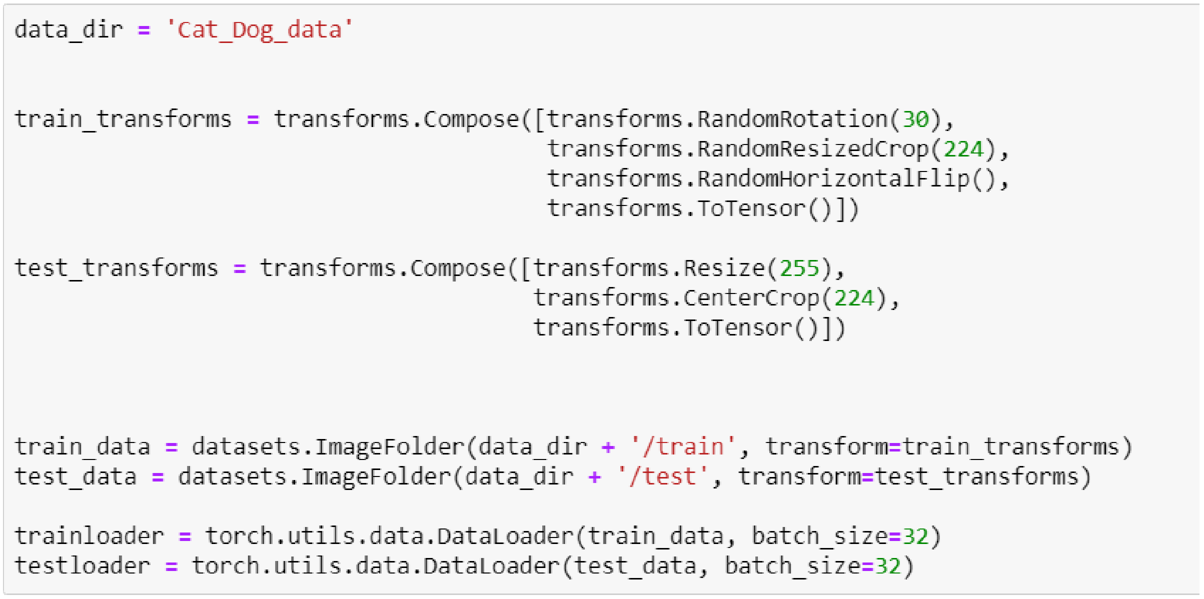


```python
class Classifier(nn.Module):
    def __init__(self):
        super().__init__()
        self.fc1 = nn.Linear(784, 256)
        self.fc2 = nn.Linear(256, 128)
        self.fc3 = nn.Linear(128, 64)
        self.fc4 = nn.Linear(64, 10)

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

Fig 27: dropout in pytorch model

For validation, we checks the model with unseen images and with gradient off because the goal here is to find the accuracy of the model and not training. model.eval() directs the model for evaluation purposes. We will load the images in model in same way as of training and calculate the validation loss. then we will calculate the probability for each output class and find the class with maximum probability and compare it with the label. This way we can calculate the accuracy and validation loss.

Loading 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)
```

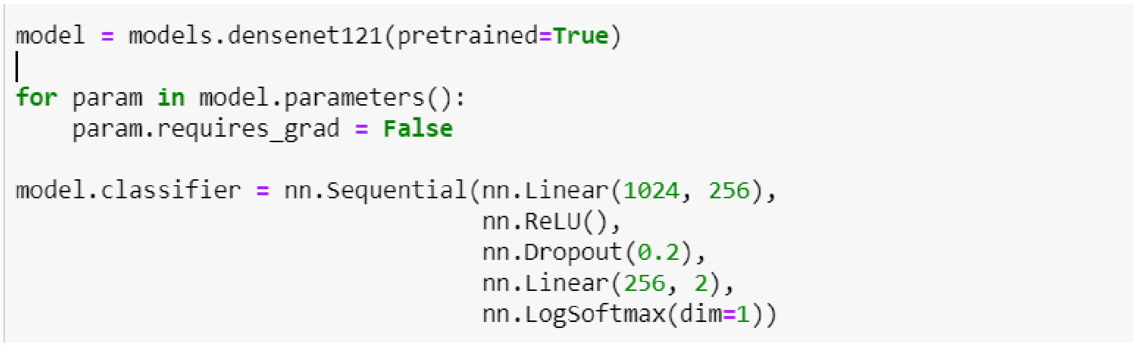
Fig 28: Loading Image datasets

Here, we have loaded train and test data separately using datasets module of torchvision. We have used ImageFolder method to get the directory. After loading the data we need to push the data in Dataloader method of torch.utils.data with a argument of batch size.

Transfer learning:

ImageNet is a massive dataset with over 1 million labeled images in 1000 categories. It's used to train deep neural networks using an architecture called convolutional layers. Once trained, these models work astonishingly well as feature detectors for images they weren't trained on. Using a pre-trained network on images not in the training set is called transfer learning.

We can import the pretrained models using torchvision.models and in the argument we have to provide pretrained=True. When we load a pre trained model we have to freeze its parameters. But loading a pretrained model does not solve our problem. Still we have to define our parameters like input , hidden and output.



```
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))
```

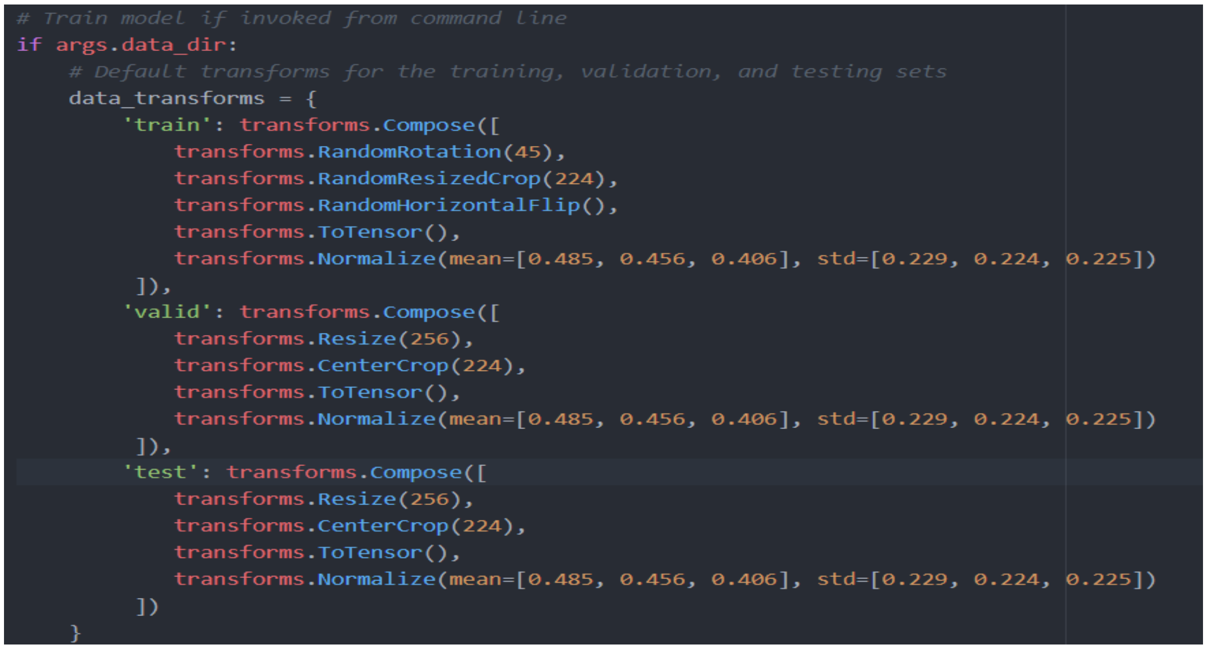
Fig 29: transfer learning

CHAPTER 4: ALGORITHMS

Algorithm 1: **Train model**

```
Define train_model(dataset,architecture,hyperparameters)
    ->Define Dataloaders using imagesets
    ->Calculate dataset sizes
    ->Load the model
    ->device= GPU (if available)
        ->else Use CPU
    ->start time
    ->training
    ->stop time
    ->statistics
```

Also if the arguments are provided through command line then value of data_transforms are pre-defiened.



```python
# Train model if invoked from command line
if args.data_dir:
    # Default transforms for the training, validation, and testing sets
    data_transforms = {
        'train': transforms.Compose([
            transforms.RandomRotation(45),
            transforms.RandomResizedCrop(224),
            transforms.RandomHorizontalFlip(),
            transforms.ToTensor(),
            transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
         ]),
        'valid': transforms.Compose([
            transforms.Resize(256),
            transforms.CenterCrop(224),
            transforms.ToTensor(),
            transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
         ]),
        'test': transforms.Compose([
            transforms.Resize(256),
            transforms.CenterCrop(224),
            transforms.ToTensor(),
            transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
         ])
    }
```

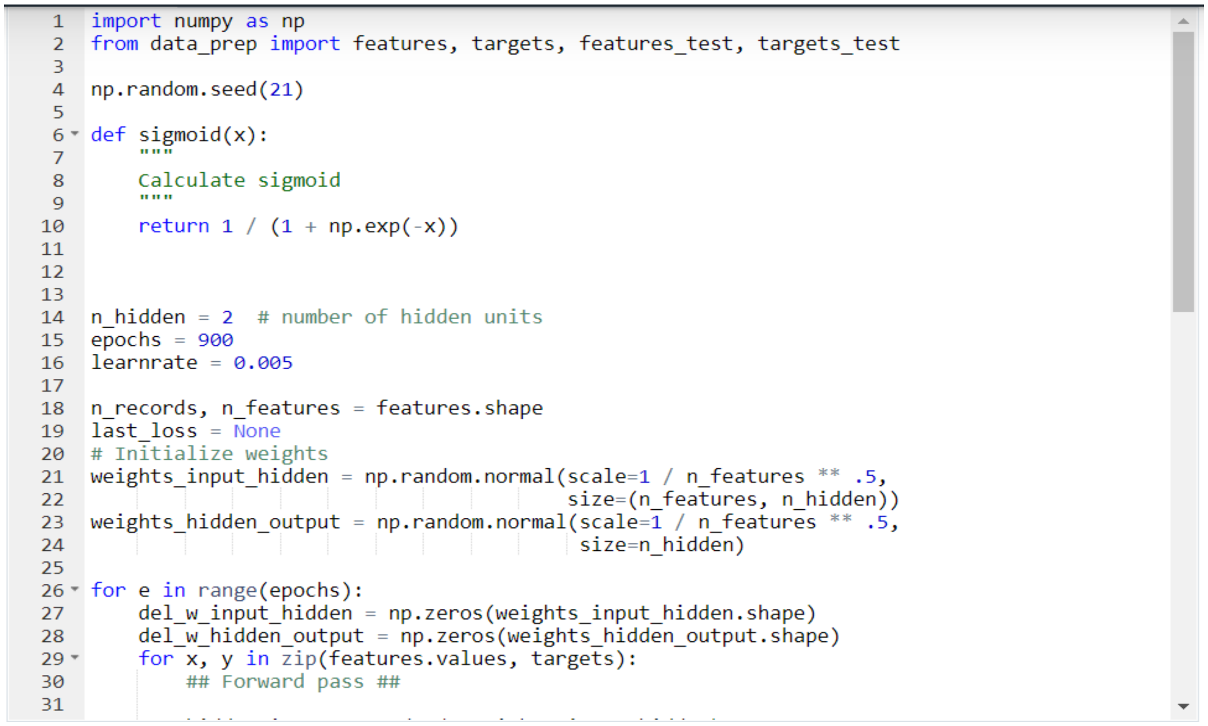
Fig. 30 -Default transform

Algorithm 2: **Process image**

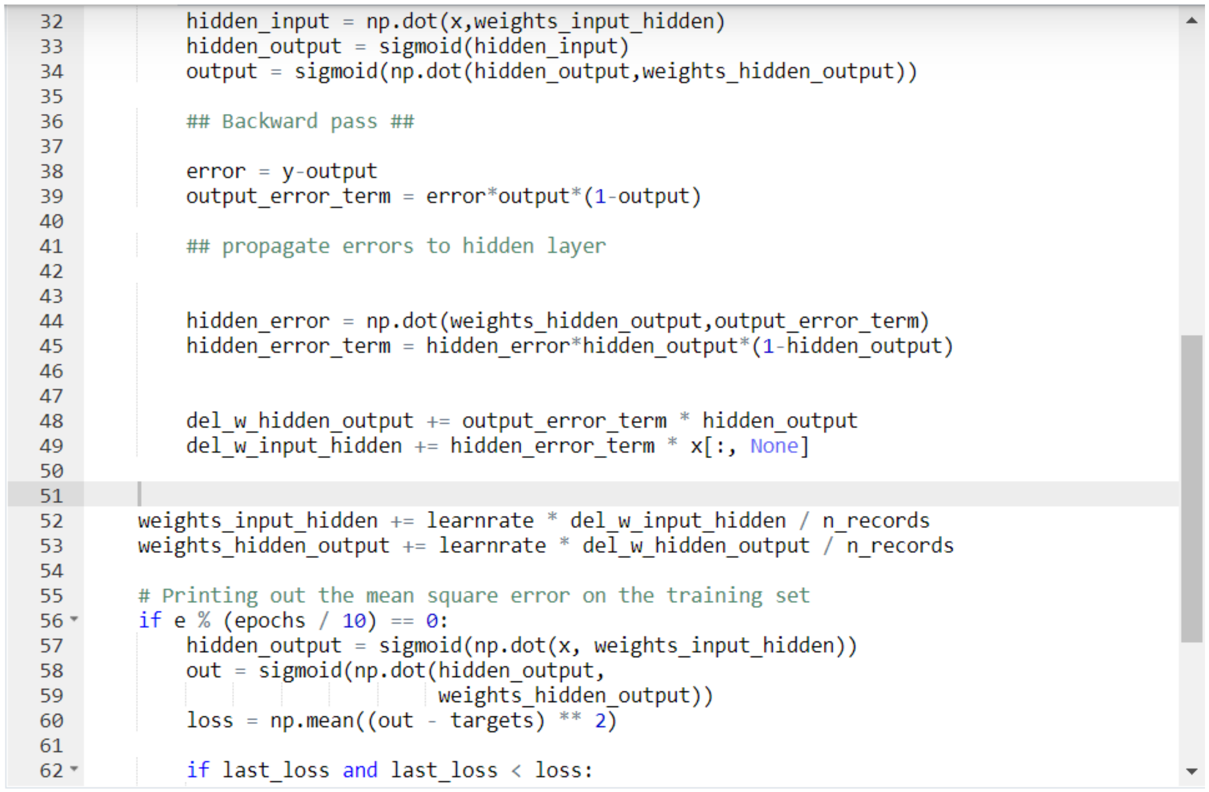
This algorithm is used to adjust and process the images so that all the images have same specification.

```
def image_process(image):
        ->open image
        ->resize image
        ->centercrop the image
        ->totensor
        ->normalize with mean and standard deviation
        ->image_tensor=adjustments(image)
        ->return image_tensor
```

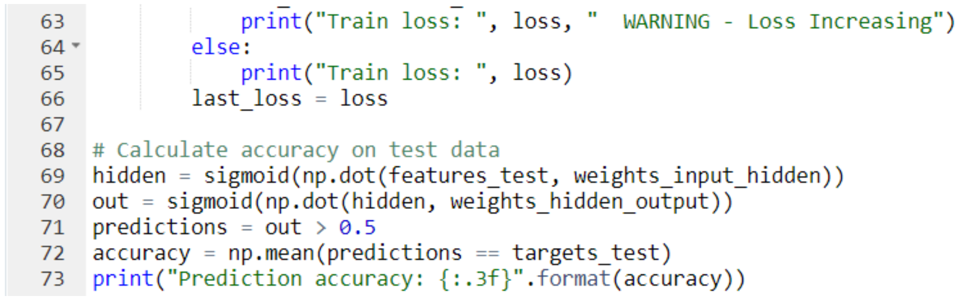
Algorithm 3: **Back propagation**



```python
import numpy as np
from data_prep import features, targets, features_test, targets_test

np.random.seed(21)

def sigmoid(x):
    """
    Calculate sigmoid
    """
    return 1 / (1 + np.exp(-x))



n_hidden = 2  # number of hidden units
epochs = 900
learnrate = 0.005

n_records, n_features = features.shape
last_loss = None
# Initialize weights
weights_input_hidden = np.random.normal(scale=1 / n_features ** .5,
                                        size=(n_features, n_hidden))
weights_hidden_output = np.random.normal(scale=1 / n_features ** .5,
                                         size=n_hidden)

for e in range(epochs):
    del_w_input_hidden = np.zeros(weights_input_hidden.shape)
    del_w_hidden_output = np.zeros(weights_hidden_output.shape)
    for x, y in zip(features.values, targets):
        ## Forward pass ##
```



```
        hidden_input = np.dot(x,weights_input_hidden)
        hidden_output = sigmoid(hidden_input)
        output = sigmoid(np.dot(hidden_output,weights_hidden_output))

        ## Backward pass ##

        error = y-output
        output_error_term = error*output*(1-output)

        ## propagate errors to hidden layer


        hidden_error = np.dot(weights_hidden_output,output_error_term)
        hidden_error_term = hidden_error*hidden_output*(1-hidden_output)


        del_w_hidden_output += output_error_term * hidden_output
        del_w_input_hidden += hidden_error_term * x[:, None]


    weights_input_hidden += learnrate * del_w_input_hidden / n_records
    weights_hidden_output += learnrate * del_w_hidden_output / n_records

    # Printing out the mean square error on the training set
    if e % (epochs / 10) == 0:
        hidden_output = sigmoid(np.dot(x, weights_input_hidden))
        out = sigmoid(np.dot(hidden_output,
                             weights_hidden_output))
        loss = np.mean((out - targets) ** 2)

        if last_loss and last_loss < loss:
```



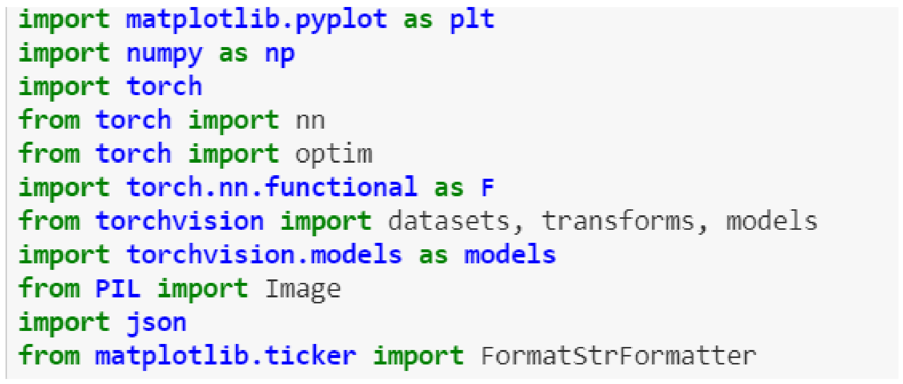
```
            print("Train loss: ", loss, "  WARNING - Loss Increasing")
        else:
            print("Train loss: ", loss)
        last_loss = loss

# Calculate accuracy on test data
hidden = sigmoid(np.dot(features_test, weights_input_hidden))
out = sigmoid(np.dot(hidden, weights_hidden_output))
predictions = out > 0.5
accuracy = np.mean(predictions == targets_test)
print("Prediction accuracy: {:.3f}".format(accuracy))
```

Fig 31: back propagation code

CHAPTER 5: TESTING

First thing to remember is to import every library which we have used in the python application, so that unnecessary error of function not found is deterring the compiling process.

Before compiling, make sure to import every library in the file:



```
import matplotlib.pyplot as plt
import numpy as np
import torch
from torch import nn
from torch import optim
import torch.nn.functional as F
from torchvision import datasets, transforms, models
import torchvision.models as models
from PIL import Image
import json
from matplotlib.ticker import FormatStrFormatter
```

Fig. 32 - Modules

5.1 **Command line testing**

Command line not merely helpful in compiling and executing the file but it can also be used in passing the parameters

-> Directory can be passed in command line along with file name

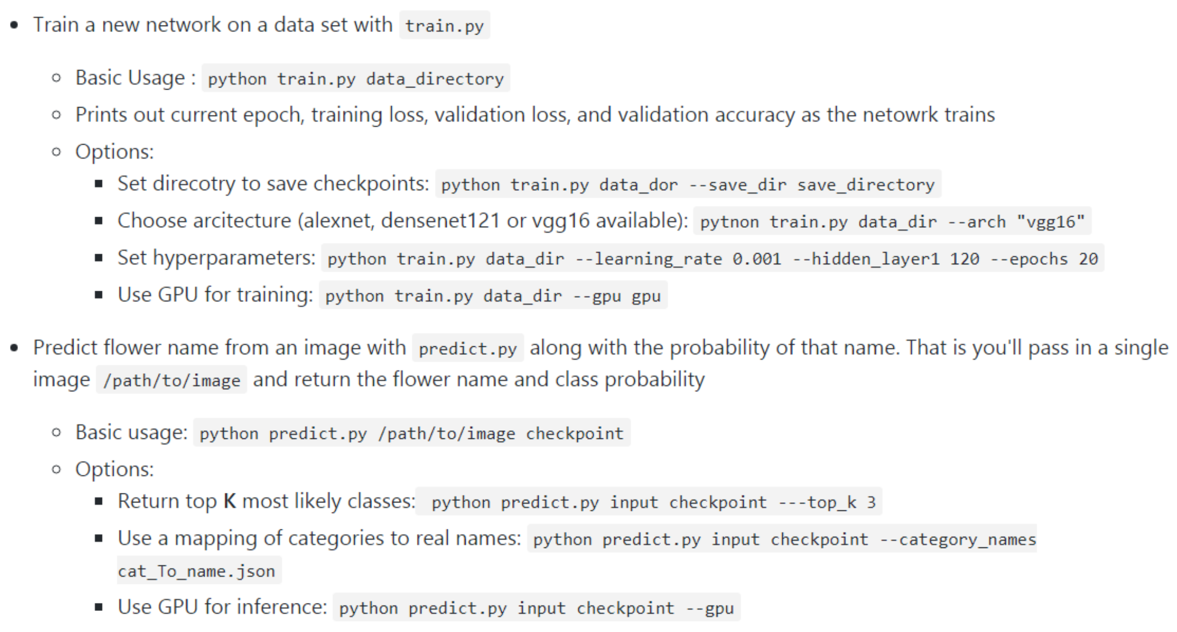
->architecture type can be passed in command line

-> set hyperparameters

->GPU selection

->k-most likely cases

->mapping to real names

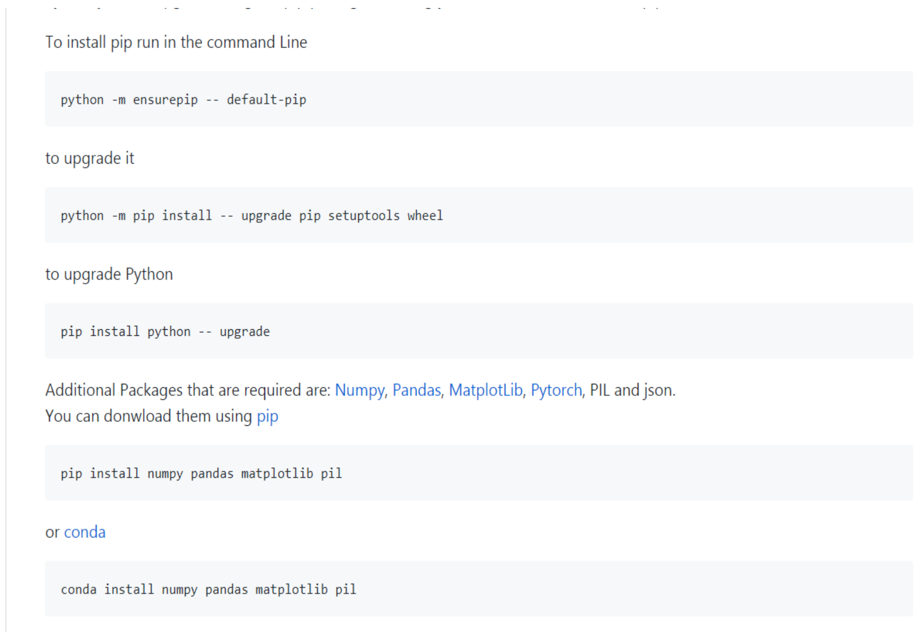


- Train a new network on a data set with `train.py`
  - Basic Usage : `python train.py data_directory`
  - Prints out current epoch, training loss, validation loss, and validation accuracy as the netowrk trains
  - Options:
    - Set direcotry to save checkpoints: `python train.py data_dor --save_dir save_directory`
    - Choose arcitecture (alexnet, densenet121 or vgg16 available): `pytnon train.py data_dir --arch "vgg16"`
    - Set hyperparameters: `python train.py data_dir --learning_rate 0.001 --hidden_layer1 120 --epochs 20`
    - Use GPU for training: `python train.py data_dir --gpu gpu`
- Predict flower name from an image with `predict.py` along with the probability of that name. That is you'll pass in a single image `/path/to/image` and return the flower name and class probability
  - Basic usage: `python predict.py /path/to/image checkpoint`
  - Options:
    - Return top **K** most likely classes: `python predict.py input checkpoint ---top_k 3`
    - Use a mapping of categories to real names: `python predict.py input checkpoint --category_names cat_To_name.json`
    - Use GPU for inference: `python predict.py input checkpoint --gpu`

Fig. 33 - Command line examples

5.2 **Software Testing**

VGG, Resnet and Alexnet needs to be downloaded from pytorch before using. Using the model before downloading will not compile the program.



To install pip run in the command Line

```
python -m ensurepip -- default-pip
```

to upgrade it

```
python -m pip install -- upgrade pip setuptools wheel
```

to upgrade Python

```
pip install python -- upgrade
```

Additional Packages that are required are: Numpy, Pandas, MatplotLib, Pytorch, PIL and json.
You can donwload them using pip

```
pip install numpy pandas matplotlib pil
```

or conda

```
conda install numpy pandas matplotlib pil
```

Fig. 34 - running python using pip

To use the program on PC requires Anaconda, Gitbash and python installed. Gitbash will act as Terminal like in Ubuntu and Mac.
A third party Code editor is required for implementation of program.

5.3 **Dataset testing**

Proportion of training and testing should be sufficient for both the section to make a efficient program.

A dataset is to be divided into 3 parts:

1. Train
2. Validate
3. Test

Train dataset is used to train the model, more and more data is used to train the model more intelligence is the model. But a proportion is to be maintained between train data and test data to train the effectively and to test for high accuracy.
Generally 80-20 is the most effectively used proportion with 80% of the total data being used for training and the rest 20% for testing.

5.4 **Algorithmic testing**

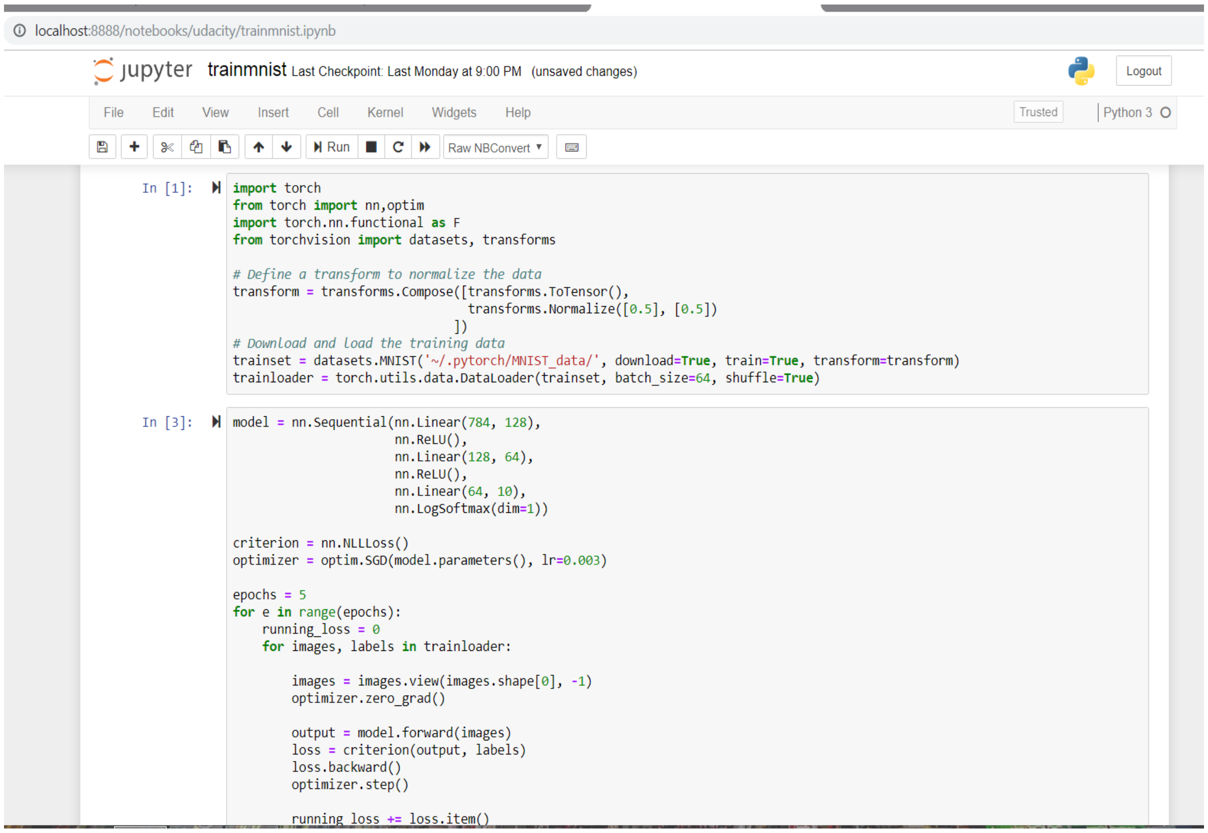
How to predict the class of the image is top 2 classes in the result is of equal probability?
A algorithm should be designed to define the class of a tie breaker image by comparing the image with both the classes' images and whichever class get the higher accuracy is the class of the tie breaker image.

CHAPTER 6: CONCLUSION AND FUTURE WORKS

Conclusion: During this project, I have implemented various types of networks whether it is self created or pre-trained networks and in those networks accuracy varies over large range for example in self made network the accuracy typically ranges from 70% to 80%, and in self made network with controlled epoches, validation data, and dropout modules the accuracy typically ranges from 80% to 85%. To further increase this, a pretrained network has been implemented which has been trained over ImageNet data, and the accuracy has been found in range of above 90%. To conclude, in terms of classification of images, if the model is small i.e number of input and output is less, than self designed classification can be implemented but if the network model is big such as number of output is very large in number than it is better to use pretrained network.

Future works: Although a lot of work can be done in terms of increasing the accuracy of self designed model, but specifically a significant amount of work can be done in defining which pretrained model works best in terms of accuracy.

Working Screenshots:

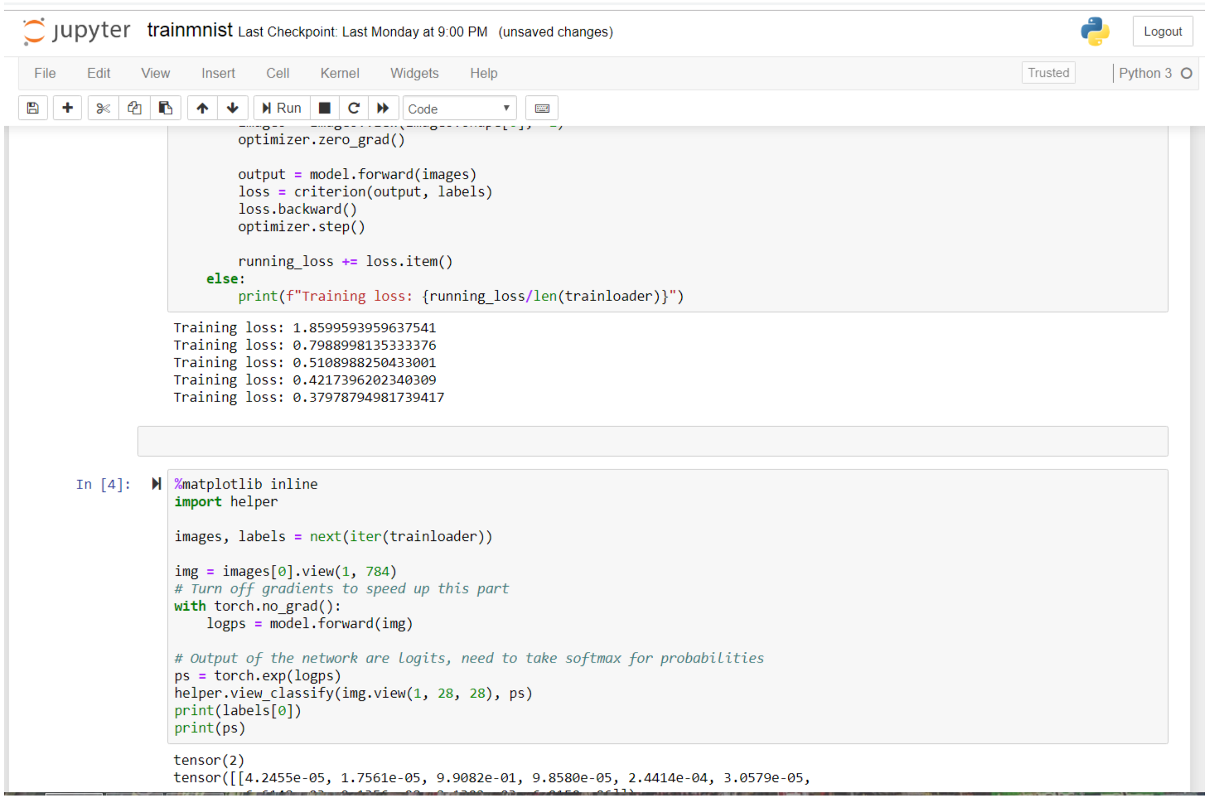


localhost:8888/notebooks/udacity/trainmnist.ipynb

jupyter trainmnist Last Checkpoint: Last Monday at 9:00 PM (unsaved changes)

In [1]:
```
import torch
from torch import nn,optim
import torch.nn.functional as F
from torchvision import datasets, transforms

# Define a transform to normalize the data
transform = transforms.Compose([transforms.ToTensor(),
                                transforms.Normalize([0.5], [0.5])
                              ])
# Download and load the training data
trainset = datasets.MNIST('~/.pytorch/MNIST_data/', download=True, train=True, transform=transform)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=64, shuffle=True)
```

In [3]:
```
model = nn.Sequential(nn.Linear(784, 128),
                      nn.ReLU(),
                      nn.Linear(128, 64),
                      nn.ReLU(),
                      nn.Linear(64, 10),
                      nn.LogSoftmax(dim=1))

criterion = nn.NLLLoss()
optimizer = optim.SGD(model.parameters(), lr=0.003)

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()
```



jupyter trainmnist Last Checkpoint: Last Monday at 9:00 PM (unsaved changes)

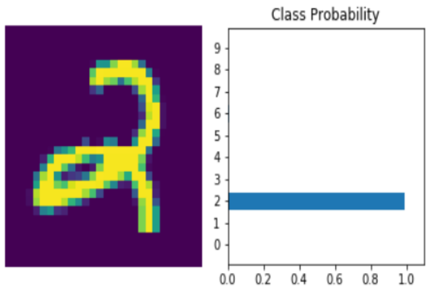
```
        optimizer.zero_grad()

        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)}")
```

```
Training loss: 1.8599593959637541
Training loss: 0.7988998135333376
Training loss: 0.5108988250433001
Training loss: 0.4217396202340309
Training loss: 0.37978794981739417
```

In [4]:
```
%matplotlib inline
import helper

images, labels = next(iter(trainloader))

img = images[0].view(1, 784)
# Turn off gradients to speed up this part
with torch.no_grad():
    logps = model.forward(img)

# Output of the network are logits, need to take softmax for probabilities
ps = torch.exp(logps)
helper.view_classify(img.view(1, 28, 28), ps)
print(labels[0])
print(ps)
```

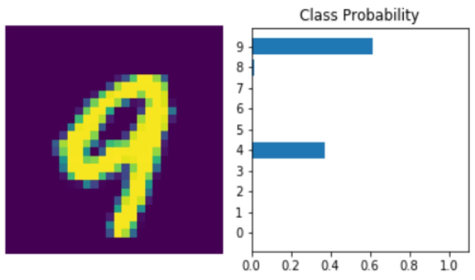
```
tensor(2)
tensor([[4.2455e-05, 1.7561e-05, 9.9082e-01, 9.8580e-05, 2.4414e-04, 3.0579e-05,
In [4]:  %matplotlib inline
         import helper

         images, labels = next(iter(trainloader))

         img = images[0].view(1, 784)
         # Turn off gradients to speed up this part
         with torch.no_grad():
             logps = model.forward(img)

         # Output of the network are logits, need to take softmax for probabilities
         ps = torch.exp(logps)
         helper.view_classify(img.view(1, 28, 28), ps)
         print(labels[0])
         print(ps)
```

```
tensor(2)
tensor([[4.2455e-05, 1.7561e-05, 9.9082e-01, 9.8580e-05, 2.4414e-04, 3.0579e-05,
         6.6148e-03, 9.1356e-08, 2.1302e-03, 6.0150e-06]])
```

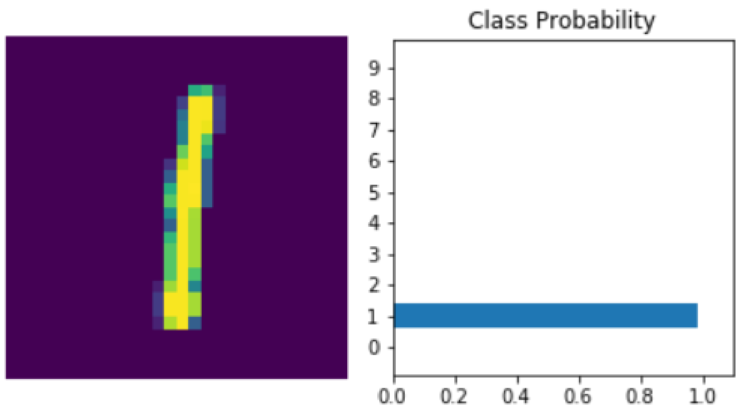


```
images, labels = next(iter(trainloader))

img = images[0].view(1, 784)
# Turn off gradients to speed up this part
with torch.no_grad():
    logps = model.forward(img)

# Output of the network are logits, need to take softmax for probabilities
ps = torch.exp(logps)
helper.view_classify(img.view(1, 28, 28), ps)
print(labels[0])
print(ps)
```

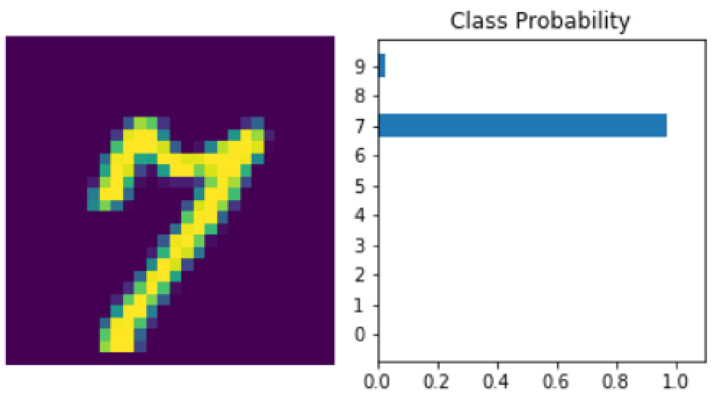
```
tensor(9)
tensor([[1.8565e-04, 1.1899e-07, 3.0189e-04, 4.9557e-05, 3.7140e-01, 2.9218e-04,
         5.0907e-04, 1.9466e-03, 1.2649e-02, 6.1266e-01]])
```



```
tensor(1)
tensor([[1.1943e-06, 9.8708e-01, 2.3312e-03, 4.9890e-03, 4.5646e-05, 1.3645e-03,
         1.2020e-04, 1.3739e-03, 2.3167e-03, 3.7504e-04]])
```



```
tensor(7)
tensor([[1.3863e-05, 1.8340e-06, 1.6791e-06, 2.1198e-05, 1.3752e-03, 7.8426e-05,
         1.7783e-07, 9.7318e-01, 7.6319e-05, 2.5248e-02]])
```



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/

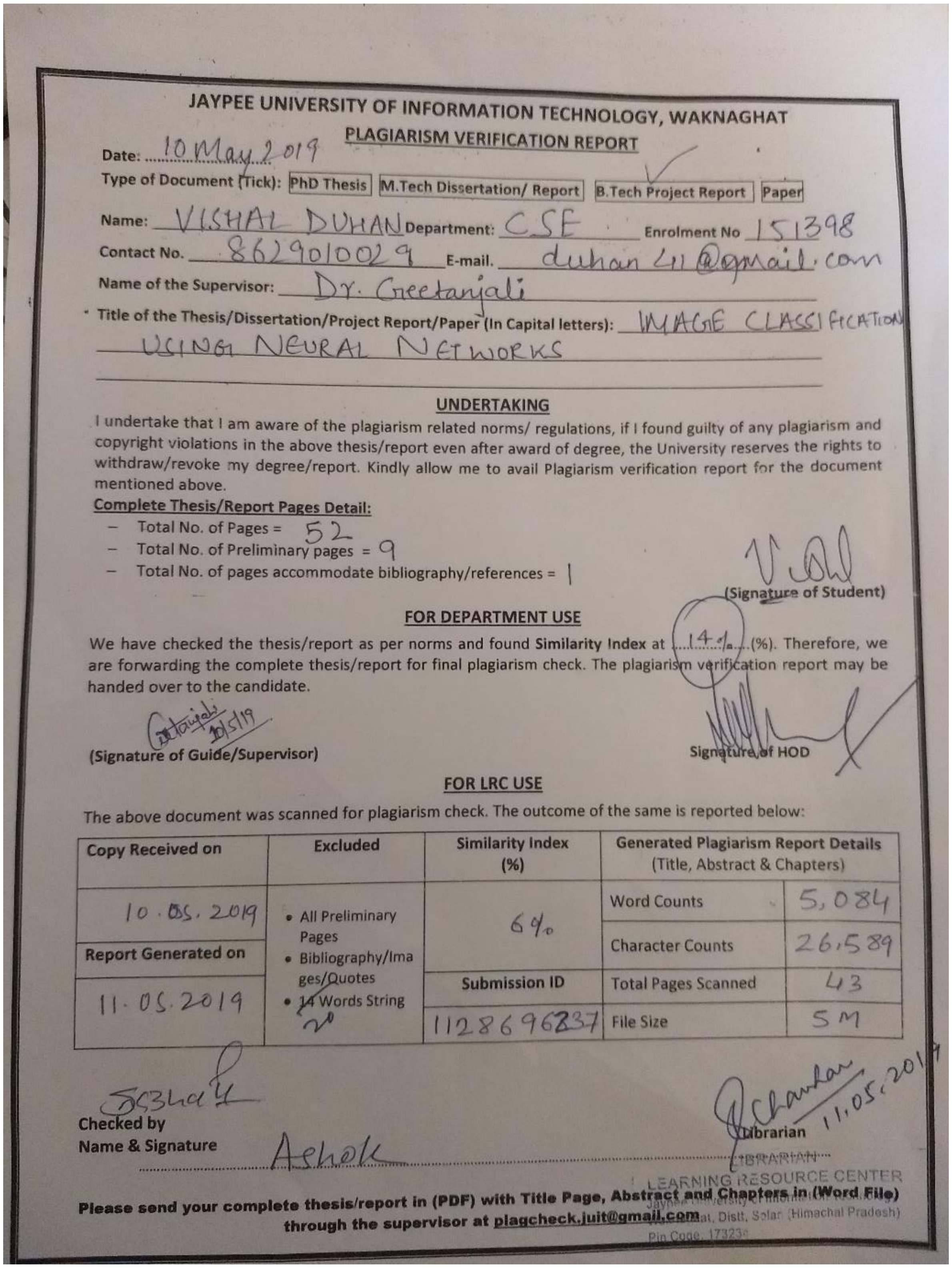
[3] (Online Resource). https://medium.com/intro-to-artificial-intelligence/simple-image-classification-using-deep-learning-deep-learning-series-2-5e5b89e97926

[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



JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY, WAKNAGHAT

PLAGIARISM VERIFICATION REPORT

Date: 10 May 2019

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

Name: VISHAL DUHAN Department: CSE Enrolment No 151398

Contact No. 8629010029 E-mail. duhan41@gmail.com

Name of the Supervisor: Dr. Geetanjali

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 = 52
- Total No. of Preliminary pages = 9
- Total No. of pages accommodate bibliography/references = 1

[illegible]
(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.

Geetanjali 10/5/19
(Signature of Guide/Supervisor)

[illegible]
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)	
10.05.2019	• All Preliminary Pages • Bibliography/Images/Quotes • ~~14~~ 20 Words String	6%	Word Counts	5,084
			Character Counts	26,589
Report Generated on		Submission ID	Total Pages Scanned	43
11.05.2019		1128696237	File Size	5M

[illegible]
Checked by
Name & Signature Ashok

[illegible] 11.05.2019
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