FASHION GENERATION USING GENERATIVE ADVERSARIAL NETWORKS

MAJOR PROJECT FINAL REPORT

Enrolment No.: 16102133

Name of Student: Pragya Goyal

Name of Supervisor: Dr. Pankaj Kumar Yadav

Submitted in Partial Fulfilment of the Degree of

Bachelor of Technology

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING,

JAYPEE INSTITUTE OF INFORMATION TECHNOLOGY, NOIDA

(I)

TABLE OF CONTENTS

(II)

CERTIFICATE

This is to certify that the work titled "**Fashion Generation using Generative Adversarial Networks**" submitted by "**Pragya Goyal"** in partial fulfilment for the award of degree of B. Tech of Jaypee Institute of Information Technology, Noida has been carried out under my supervision. This work has not been submitted partially or wholly to any other University or Institute for the award of this or any other degree or diploma.

Signature of Supervisor …….

Name of Supervisor Dr. Pankaj Kumar Yadav

Designation Assistant Professor (Sr. Grade)

Date …….

(III)

ACKNOWLEDGEMENT

We firmly value and acknowledge the sincere guidance and help of **Dr. Pankaj Kumar Yadav,** Department of Electronics and Communication Engineering, JIIT, NOIDA. His valuable suggestions have constantly added value and creative spirit to the project. His guidance and persistence helped keep us directed during the course of this project. We thank our revered teacher for his constant support during the course of the project.

Signature of Student

Name of Student: Pragya Goyal

Date …….

(IV)

SUMMARY

We learnt about Artificial Intelligence (AI) and Machine Learning (ML), which deals with making machines smarter. We studied about Deep Learning (DL), which is a form of ML (or subset of ML) that is applied to large data sets. It is inspired by the structure of the human brain and is particularly effective in Feature Detection. Deep Learning models are trained with neural networks.

We understood and built basic neural networks with TensorFlow library. TensorFlow is an end-to-end open source platform for ML. We started with building and analysing neural networks for some basic regression and classification problems using TensorFlow and Google-Colaboratory for example, Housing-prices problem. Furthermore, we did detailed study of classification models (Computer vision) through building neural networks for Image classifiers such as Handwriting recognition, Fashion Item Recognition, Horse-Human classifier, Cat-Dog classifier and Happy Sad classifier, etc.

We took this a little further by using convolutions that spot features in an image, and then learn and classify based on those features. We went a little deeper into convolutional neural nets, learning how we can go into depth with real-world images. In Convolutional Neural Networks, we explored how to take advantage of Dropouts, Augmentation, Regularization and Transfer learning, and also looked at binary as well as multi-class classification.

Further, we were introduced to Generative Adversarial Networks and learnt their concept and working. We looked at various types of GANs and understood how to build them. We also learnt their individual advantages and use- cases. Now, we have aimed to apply our knowledge of GANs to real-world scenarios in the fashion industry. We have tried to develop a software which can assist fashion enthusiasts and designers to communicate their ideas quickly and effectively by translating the text description of their imagination to images.

Signature of Student
Name: Pragya Goyal
Date:

Signature of Supervisor
Name: Dr. Pankaj Kumar Yadav
Date:

(V)

LIST OF FIGURES

26.	**1.26**	**15**
27.	**1.27**	**15**
28.	**1.28**	**15**
29.	**1.29**	**15**
30.	**1.30**	**15**
31.	**1.31**	**15**
32.	**1.32**	**15**
33.	**1.33**	**15**
34.	**1.34**	**15**
35.	**1.35**	**15**
36.	**1.36**	**16**
37.	**1.37**	**16**
38.	**1.38**	**16**
39.	**1.39**	**16**
40.	**1.40**	**16**
41.	**2.1**	**17**
42.	**2.2**	**18**
43.	**2.3**	**19**
44.	**2.4**	**19**
45.	**2.5**	**20**
46.	**2.6**	**21**
47.	**3.1**	**23**
48.	**3.2**	**24**
49.	**3.3**	**25**
50.	**3.4**	**25**
51.	**3.5**	**28**
52.	**3.6**	**28**
53.	**5.1**	**41**

(VI)

LIST OF SYMBOLS AND ACRONYMS

AI	Artificial Intelligence
ML	Machine Learning
DL	Deep Learning
NN	Neural Network
CNN	Convolutional Neural Networks
GAN	Generative Adversarial Network
CGAN	Conditional GAN
DCGAN	Deep Convolutional GAN
ACGAN	Auxiliary Classifier GAN
tanh	Hyperbolic Tangent
ReLu	Rectified Linear
HTTP	HyperText Transfer Protocol
URL	Uniform Resource Locator
OS	Operating System
Rps	Rock – Paper - Scissor
MNIST	Modified National Institute of Standards and Technology
2D	2 - Dimensional
MATLAB	MATrix LABoratory
RGB	Red – Green - Blue
3D	3 - Dimensional
RMSprop	Root Mean Square Propagation
PIL	Python Imaging Library
NumPy	Numeric Python
CSV	Comma Separated Values
API	Application Programming Interface
CNTK	Microsoft Cognitive Toolkit
CPU	Central Processing Unit
GPU	Graphical Processing Unit
TFD	Toronto Faces Dataset

1D	1 – Dimensional
Colab	Google Colaboratory
PIL	Python Imaging Library
CV	Computer Vision
NLP	Natural Language Processing
RV	Random Variable
Z	Latent Space
G(z)	Generated image/ Generator output
D(G(z))	Discriminator output on fake images
x	Real images
D(x)	Discriminator output on real images
c	Class -label

Chapter -1

INTRODUCTION

Project Objective

Artificial Intelligence has recently been employed for various applications in the fashion industry. From recommendation systems at e-commerce sites like Myntra to virtual clothes' try-on facilities. However, there are still many aspects of the fashion industry that Artificial Intelligence methods have not been applied to. We aim to develop one such application that can assist fashion designers and enthusiasts in describing their ideas by translating their verbal descriptions to images. Thus, given the text description of a particular clothing item, we aim to be able to generate images that match the given description.

Project Scope

- To assist fashion enthusiasts in communicating their ideas
- To reduce manpower required for conversion of new fashion ideas to reality

Project Requirements

- Understanding of Python Programming
- Understanding of Artificial Intelligence, Machine Learning, Deep Learning
- Knowledge of how to build neural networks
- Grasp on the concept and working of Generative Adversarial Networks
- A Categorical Dataset with cloth images diversified with multiple labels

Dataset Preparation

1. Decided on the **constraints** of the Dataset:
    a. Gender: Men, Women (2)
    b. Cloth Type: Tshirt (1)
    c. Colour: Black, White (2)
    d. Pattern: Solid/Plain, Horizontal Striped (2)

    (2x1x2x2 = 8 categories)

2. Scraped images for all categories from Myntra, and Google images.
3. Sorted and renamed all images corresponding to their category.
4. Made 32 folders corresponding to 32 categories and copied images to their corresponding folders.
5. Wrote and executed code for resizing of all images within all categories to 28x28 size and to convert them to grayscale.

Dataset Description

Compiled a dataset comprising of 125 images distributed among the following categories :-

Fig 1.1-1.5: Men-Tshirt-Black –Solid

Fig 1.6-1.10: Men-Tshirt-Black –Striped

Fig 1.11-1.15: Men-Tshirt-White-Solid

Fig 1.16-1.20: Men-Tshirt-White-Striped

Fig 1.21-1.25: Women-Tshirt-Black-Solid

Fig 1.26-1.30: Women-Tshirt-Black-Striped

Fig 1.31-1.35: Women-Tshirt-White-Solid

Fig 1.36-1.40: Women-Tshirt-White-Striped

Chapter -2

FOUNDATIONS OF ML & DL

What are AI, ML & DL?

Artificial Intelligence (AI) is the hood that covers anything related to making machines smart. For example, a robot, a car, a refrigerator, a software application; if you are making them smart, then you are using AI.

Machine Learning (ML) though similar to AI, is not the same. In fact, it can be said to be a subset of AI. It is a branch of Artificial Intelligence where machines can learn by themselves. Over time, the machines get smarter and smarter without human intervention. Machines apply algorithms that parse/ go-through data, learn from that data and apply their learning to make informed decisions.

Deep Learning (DL) is further a subset of Machine Learning where machines learn from very large data sets. Deep Learning is influenced by the structure of the human brain. It works especially well for feature detection. In DL, algorithms are created and work same as in the case of ML, but there are many layers of these algorithms, making a kind of a network of algorithms called as Artificial Neural Networks. They are named so in accordance to the fact that they attempt to imitate the actual neural networks in the human brain.

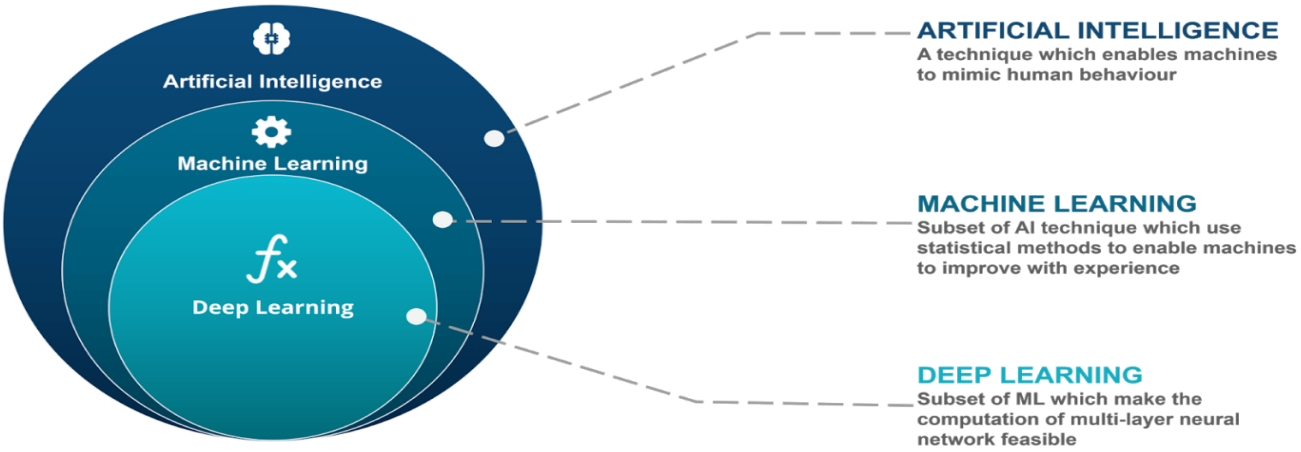


Fig 2.1: Representation of AI, ML and DL

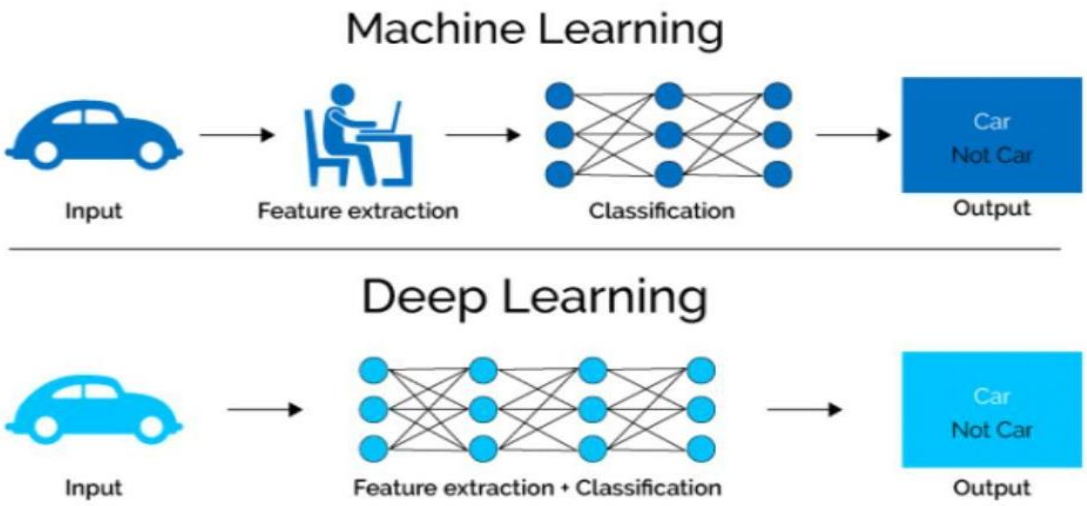


Fig 2.2: Difference between ML and DL

Neural Networks

Neural networks (or Neural nets or NNs) refer to a network of 'algorithms'. These networks are loosely modelled on the structure of the human brain and are designed to recognize patterns. They recognize patterns in a numerical, contained-in-vectors' form, into which all the real-world data, be it text, images, sound or time- series, must be translated.

Neural networks can be thought of as a clustering and classification layer on top of the data we store and manage. They are fed with some labelled examples as input, which they are trained on. Neural networks learn from this input data and can apply this understanding to classify some new unlabelled data by finding similarities with the training data.

In simpler words, a Neural network is a type of model that can be trained to identify patterns in data. It consists of an input layer, an output layer and at least one hidden layer. The neurons in each of these learn increasingly abstract representations of data. These representations (or learned features) make it possible to classify data.

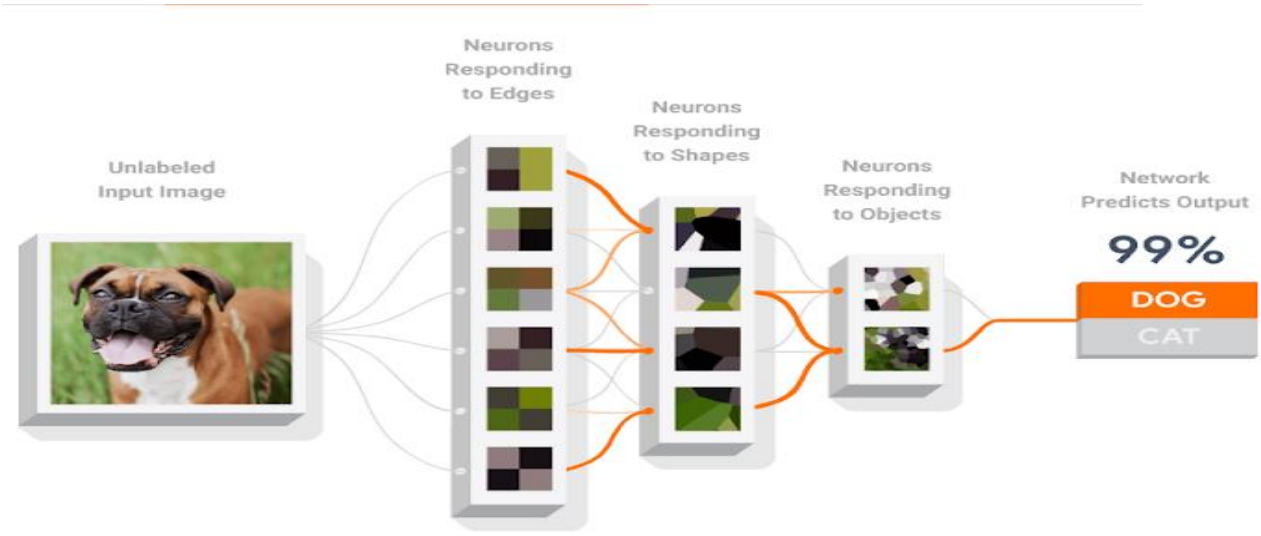


Fig 2.3: Representation of a Neural Network

Convolutional Neural Networks

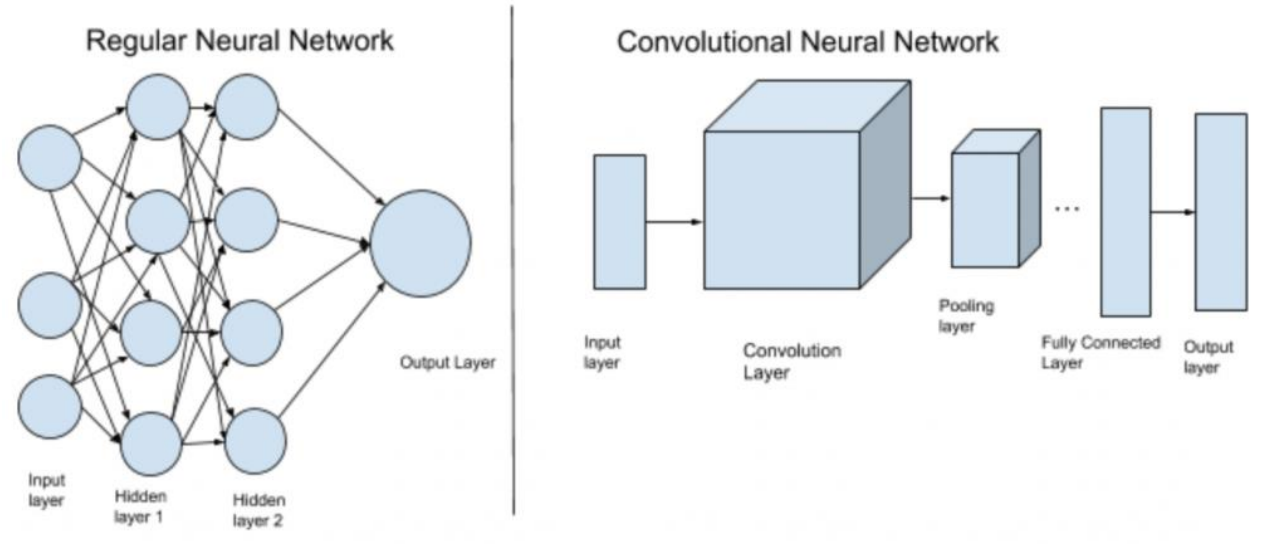


Fig 2.4: Convolutional Neural Network

Applications :-

- Image processing
- Computer Vision
- Speech Recognition
- Machine translation

Convolutional neural nets contain a 3-dimensional arrangement of neurons, instead of the standard two-dimensional matrix. The first layer is called a convolutional layer. Only the information from a small part of the visual field is processed by each neuron in this layer. Input features are taken in batch wise like a filter. The network understands the data in parts and can compute these operations multiple times to complete data processing.

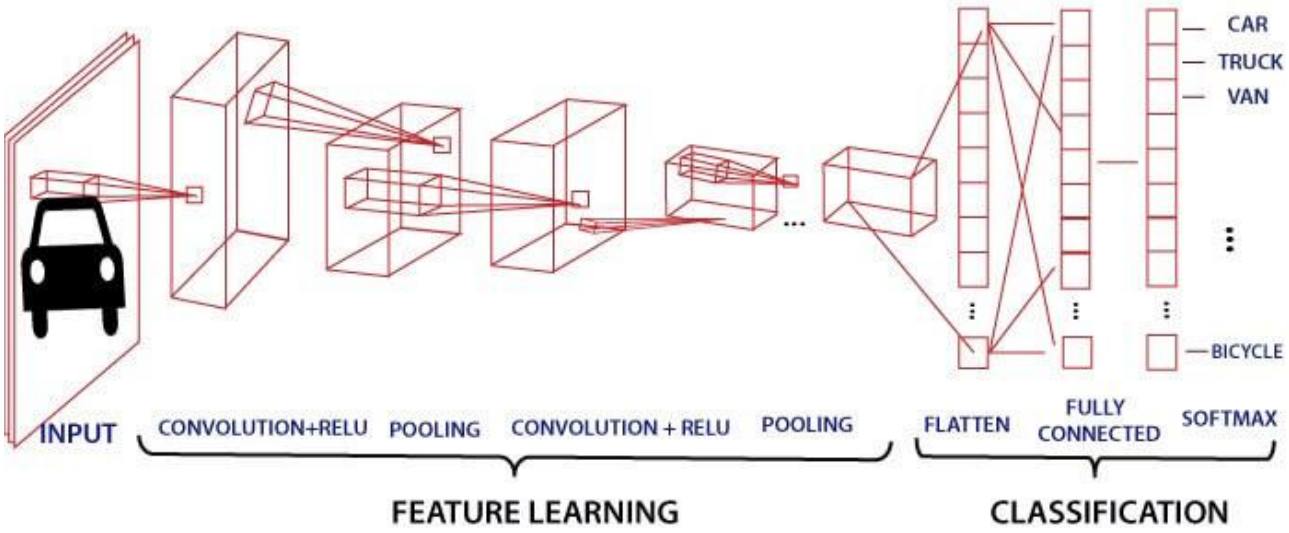


Fig 2.5: Representation of CNN

Advantages :-

- Used for deep learning with few parameters
- Less parameters to learn as compared to fully connected layer

Disadvantages :-

- Comparatively complex to design and maintain
- Comparatively slow [depends on the number of hidden layers]

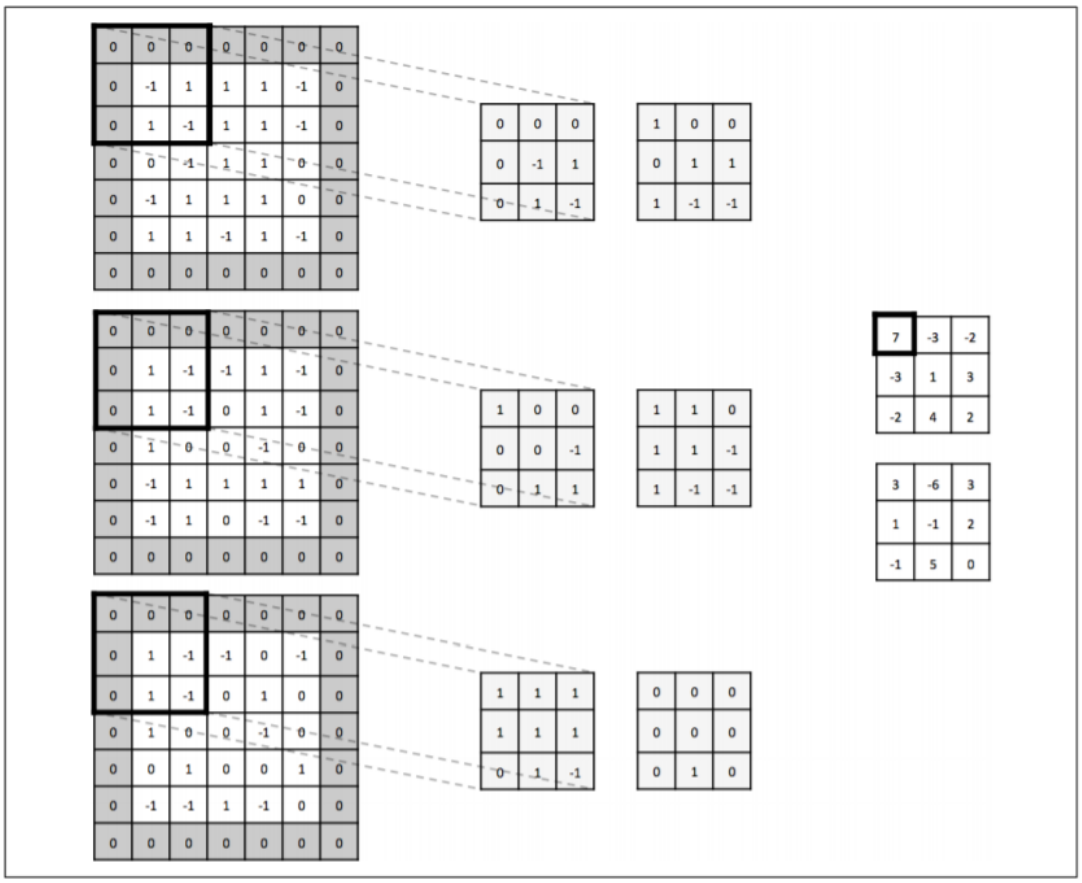


Fig 2.6: This is a convolutional layer with an input volume that has width and height of 5, depth of 3, and zero padding 1. There are 2 filters, with spatial extent 3x3 and applied with a stride of 2. It results in an output volume with width 3, height 3, and depth 2. We apply the first convolutional filter to the upper left most 3 x 3 piece of the input volume to generate the upper-left most entry of the first depth slice.

Jupyter Notebook

The Jupyter Notebook is an open-source web application that allows us to create and share documents that contain live code, visualizations, equations and narrative text.

Uses include: machine learning, numerical simulation, data cleaning and transformation, data visualization, statistical modelling, and much more.

Google Colaboratory

Colaboratory is a tool for machine learning education and research. It's a Jupyter notebook environment which requires no setup and runs entirely in the cloud. Colaboratory supports Python 2.7 and Python 3.6.

Jupyter is the open source project on which Colaboratory is based. Colaboratory allows you to use and share Jupyter notebooks with others without having to install, download, or run anything on our computer other than a browser.

Keras

Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano. It enables fast experimentation.
Being able to go from idea to result with the least possible delay is key to doing good research.

Features of Keras :-

- Allows for easy and fast prototyping (through user friendliness, modularity, and extensibility).
- Supports both convolutional networks and recurrent networks, as well as combinations of the two.
- Runs seamlessly on CPU and GPU.

Keras is compatible with: **Python 2.7-3.6**.

Chapter -3

INTRODUCTION TO GENERATIVE ADVERSARIAL NETWORKS (GANs)

Introduction

Generative Adversarial Networks are a class of Machine Learning frameworks invented by Ian Goodfellow and his colleagues in 2014. Two neural nets (a Generator and a Discriminator) compete with each other in a game. Given a training dataset, this technique learns to generate new data with the same statistics as the training set.

For example, a GAN trained on human faces can generate new faces that look at least superficially authentic to human observers, having many realistic characteristics.

Fig 3.1: Illustration of GANs abilities by Ian Goodfellow and co-authors. These are samples generated by Generative Adversarial Networks after training on two datasets: MNIST and TFD. For both, the rightmost column contains true data that are the nearest from the direct neighboring generated samples. This shows us that the produced data are really generated and not only memorised by the network.

Method

The generative network generates samples while the discriminative network evaluates them. The competition operates in terms of data distributions. Typically, the generative network learns to map from a latent space to a data distribution of interest, while the discriminative network distinguishes candidates produced by the generator from the true data distribution. The generative network's

training objective is to increase the error rate of the discriminative network (i.e., "fool" the discriminator network by producing novel samples that the discriminator thinks are not synthesized (are part of the true data distribution)).

1. The discriminator is trained using some known dataset, until it achieves acceptable accuracy.
2. The generator trains based on whether it succeeds in fooling the discriminator. Typically, the generator is seeded with randomized input that is sampled from a predefined latent space (e.g. a multivariate normal distribution).
3. Thereafter, samples made by the generator are fed to and evaluated by the discriminator.
4. Backpropagation is applied to both networks so that the discriminator becomes more skilled at identifying synthetic images, while the generator produces better images.

The generator is typically a deconvolutional neural network, and the discriminator is a convolutional neural network.

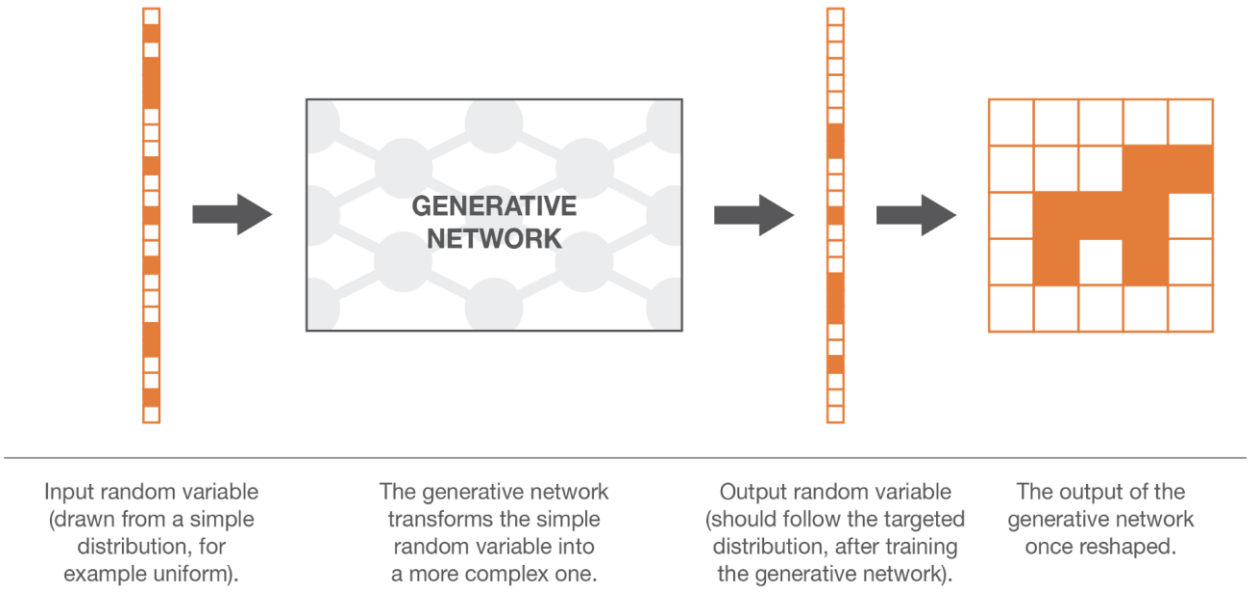


Fig 3.2: Illustration of the notion of generative models using neural networks. Obviously, the dimensionality we are really talking about are much higher than represented here.

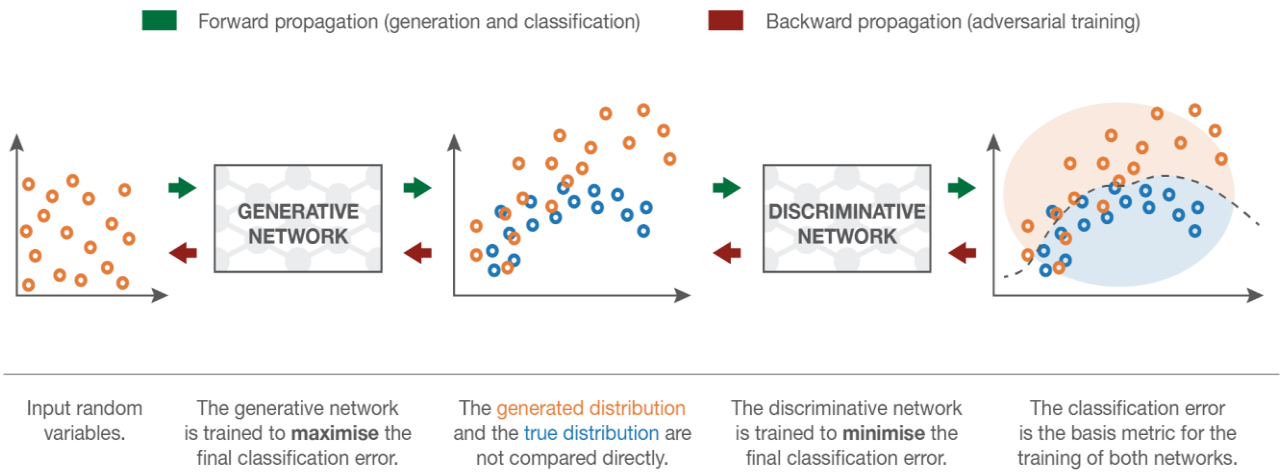


Fig 3.3: Explanation of the working of GAN model

Generative Adversarial Networks representation. The generator takes simple random variables as inputs and generate new data. The discriminator takes "true" and "generated" data and try to distinguish them, building a classifier. The goal of the generator is to fool the discriminator (increase the classification error by mixing up as much as possible generated data with true data) and the goal of the discriminator is to distinguish between true and generated data.

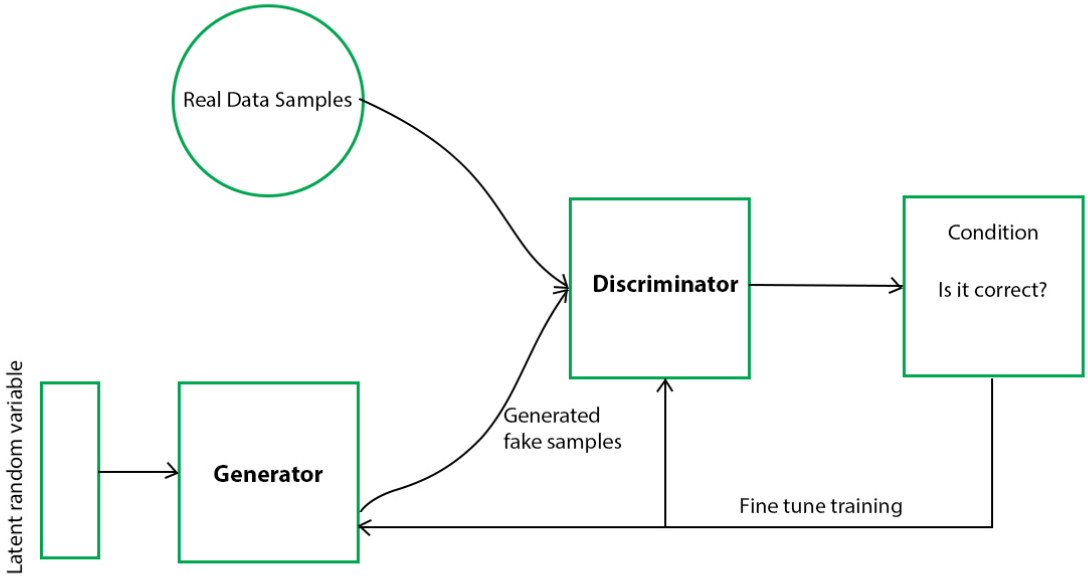


Fig 3.4: Working of a basic Generative Adversarial Network with the help of block diagram

Applications

- Generate Examples for Image Datasets
- Generate Photographs of Human Faces
- Generate Realistic Photographs
- Photos to Emojis
- Generate Cartoon Characters
- Face Aging
- Image-to-Image Translation
- Text-to-Image Translation
- Semantic-Image-to-Photo Translation
- Generate New Human Poses
- Photograph Editing
- Photo Blending
- Super Resolution
- Photo Inpainting
- Clothing Translation
- Video Prediction
- 3D Object Generation
- Face Frontal View Generation

Some Common Types of GANs

- Basic GAN

  GANs often suffer from a "mode collapse" where they fail to generalize properly, missing entire modes from the input data. For example, a GAN trained on the MNIST dataset containing many samples of each digit, might nevertheless timidly omit a subset of the digits from its output. Many solutions have been proposed.

  Presentation: https://1drv.ms/p/s!Ag4U2hk3CrNy7xEV5J5xJHG1QQve

- Conditional GAN (CGAN)

  In an unconditioned generative model, there is no control on modes of the data being generated. However, by conditioning the model on additional information it is possible to direct the data generation process. Such conditioning could be based on class labels, on some part of data for in painting, or even on data from different modality [3].

  Presentation: https://1drv.ms/p/s!Ag4U2hk3CrNy7xR9fgjh-SHpXs7v
  Google Colaboratory Link:
  https://colab.research.google.com/drive/1y2vM4X3PNWaIsMtk3dkNDqIKmK8NtcEV?usp=sharing

- Deep Convolutional GAN (DCGAN)

  Provides an architectural topology which makes generative adversarial network models more stable to train in most settings. They are also a strong candidate for unsupervised learning [4].

  Google Colaboratory Link:
  https://colab.research.google.com/drive/1ZxU2jdV7UqepQjGmthxkrlPoDn7m05Tq?usp=sharing

- StackGAN

  For synthesizing high-quality images from text descriptions using generative adversarial training. Conditioned on given text descriptions, conditional GANs are able to generate images that are highly related to the text meanings [5]. Used StackGAN to convert text descriptions of birds, flowers and objects with various backgrounds, to images.

- Auxiliary Classifier GAN (ACGAN)

  ACGAN is similar in principle to the Conditional GAN (CGAN) but in ACGAN, the discriminator also returns the class- label along with probability that the image is real. It is really useful if we only want to generate images for some particular class-labels[15].

  Google Colaboratory Link:
  https://colab.research.google.com/drive/19DQwbyDx2aKrqkfjsFAPZKIuSfNMoGSE?usp=sharing

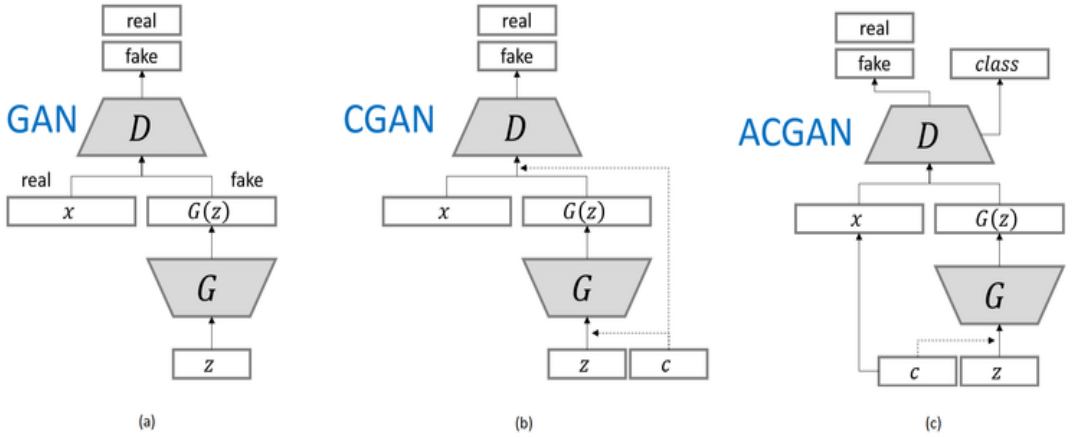


Fig 3.5: Comparison of basic GAN, CGAN and ACGAN with help of block diagrams

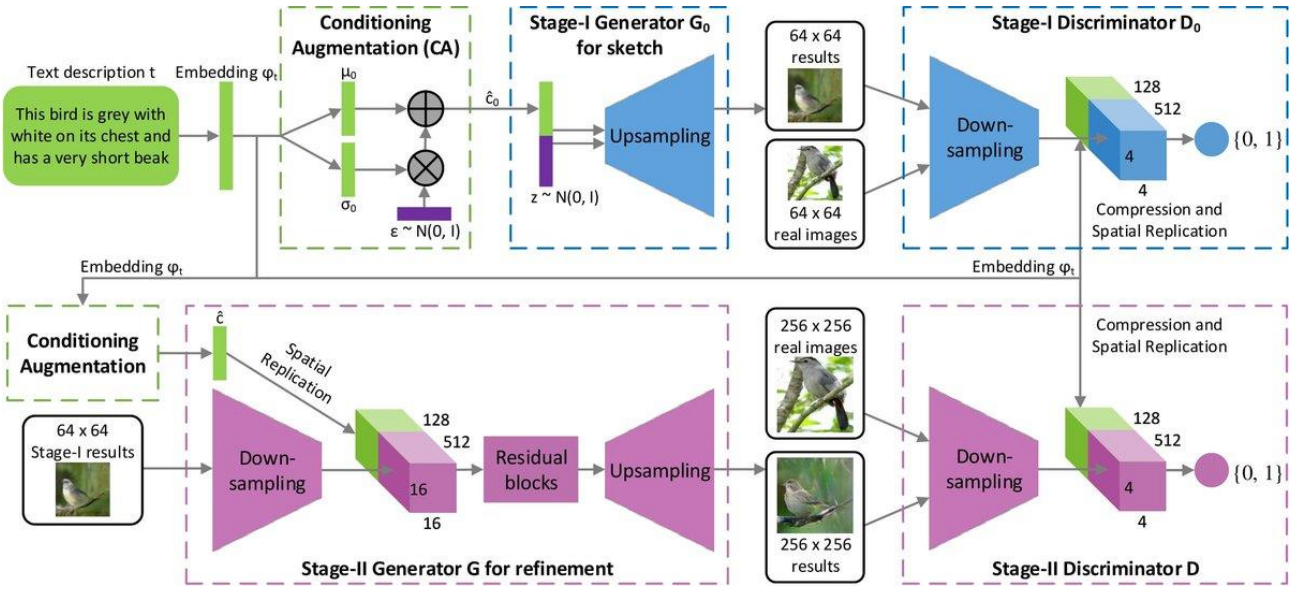


Fig 3.6: Representation of StackGAN

Chapter -4

CODE & DESCRIPTION

To Convert images to size 28x28 and grayscale :-

```
# -*- coding: utf-8 -*-
"""
Created on Sat May 23 13:19:45 2020

@author: PRAGYA
"""

import os
import PIL
import glob
print('Pillow Version:', PIL.__version__)
from PIL import Image
from PIL import ImageOps
root_dir = "Dataset"
cnt=0
classes_dir = ['MBkSl', 'MBkSt', 'MWSl', 'MWSt', 'WBkSl', 'WBkSt', 'WWSl', 'WWSt']
for cls in classes_dir:
    src = root_dir +'\\'+cls
    data_path = os.path.join(src,'*g')
    files = glob.glob(data_path)
    #allFileNames = os.listdir(src)
    for name in files:
        img=Image.open(name,"r")
        img=img.resize((28,28))
        img=img.convert('L')
        cnt+=1
        img.save(name)
print(cnt)
```

Importing OS module which provides a way for using operating system functionality.

Importing PIL (Python Imaging Library) which is a free library that can open, manipulate and save many different image file formats.

Importing glob for finding all the pathnames matching a specified pattern. (In our case '*g' for finding all image files).

From PIL library, we import Image and ImageOps modules.

We made a variable named root_dir which stores the path for the 8 folders in which the images are stored.

We made a list of the names of all the 8 folders in the root-dir. These also correspond to the respective 8 categories.

We loop a variable cls for all the classes_dir.

Inside the loop, we define a variable src which will contain the complete path required to access the images of each label.

We make use of OS and glob modules to find the names of all the image files in the src directory.
We loop through all these image filenames, open each image, resize it, convert it to grayscale and save it by the same filename.

Preparing and Labelling dataset :-

In [1]:
```
import numpy as np
import os
import PIL
import glob
from numpy import asarray
print('Pillow Version:', PIL.__version__)
from PIL import Image
trainy=np.zeros(shape=(125))
trainy.fill(-1)
trainX=np.zeros(shape=(125, 28, 28))
i=0
rev_map = {0:'MBkSl', 1:'MBkSt', 2:'MWSl', 3:'MWSt', 4:'WBkSl', 5:'WBkSt', 6:'WWSl', 7:'WWSt'}
_map = {v: k for k, v in rev_map.items()}
root_dir = "C:\\Users\\PRAGYA\\Desktop\\MAJOR PROJECT\\Dataset"
classes_dir = ['MBkSl', 'MBkSt', 'MWSl', 'MWSt', 'WBkSl', 'WBkSt', 'WWSl', 'WWSt']
for cls in classes_dir:
    src = root_dir +'\\'+cls
    data_path = os.path.join(src,'*g')
    files = glob.glob(data_path)
    for name in files:
        img=Image.open(name,"r")
        #data = asarray(img)
        #print(data.shape)
        trainX[i]=img
        #trainX[i] = trainX[i]/255
        trainy[i]=_map[cls]
        #print(trainy)
        i+=1
print(trainy)
```

Imported necessary libraries and modules.
Made an array of zeroes named trainy of 125 length for storing the image labels corresponding to each of the 125 images; and initialized each element to -1.
Made another array trainX for storing the images (of dimension 28x28).
Made 2 dictionaries for easier interpretation of the 8 categories.
We made a variable named root_dir which stores the path for the 8 folders in which the images are stored.
We made a list of the names of all the 8 folders in the root-dir. These also correspond to the respective 8 categories.
We loop a variable cls for all the classes_dir.
Inside the loop, we define a variable src which will contain the complete path required to access the images of each label.

We make use of OS and glob modules to find the names of all the image files in the src directory. We loop through all these image filenames, open each image, store it at the ith index of trainX and its corresponding label at ith index of trainy.

```
Pillow Version: 5.4.1
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 2. 2. 2.
 2. 2. 2. 2. 2. 2. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 4. 4.
 4. 4. 4. 4. 4. 4. 4. 4. 4. 4. 4. 4. 4. 4. 4. 5. 5. 5. 5. 5. 5. 5. 5.
 5. 5. 5. 5. 5. 5. 5. 5. 5. 6. 6. 6. 6. 6. 6. 6. 6. 6. 6. 6. 6. 6. 6.
 6. 6. 6. 6. 7. 7. 7. 7. 7. 7. 7. 7. 7. 7. 7. 7. 7. 7. 7. 7. 7. 7. 7.
 7. 7. 7. 7. 7.]
```

Writing code for the ACGAN model :-

```
from numpy import zeros
from numpy import ones
from numpy import expand_dims
from numpy.random import randn
from numpy.random import randint
#from keras.datasets.fashion_mnist import load_data
from keras.optimizers import Adam
from keras.models import Model
from keras.layers import Input
from keras.layers import Dense
from keras.layers import Reshape
from keras.layers import Flatten
from keras.layers import Conv2D
from keras.layers import Conv2DTranspose
from keras.layers import LeakyReLU
from keras.layers import BatchNormalization
from keras.layers import Dropout
from keras.layers import Embedding
from keras.layers import Activation
from keras.layers import Concatenate
from keras.initializers import RandomNormal
from matplotlib import pyplot
```

Importing all the necessary libraries and modules.

```
# define the standalone discriminator model
def define_discriminator(in_shape=(28,28,1), n_classes=8):
    # weight initialization
    init = RandomNormal(stddev=0.02)
    # image input
    in_image = Input(shape=in_shape)
    # downsample to 14x14
    fe = Conv2D(32, (3,3), strides=(2,2), padding='same', kernel_initializer=init)(in_image)
    fe = LeakyReLU(alpha=0.2)(fe)
    fe = Dropout(0.5)(fe)
    # normal
    fe = Conv2D(64, (3,3), padding='same', kernel_initializer=init)(fe)
    fe = BatchNormalization()(fe)
    fe = LeakyReLU(alpha=0.2)(fe)
    fe = Dropout(0.5)(fe)
    # downsample to 7x7
    fe = Conv2D(128, (3,3), strides=(2,2), padding='same', kernel_initializer=init)(fe)
    fe = BatchNormalization()(fe)
    fe = LeakyReLU(alpha=0.2)(fe)
    fe = Dropout(0.5)(fe)
    # normal
    fe = Conv2D(256, (3,3), padding='same', kernel_initializer=init)(fe)
    fe = BatchNormalization()(fe)
    fe = LeakyReLU(alpha=0.2)(fe)
    fe = Dropout(0.5)(fe)
    # flatten feature maps
    fe = Flatten()(fe)
    # real/fake output
    out1 = Dense(1, activation='sigmoid')(fe)
    # class label output
    out2 = Dense(n_classes, activation='softmax')(fe)
    # define model
    model = Model(in_image, [out1, out2])
    # compile model
    opt = Adam(lr=0.0002, beta_1=0.5)
    model.compile(loss=['binary_crossentropy', 'sparse_categorical_crossentropy'], optimizer=opt)
```

```
    return model
```

Defining model for the discriminator.

Input shape is size of 1 image that is, 28x28x1.

Gives two outputs:

- The probability of the input image being real.
- The most probable class –label of the input image.

```python
# define the standalone generator model
def define_generator(latent_dim, n_classes=8):
    # weight initialization
    init = RandomNormal(stddev=0.02)
    # label input
    in_label = Input(shape=(1,))
    # embedding for categorical input
    li = Embedding(n_classes, 50)(in_label)
    # linear multiplication
    n_nodes = 7 * 7
    li = Dense(n_nodes, kernel_initializer=init)(li)
    # reshape to additional channel
    li = Reshape((7, 7, 1))(li)
    # image generator input
    in_lat = Input(shape=(latent_dim,))
    # foundation for 7x7 image
    n_nodes = 384 * 7 * 7
    gen = Dense(n_nodes, kernel_initializer=init)(in_lat)
    gen = Activation('relu')(gen)
    gen = Reshape((7, 7, 384))(gen)
    # merge image gen and label input
    merge = Concatenate()([gen, li])
    # upsample to 14x14
    gen = Conv2DTranspose(192, (5,5), strides=(2,2), padding='same', kernel_initializer=init)(merge)
    gen = BatchNormalization()(gen)
    gen = Activation('relu')(gen)
    # upsample to 28x28
    gen = Conv2DTranspose(1, (5,5), strides=(2,2), padding='same', kernel_initializer=init)(gen)
    out_layer = Activation('tanh')(gen)
    # define model
    model = Model([in_lat, in_label], out_layer)
    return model
```

Defining model for generator.

Inputs are a latent space of 100 dimensionality and class -label.

Outputs a 28x28 array representing an image.

```
# define the combined generator and discriminator model, for updating the generator
def define_gan(g_model, d_model):
    # make weights in the discriminator not trainable
    d_model.trainable = False
    # connect the outputs of the generator to the inputs of the discriminator
    gan_output = d_model(g_model.output)
    # define gan model as taking noise and label and outputting real/fake and label outputs
    model = Model(g_model.input, gan_output)
    # compile model
    opt = Adam(lr=0.0002, beta_1=0.5)
    model.compile(loss=['binary_crossentropy', 'sparse_categorical_crossentropy'], optimizer=opt)
    return model

# load images
def load_real_samples():
    # load dataset
    #(trainX, trainy), (_, _) = load_data()
    # expand to 3d, e.g. add channels
    X = expand_dims(trainX, axis=-1)
    # convert from ints to floats
    X = X.astype('float32')
    # scale from [0,255] to [-1,1]
    X = (X - 127.5) / 127.5
    print(X.shape, trainy.shape)
    return [X, trainy]

# select real samples
def generate_real_samples(dataset, n_samples):
    # split into images and labels
    images, labels = dataset
    # choose random instances
    ix = randint(0, images.shape[0], n_samples)
    # select images and labels
    X, labels = images[ix], labels[ix]
    # generate class labels
```

Defining model for GAN.

Defining a function called load_real_samples() which expands the dimensions of trainX from 28x28 to 28x28x1. We also normalize the matrix trainX values.

```
# select real samples
def generate_real_samples(dataset, n_samples):
    # split into images and labels
    images, labels = dataset
    # choose random instances
    ix = randint(0, images.shape[0], n_samples)
    # select images and labels
    X, labels = images[ix], labels[ix]
    # generate class labels
    y = ones((n_samples, 1))
    return [X, labels], y

# generate points in latent space as input for the generator
def generate_latent_points(latent_dim, n_samples, n_classes=8):
    # generate points in the latent space
    x_input = randn(latent_dim * n_samples)
    # reshape into a batch of inputs for the network
    z_input = x_input.reshape(n_samples, latent_dim)
    # generate labels
    labels = randint(0, n_classes, n_samples)
    return [z_input, labels]

# use the generator to generate n fake examples, with class labels
def generate_fake_samples(generator, latent_dim, n_samples):
    # generate points in latent space
    z_input, labels_input = generate_latent_points(latent_dim, n_samples)
    # predict outputs
    images = generator.predict([z_input, labels_input])
    # create class labels
    y = zeros((n_samples, 1))
    return [images, labels_input], y
```

Defining a function called generate_real_samples() which returns n_samples (say, 32) number of random images picked from the training dataset along with their labels and an array y of length n_samples (32) of ones, representing that these images are real and not synthesized by the generator.

Defining a function named generate_latent_points() for generating input to the generator. Returns z_input, which is latent space input for the generator of the dimensions latent_dimxn_samples (say, 100x32); and labels, which is an array of length n_samples (say, 32) and consists of an integer between 0 and 7 (representing class – label) assigned randomly.

Defining a function generate_fake_samples() which returns generator output. It calls on the function generate_latent_points() and then passes its output as input to the generator model. The generator returns an array of synthesized images which contains n_samples (say, 32) number of images of the label corresponding to labels_input that was fed to the generator. We also create an array of zeroes y of length n_samples (say,32) and dimensionality 1 (representing fake/ synthesized images).

```
# generate samples and save as a plot and save the model
def summarize_performance(step, g_model, latent_dim, n_samples=32):
    # prepare fake examples
    [X, _], _ = generate_fake_samples(g_model, latent_dim, n_samples)
    # scale from [-1,1] to [0,1]
    X = (X + 1) / 2.0
    # plot images
    for i in range(32):
        # define subplot
        pyplot.subplot(4, 8, 1 + i)
        # turn off axis
        pyplot.axis('off')
        # plot raw pixel data
        pyplot.imshow(X[i, :, :, 0], cmap='gray_r')
    # save plot to file
    filename1 = 'Final_generated_plot_%04d.png' % (step+1)
    pyplot.savefig(filename1)
    pyplot.close()
    # save the generator model
    filename2 = 'Final_model_%04d.h5' % (step+1)
    g_model.save(filename2)
    print('>Saved: %s and %s' % (filename1, filename2))
```

Defining a function named summarize_performance() to save the generator model and some (n_samples= 32) generated images synthesized at the current state of the generator for some random class – labels.

```
# train the generator and discriminator
def train(g_model, d_model, gan_model, dataset, latent_dim, n_epochs=10000, n_batch=25):
    # calculate the number of batches per training epoch
    bat_per_epo = int(dataset[0].shape[0] / n_batch)
    # calculate the number of training iterations
    n_steps = bat_per_epo * n_epochs
    # calculate the size of half a batch of samples
    half_batch = int(n_batch / 2)
    # manually enumerate epochs
    for i in range(n_steps):
        # get randomly selected 'real' samples
        [X_real, labels_real], y_real = generate_real_samples(dataset, half_batch)
        # update discriminator model weights
        _,d_r1,d_r2 = d_model.train_on_batch(X_real, [y_real, labels_real])
        # generate 'fake' examples
        [X_fake, labels_fake], y_fake = generate_fake_samples(g_model, latent_dim, half_batch)
        # update discriminator model weights
        _,d_f,d_f2 = d_model.train_on_batch(X_fake, [y_fake, labels_fake])
        # prepare points in latent space as input for the generator
        [z_input, z_labels] = generate_latent_points(latent_dim, n_batch)
        # create inverted labels for the fake samples
        y_gan = ones((n_batch, 1))
        # update the generator via the discriminator's error
        _,g_1,g_2 = gan_model.train_on_batch([z_input, z_labels], [y_gan, z_labels])
        # summarize loss on this batch
        print('>%d, dr[%.3f,%.3f], df[%.3f,%.3f], g[%.3f,%.3f]' % (i+1, d_r1,d_r2, d_f,d_f2, g_1,g_2))
        # evaluate the model performance every 'epoch'
        if (i+1) % (bat_per_epo * 100) == 0:
            summarize_performance(i, g_model, latent_dim)
```

Defining the train() function. We make a variable called bat_per_epo (batch per epoch). Its value in the current scenario is 5 as dataset[0].shape[0] =125 as we have total 125 images and n_batch is kept as 25. We define another variable n_steps representing the total number of steps in the training of the GAN model. Its value in the current scenario is 50,000 as bat_per_epo is 5 and n_epochs is kept at 10,000. Another variable called half_batch contains 12.

We iterate 0 to 49,999 through n_steps. In each iteration, we call train_on_batch() function for batch training of the discriminator (on both fake and real samples) and the GAN model; and also print out the losses.

After every 500 (bat_per_epo(=5)*100 =500) iterations, the summarize_performance() function is called to save the current state of the generator model.

```
# size of the latent space
latent_dim = 100
# create the discriminator
discriminator = define_discriminator()
# create the generator
generator = define_generator(latent_dim)
# create the gan
gan_model = define_gan(generator, discriminator)
# load image data
dataset = load_real_samples()
print("Done!")
# train model
train(generator, discriminator, gan_model, dataset, latent_dim)
```

Calling the functions to create the discriminator, generator and GAN models.

Loading the training dataset.

Calling the function train() to begin training.

```
Using TensorFlow backend.
WARNING:tensorflow:From C:\Users\PRAGYA\Downloads\Anaconda\lib\site-packages\tensorflow\python\ops\nn_impl.py:180: add_dispat
ch_support.<locals>.wrapper (from tensorflow.python.ops.array_ops) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
(125, 28, 28, 1) (125,)
Done!
```

```
>1, dr[0.850,2.897], df[1.858,2.231], g[0.718,3.125]
C:\Users\PRAGYA\Downloads\Anaconda\lib\site-packages\keras\engine\training.py:297: UserWarning: Discrepancy between trainable
weights and collected trainable weights, did you set `model.trainable` without calling `model.compile` after ?
  'Discrepancy between trainable weights and collected trainable'
>2, dr[0.413,2.616], df[1.745,3.692], g[0.830,3.434]
>3, dr[0.485,2.631], df[0.947,3.491], g[1.024,2.988]
>4, dr[0.953,2.184], df[1.090,2.638], g[1.033,3.329]
>5, dr[0.656,3.056], df[0.796,2.469], g[1.575,3.179]
>6, dr[1.223,2.502], df[1.118,2.941], g[1.255,2.445]
>7, dr[0.752,2.583], df[0.635,3.262], g[0.810,2.944]
>8, dr[0.991,2.597], df[0.543,2.651], g[1.256,2.715]
>9, dr[1.068,2.323], df[0.660,2.977], g[0.992,2.682]
>10, dr[0.619,1.929], df[0.728,3.166], g[0.937,2.832]
>11, dr[0.756,2.319], df[0.670,1.891], g[1.107,2.407]
```

```
>49983, dr[0.141,0.155], df[0.161,0.252], g[3.501,0.074]
>49984, dr[0.416,0.001], df[0.309,0.013], g[4.139,0.086]
>49985, dr[0.439,0.000], df[0.160,0.917], g[3.421,0.581]
>49986, dr[0.080,0.055], df[0.305,0.005], g[3.245,0.002]
>49987, dr[0.362,0.000], df[0.144,0.001], g[3.397,0.000]
>49988, dr[0.230,0.228], df[0.381,0.001], g[2.945,0.048]
>49989, dr[0.419,0.197], df[0.171,0.042], g[2.819,0.413]
>49990, dr[1.080,0.003], df[0.521,0.437], g[3.485,0.040]
>49991, dr[0.069,0.002], df[0.521,0.336], g[2.652,0.004]
>49992, dr[0.592,0.000], df[0.456,0.010], g[2.949,0.005]
>49993, dr[0.156,0.013], df[0.093,0.024], g[2.393,0.072]
>49994, dr[0.088,0.034], df[0.259,0.205], g[4.111,0.006]
>49995, dr[0.018,0.000], df[0.194,0.000], g[3.131,0.247]
>49996, dr[0.537,0.001], df[0.064,0.576], g[2.993,0.198]
>49997, dr[0.022,0.005], df[0.290,0.001], g[3.203,0.353]
>49998, dr[0.033,0.325], df[0.418,0.008], g[3.923,0.005]
>49999, dr[0.084,0.033], df[0.211,0.000], g[2.830,0.047]
>50000, dr[0.452,0.002], df[0.251,0.001], g[3.890,0.000]
>Saved: Final_generated_plot_50000.png and Final_model_50000.h5
```

Code for loading the saved generator model and generating images of specific category with it:-

```
In [*]: # example of loading the generator model and generating images
from math import sqrt
from numpy import asarray
from numpy.random import randn
from keras.models import load_model
from matplotlib import pyplot
# generate points in latent space as input for the generator
def generate_latent_points(latent_dim, n_samples, n_class):
    # generate points in the latent space
    x_input = randn(latent_dim * n_samples)
    # reshape into a batch of inputs for the network
    z_input = x_input.reshape(n_samples, latent_dim)
    # generate labels
    labels = asarray([n_class for _ in range(n_samples)])
    return [z_input, labels]

# create and save a plot of generated images
def save_plot(examples, n_examples):
    # plot images
    for i in range(n_examples):
        # define subplot
        pyplot.subplot(sqrt(n_examples), sqrt(n_examples), 1 + i)
        # turn off axis
        pyplot.axis('off')
        # plot raw pixel data
        pyplot.imshow(examples[i, :, :, 0], cmap='gray_r')
    pyplot.show()
```

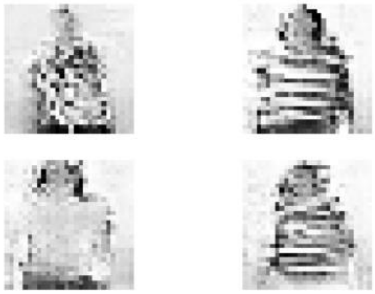
Importing necessary libraries and modules.

Defining a function named generate_latent_points() for generating input to the generator. Returns z_input, which is latent space input for the generator of the dimensions latent_dimxn_samples (say, 100x32); and labels, which is an array of length n_samples (say, 32) and consists of an integer between 0 and 7 (representing class – label) passed to the function.

Defining a function known as save_plot() for showing the generated/ synthesized images and arranging them in a square.

```
# load model
model = load_model('Final_model_50000.h5')
latent_dim = 100
n_examples = 4 # must be a square
n_class = 7 # WWSt
# generate images
latent_points, labels = generate_latent_points(latent_dim, n_examples, n_class)
# generate images
X  = model.predict([latent_points, labels])
# scale from [-1,1] to [0,1]
X = (1-X) / 2.0
# plot the result
save_plot(X, n_examples)
```

Loading the last saved generator model and generating images of a specific category (in this case 7, that is Women-Tshirt-White-horizontal Striped) and plotting those images.



Chapter -5

RESULTS

On successful compilation of the code, the following results were obtained for each of the categories:-

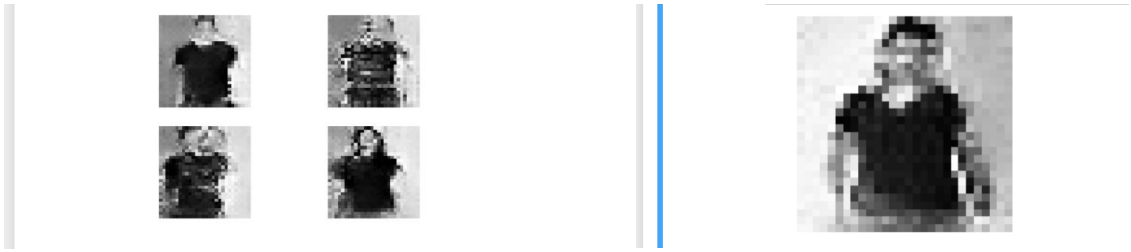


Fig 5.1: Men-Tshirt-Black –Solid

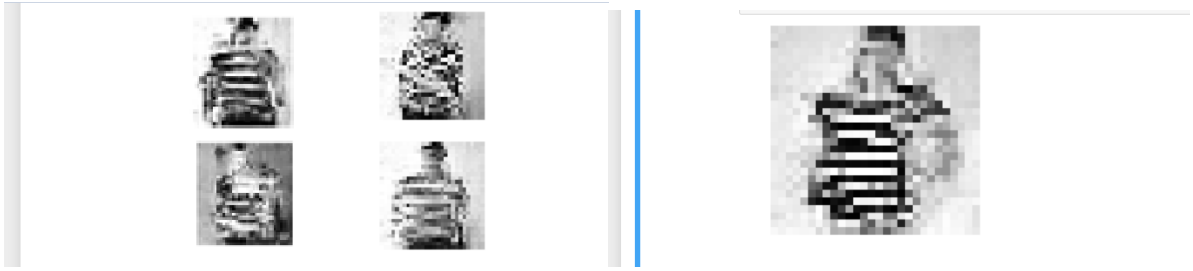


Fig 5.2: Men-Tshirt-Black –Striped

Fig 5.3: Men-Tshirt-White-Solid

Fig 5.4: Men-Tshirt-White-Striped

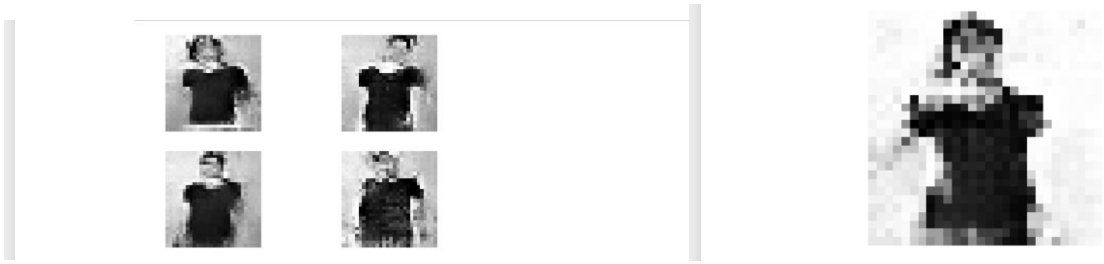


Fig 5.5: Women-Tshirt-Black-Solid

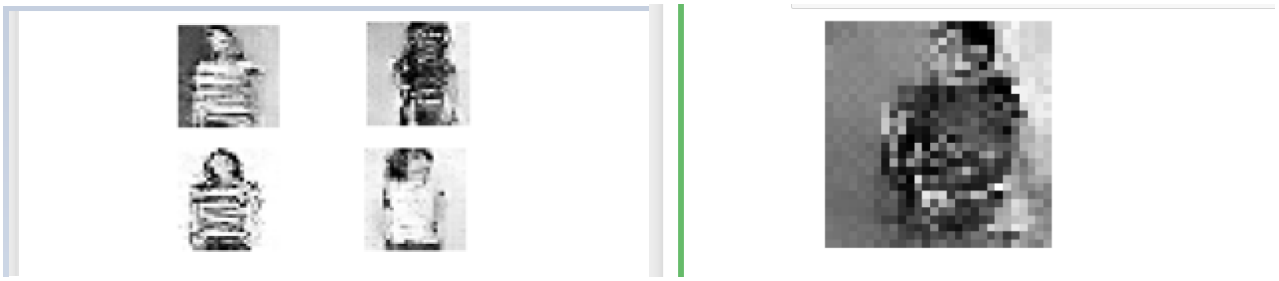


Fig 5.6: Women-Tshirt-Black-Striped

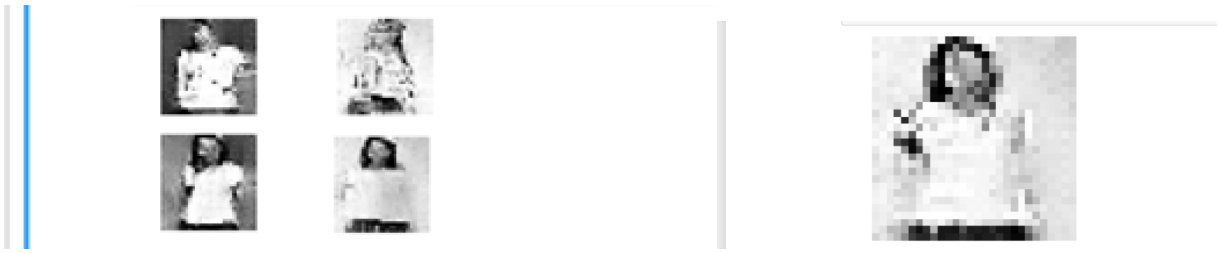


Fig 5.7: Women-Tshirt-White-Solid

Fig 5.8: Women-Tshirt-White-Striped

Future Work

- Project may be extended for RGB images.
- Project may be extended for larger and high- resolution images.
- Project may be made to handle more features like neck type, sleeve length, colour of stripes, etc.
- Project may be extended to handle more categories like Vertical stripes, blue and red colours.
- The project may also be extended to other items of clothing like shirts, pants, shoes, etc.
- The project may be altered to handle ambiguity in some attributes.

To handle these extensions, we would need to build a much larger categorical dataset. Right now, the dataset used has only 125 images distributed into 8 categories as explained in the above section. To improve results, and to take care of some of the above useful extensions to the project, we would need a dataset of the order of 100,000 images.

Chapter -6

CONCLUSION

We learnt about the concepts of Artificial Intelligence (AI), Machine Learning (ML) and Deep Learning (DL). We learnt the basic terminology in Machine Learning and also looked at various algorithms for the 2 most popular ML Tasks: Regression & Classification. Further, we understood how Deep Learning models work. Deep Networks are inspired by the structure of the human brain and are particularly effective in Feature Detection.

We understood and built basic neural networks with TensorFlow library. TensorFlow is an end-to-end open source platform for ML.

We started with building and analysing neural networks for some basic regression and classification problems using TensorFlow and Google-Colaboratory for example, Housing-prices problem. Furthermore, we did detailed study of classification models through building neural networks for Image classifiers such as Clothing-item recognition, Handwriting recognition, Happy-Sad classifier, Horse-Human classifier and Cat-Dog classifier, etc.

We took this a little further by using Convolutions that spot features in an image, and then classify and learn based on those features. We went a little deeper into Convolutions, learning how we can go into depth with real-world images and worked with image filtering.

In Convolutional Neural Networks, we explored how to use them with large datasets, take advantage of Augmentation, Dropouts, Regularization and Transfer learning, and of course looking at the coding considerations between binary or multi-class classification.

We analysed and understood a multi-class image classifier which identified whether the image was a Rock / Paper / Scissor pose.

Further, we were introduced to Generative Adversarial Networks and learnt their concept and working. We looked at various types of GANs and understood how to build them. We also learnt their individual advantages and use- cases. Now, we have aimed to apply our knowledge of GANs to real-world scenarios in the fashion industry. We have successfully developed a software which can

assist fashion enthusiasts and designers to communicate their ideas quickly and effectively by translating the text description of their imagination to images.

REFERENCES

[1] goodfellow2014generative, title= {Generative Adversarial Networks}, author={Ian J. Goodfellow and Jean Pouget-Abadie and Mehdi Mirza and Bing Xu and David Warde-Farley and Sherjil Ozair and Aaron Courville and Yoshua Bengio}, year={2014}, eprint={1406.2661}, archivePrefix={arXiv}, primaryClass={stat.ML}

[2] LeCun, Yann. "RL Seminar: The Next Frontier in AI: Unsupervised Learning".

[3] mirza2014conditional, title= {Conditional Generative Adversarial Nets}, author= {Mehdi Mirza and Simon Osindero}, year={2014}, eprint={1411.1784}, archivePrefix={arXiv}, primaryClass={cs.LG}

[4] radford2015unsupervised, title= {Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks}, author= {Alec Radford and Luke Metz and Soumith Chintala}, year= {2015}, eprint= {1511.06434}, archivePrefix={arXiv}, primaryClass={cs.LG}

[5] zhang2016stackgan, title={StackGAN: Text to Photo-realistic Image Synthesis with Stacked Generative Adversarial Networks}, author={Han Zhang and Tao Xu and Hongsheng Li and Shaoting Zhang and Xiaogang Wang and Xiaolei Huang and Dimitris Metaxas}, year={2016}, eprint={1612.03242}, archivePrefix={arXiv}, primaryClass={cs.CV}

[6] Kalantidis, Y., Kennedy, L., and Li, L.-J. Getting the look: clothing recognition and segmentation for automatic product suggestions in everyday photos. In Proceedings of the 3rd ACM conference on International conference on multimedia retrieval, pp. 105–112. ACM, 2013.

[7] Simo-Serra, E., Fidler, S., Moreno-Noguer, F., and Urtasun, R. Neuroaesthetics in fashion: Modeling the perception of fashionability. In CVPR, volume 2, pp. 6, 2015.

[8] Han, X., Wu, Z., Wu, Z., Yu, R., and Davis, L. S. Viton: An image-based virtual try-on network. arXiv preprint arXiv:1711.08447, 2017.

[9] Liang, X., Lin, L., Yang, W., Luo, P., Huang, J., and Yan, S. Clothes co-parsing via joint image segmentation and labeling with application to clothing retrieval. IEEE Transactions on Multimedia, 18(6):1175–1186, 2016.

[10] Reed, S., Akata, Z., Yan, X., Logeswaran, L., Schiele, B., and Lee, H. Generative adversarial text to image synthesis. arXiv preprint arXiv:1605.05396, 2016a

[11] Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollar, P., and Zitnick, C. L. Microsoft coco: ´ Common objects in context. In European conference on computer vision, pp. 740–755. Springer, 2014.

[12] Nilsback, M.-E. and Zisserman, A. Automated flower classification over a large number of classes. In Computer Vision, Graphics & Image Processing, 2008. ICVGIP'08. Sixth Indian Conference on, pp. 722–729. IEEE, 2008.

[13] Liu, Z., Luo, P., Wang, X., and Tang, X. Deep learning face attributes in the wild. In Proceedings of the IEEE International Conference on Computer Vision, pp. 3730– 3738, 2015.

[14] Liu, Z., Luo, P., Qiu, S., Wang, X., and Tang, X. Deepfashion: Powering robust clothes recognition and retrieval with rich annotations. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 1096–1104, 2016.

[15] odena2016conditional, title={Conditional Image Synthesis With Auxiliary Classifier GANs}, author={Augustus Odena and Christopher Olah and Jonathon Shlens}, year={2016}, eprint={1610.09585}, archivePrefix={arXiv}, primaryClass={stat.ML}

[16] https://www.coursera.org/learn/introduction-tensorflow/

[17] https://www.coursera.org/learn/convolutional-neural-networks-tensorflow/

[18] https://keras.io/

[19] https://www.tensorflow.org/

[20] https://keras.io/preprocessing/image/

[21] https://www.greatlearning.in/

RESUME

PRAGYA GOYAL

(+91) 9971971308
prggoyal9@gmail.com
www.linkedin.com/in/goyal-pragya/

Highly-motivated B. Tech student (current CGPA: 8.6) seeking an opportunity to leverage my skills while helping the company with the upcoming challenges with my fast-learning and time-management capabilities.

SPECIAL COURSES

Advanced Radio Access Network (ARAN)
by Ericsson JANUARY 2019-MAY 2019
The course started with an introduction to Radio Technology and the Evolution of radio networks. I understood the basics of RF, signal processing and antenna theory. Then I learnt about LTE and how it works. Lastly, I looked and understood how companies like Ericsson handle, optimize and test the Radio Access Network.

Intro to TensorFlow for Artificial Intelligence, Machine Learning & Deep Learning
by deeplearning.ai SEPTEMBER 2019
TensorFlow is popular open-source framework for machine learning. I learnt how to build a basic neural network in TensorFlow and train it for computer vision applications. I also understood how to use convolutions (CNN) to improve neural networks.

Data Science: R Basics
by HarvardX AUGUST 2019
Data Science: R Basics is an introductory course to R by Harvard University on edX. It introduced me to data wrangling, data analysis and visualization of data through R.

PROJECTS

Terrain Explorer with Smart Base Station MARCH 2019 - MAY 2019

Developed an Arduino-operated car that was controlled via the Blynk app. The car transmitted values of various sensors via Wi-Fi Module to the Blynk app as well as Node-RED software. There, the sensor values were plotted and analyzed. For debugging and retrospection purposes, the real-time data was also saved to a database along with the date-time stamp.

Software Used: Arduino IDE, Node-RED, Blynk App

Quadcopter with Air Quality Analysis JANUARY 2019 - APRIL 2019

Assembled and programmed a QC using Arduino UNO. It was responsible for measuring air quality using sensors and sending the corresponding data via the Blynk cloud server to Node-RED. There, the real-time data was interpreted and stored in the database along with the date and time stamp.

Software Used: Arduino IDE, Node-RED, Blynk App

PCB Fabrication of Arm board- Atmega 8 & Atmega 16 AVR Microcontroller ICs SEPTEMBER 2018 - OCTOBER 2018

Fabricated Arm Board for Atmega 8 & Atmega 16 AVR MC ICs in the workshop organized by the CICE Hub, JIIT.

Software Used: Eagle 7.6.0 Layout Editor

SKILLS

Languages: C/C++, Python, SQL, R

Tools/Frameworks: Tableau, MySQL, MATLAB, Node-RED, P-Spice, Eagle, Scratch, Google Colaboratory, MS Office

INTERNSHIP

Bharat Heavy Electricals Limited (BHEL), Noida
Project: Auxiliary PRDS System
(JUNE - JULY 2019)

Ramped up on the processes and overall schematics of a rudimentary power plant. Inspected piping and instrumentation diagrams of various systems involved in a power plant. Investigated the control and application of Auxiliary PRDS System in a power plant.

EDUCATION

B. Tech (ECE), JIIT Noida
CGPA: 8.6/10 (till 7th semester)

Class 12th, DPS Indirapuram
Percentage: 90% (2016)

Class 10th, DPS Indirapuram
CGPA: 10/10 (2014)

ACHIEVEMENTS

1st in Conventional Rangoli Making competition organized by Kalakriti Hub of JIIT on IMPRESSIONS 2018

Obtained percentile scores of 98.5% in Language Conventions, 99.65% in Qualitative Reasoning and 84.2% in Quantitative Reasoning in Problem Solving Assessment (PSA) conducted by CBSE in Class 11th (2014)

Attained award for proficiency in academics for six consecutive years (2009 - 2014)

Achieved DISTINCTION in Mathematics International Educational Assessment conducted by UNSW Global, Australia in Class 10th (2013)