Bridge the Gap

Title: A case study on Artificial Intelligence and its application

Aim: To study AI and its applications

Objective:

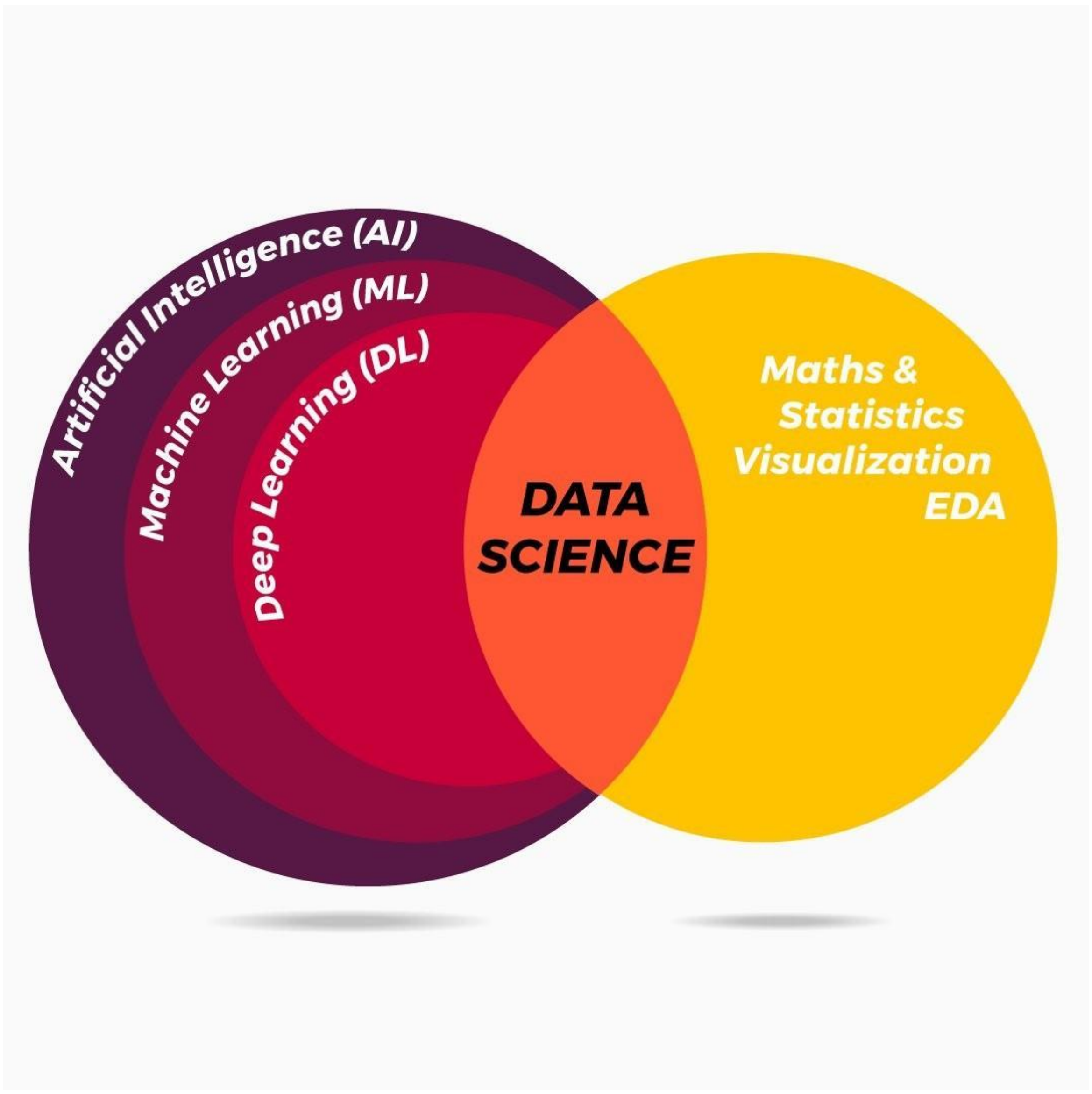
a. Understand what AI is

b. Study the use cases of AI

c. Study the real-world implementation of AI

Artificial Intelligence: -

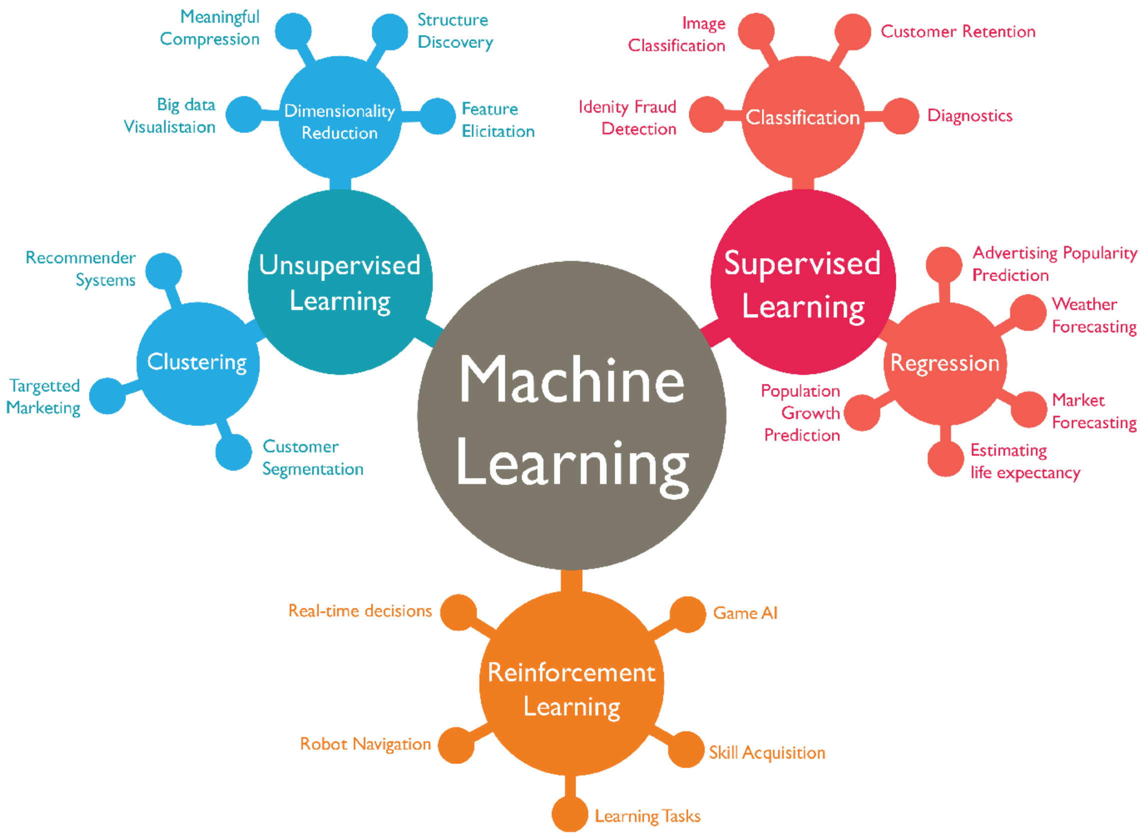
Computers are superfast in calculation or data analysis, but they don't have intelligence like human beings. Computer only gives result, if they have data. If any data is missing, they are unable to give output. So, we give intelligence to computer. So, they can predict future data by historical data. This is called Artificial Intelligence.



By the help of Artificial Intelligence, AI systems will typically demonstrate at least some of the following behaviors associated with human intelligence: planning, learning, reasoning, problem solving, knowledge representation, perception, motion, and manipulation and, to a lesser extent, social intelligence and creativity.

Machine Learning: -

Machine Learning Machine Learning is the field of study that gives computers the capability to learn without being explicitly programmed. ML is one of the most exciting technologies that one would have ever come across. As it is evident from the name, it gives the computer that makes it more similar to humans: The ability to learn.



Machine learning is where a computer system is fed large amounts of data, which it then uses to learn how to carry out a specific task, such as understanding speech or captioning a photograph.

How Companies use AI/ML

1.Coca-Cola (Soft drinks):-

Everyday people consume 1.9 billion servings of Coca Cola drinks. Due to this huge market share in the beverage space, Coca Cola generates a lot of data that it uses to make strategic decisions. Coca Cola was the earliest non-IT company to adopt AI and Big Data.

Coca Cola is known for investing heavily in research and development. After having a successful launch of self-service soft drinks and fountains, Coca Cola gathered all this data. It then combined it with AI and using the insights acquired it launched a new beverage brand—Cherry Sprite. Cherry Sprite was based on the data collected by clients mixing their drinks to create the perfect cocktail beverage.

2. McDonald's

McDonald's, one of the biggest fast-food chains on the planet, is one company that needs no introduction. For a long time, the fast-food chain did not see the need to implement AI and Big Data but upon seeing the success that their competitors were having, they revised their strategy. As the largest fast-food chain in the world, serving close to 70 million people daily, it's clear that it generates a lot of data. Moreover, they have leveraged that data in many ways as shown below: -

1. Use of digital menus: - Digital menus are a common phenomenon these days, but McDonald's has taken it a notch higher by introducing digital menus that change based on real-time data analysis using AI. The menus vary also based on parameters like weather and time of day. Due to this innovation, they recorded a 3% increase in sales in Canada.
2. Customer Experience: - McDonalds are leveraging the application to make it a win-win situation for them and their clients. Users of their app get various benefits like:

- Exclusive deals on the app.
- Avoiding long queues at the counter and drive-thru

On the other hand, when customers pay through the application, McDonald's not only got money but also vital customer data that includes metrics like:

- Where and when the client goes to the restaurant
- The frequency of their visits
- Preference between a drive-thru or a restaurant
- By using this data, McDonald's can make recommendations and promote deals to increase sales. As a result of this data, they have noticed a more than 30% increase in sales in Japan for clients that used the app.

3. Autonomous ships

Most people know that the future lies in autonomous automobiles, but not many people know that there are also plans to launch autonomous ships. This is due to a collaboration between Google and Rolls-Royce to create autonomous and smart ships. Rolls-Royce will be using the Machine Learning Engine on Google Cloud in its applications to make its vision of smarter and autonomous ships come true.

At first, AI algorithms will be trained using machine learning to identify objects that can be encountered at sea and classify them based on hazard they may pose. The Machine learning algorithms that are currently being used by Google Voice and image search applications. They will also be augmented by massive data sets produced from various devices like sensors, cameras, and cameras on vessels. By combining the cloud-based AI and Big Data application enable data to be shared in real-time to any ship and also to on-shore control centers.

4. Healthcare

- One of the issues that many healthcare systems face is matching staffing volumes to patient numbers. At one time you can have very few patients and a huge staff roster. During other times, you have an overflow of patients and a strained workforce. So how do you solve this problem?

- Embrace AI and Big Data by following the example set by some hospitals in Paris. Four hospitals in Paris have managed to leverage AI and Big Data to enable nurses, doctors, and hospital administrators to forecast admission and visit rates for two weeks. This enables them to draft in extra staff when they expect high patient volumes leading to reduced wait times and better-quality care. So how does the system work?

- Using an open-source AI Analytics platform, the hospitals compiled admission data for the last ten years and external data sets like flu patterns, weather and public holidays. The insights were then used to predict admission rates at different times. Apart from just being used to predict admission rates, such data can be used to reduce wastage and enhance healthcare delivery by forecasting the demand for services.

5. Security

When it comes to security screening, most of us expect that you will find a security person screening you individually using a face-to-face approach. Although customs and immigration officers are highly trained to detect someone that is lying about their intention’s mistakes do still happen. Also, there is the fact that humans get tired and can be distracted leading to errors. So how do we avoid such human errors? Apply AI and Big Data to screen passengers.

Homeland security has developed a new system called AVATAR that screens people's facial expressions and body gestures and picks up small variations that may raise suspicion. The system has a screen with a virtual face that asks the passenger questions.

It monitors the person's responses for changes in voice tone as well as differences in what was said. Data collected is compared to a database and compared against factors that show someone might be lying. If the passenger is flagged as being suspicious, then they are highlighted for further inspection.

Conclusion:

Studied various real-world applications of Artificial Intelligence.

ASSIGNMENT NO.2

Title: Study of Deep learning Packages: TensorFlow, Keras, Theano and PyTorch. Document the distinct features and functionality of the packages.

Aim: Study and installation of following Deep learning Packages:

I. Tensor Flow
II. Keras
III. Theano
IV. PyTorch

Theory:

Deep Learning

Deep learning is a machine learning technique that teaches computers to do what comes naturally to humans: learn by example. Deep learning is a key technology behind driverless cars, enabling them to recognize a stop sign, or to distinguish a pedestrian from a lamppost. It is the key to voice control in consumer devices like phones, tablets, TVs, and hands-free speakers. Deep learning is getting lots of attention lately and for good reason. It's achieving results that were not possible before.

In deep learning, a computer model learns to perform classification tasks directly from images, text, or sound. Deep learning models can achieve state-of-the-art accuracy, sometimes exceeding human-level performance. Models are trained by using a large set of labeled data and neural network architectures that contain many layers.

Various packages in python for supporting Machine Learning libraries and which are mainly used for Deep Learning

i. NumPy
ii. Scikit-learn.
iii. Pandas.
iv. TensorFlow.
v. Seaborn.
vi. Theano.
vii. Keras.
viii. PyTorch.

Comparison between TensorFlow/Keras/Theano and PyTorch on following points(make table) :

I. Available Functionality
II. GUI status
III. Versions.
IV. Features
V. Compatibility with other environments.
VI. Specific Application domains.

Keras	**PyTorch**	**TensorFlow**	**Theano**
API Level	High	Low	High and Low
Architecture	Simple, concise, readable	Complex, less readable	Not easy to use
Datasets	Smaller datasets	Large datasets, high performance	Large datasets, high performance
Debugging	Simple network, so debugging is not often needed	Good debugging capabilities	Difficult to conduct debugging
Does It Have Trained Models?	Yes	Yes	Yes
Popularity	Most popular	Third most popular	Second most popular
Speed	Slow, low performance	Fast, high-performance	Fast, high-performance
Written In	Python	Lua	C++, CUDA

List of Models Datasets and pretrained models, Libraries and Extensions , Tools related to TensorFlow also discuss any two case studies like (PayPal, Intel, Etc.) related to TensorFlow.

KerasTuner

KerasTuner is an easy-to-use, scalable hyperparameter optimization framework that solves the pain points of hyperparameter search. Easily configure your search space with a define-by-run syntax, then leverage one of the available search algorithms to find the best hyperparameter values for your models. KerasTuner comes with Bayesian Optimization, Hyperband, and Random Search algorithms built-in, and is also designed to be easy for researchers to extend in order to experiment with new search algorithms.

KerasNLP

KerasNLP is a simple and powerful API for building Natural Language Processing (NLP) models. KerasNLP provides modular building blocks following standard Keras interfaces (layers, metrics) that allow you to quickly and flexibly iterate on your task. Engineers working in applied NLP can leverage the library to assemble training and inference pipelines that are both state-of-the-art and production-grade. KerasNLP is maintained directly by the Keras team.

AutoKeras

AutoKeras is an AutoML system based on Keras. It is developed by DATA Lab at Texas A&M University. The goal of AutoKeras is to make machine learning accessible for everyone. It provides high-level end-to-end APIs such as ImageClassifier for TextClassifier to solve machine learning problems in a few lines, as well as flexible building blocks to perform architecture search.

Model optimization toolkit

The TensorFlow Model Optimization Toolkit is a set of utilities to make your inference models faster, more memory-efficient, and more power-efficient, by performing post-training weight quantization and pruning-aware training. It has native support for Keras models, and its pruning API is built directly on top of the Keras API.

Sequential Model in Keras

It allows us to create models layer by layer in sequential order. But it does not allow us to create models that have multiple inputs or outputs. It is best for a simple stack of layers which have 1 input tensor and 1 output tensor.This model is not suited when any of the layers in the stack has multiple inputs or outputs. Even if we want non-linear topology, it is not suited.

Parameter Optimization

Parameter optimization is used to identify optimal settings for the inputs that you can control. Engage searches a range of values for each input to find settings that meet the defined objective and lead to better performance of the system.

Theano

Theano is a Python library that allows us to evaluate mathematical operations including multi-dimensional arrays so efficiently. It is mostly used in building Deep Learning Projects. It works a way faster on the Graphics Processing Unit (GPU) rather than on CPU. Theano attains high speeds that gives a tough competition to C implementations for problems involving large amounts of data. It can take advantage of GPUs which makes it perform better than C on a CPU by considerable orders of magnitude under some certain circumstances.
It knows how to take structures and convert them into very efficient code that uses NumPy and some native libraries. It is mainly designed to handle the types of computation required for large neural network algorithms used in Deep Learning. That is why it is a very popular library in the field of Deep Learning.

```
# Python program showing
# addition of two scalars

# Addition of two scalars
import numpy
import theano.tensor as T
from theano import function

# Declaring two variables
x = T.dscalar('x')
y = T.dscalar('y')

# Summing up the two numbers
z = x + y

# Converting it to a callable object
# so that it takes matrix as parameters

f = function([x, y], z)
f(5, 7)
Output: array(12.0)
```

PyTorch Tensors

A Pytorch Tensor is basically the same as a NumPy array. This means it does not know anything about deep learning or computational graphs or gradients and is just a generic n-dimensional array to be used for arbitrary numeric computation. However, the biggest difference between a NumPy array and a PyTorch Tensor is that a PyTorch Tensor can run on either CPU or GPU. To run operations on the GPU, just cast the Tensor to a cuda data type using:

Uber's Pyro

Pyro enables flexible and expressive deep probabilistic modeling, unifying the best of modern deep learning and Bayesian modeling. It was designed with these key principles: Universal: Pyro can represent any computable probability distribution. Scalable: Pyro scales to large data sets with little overhead

Tesla Autopilot

The raw images of the car camera are processed by a residual neural network (RegNet) that extracts multiple blocks or feature layers in width (W) x height (H) x channel (C).

The first feature output has a high resolution (160 x 120) and focuses on all the details in the image. Moving all the way to the top layer, with a low resolution (20 x 15) but with a greater channel count (512). Intuitively, you could visualize each channel as a different filter that activates certain parts of the feature map or image, for example, one channel puts more emphasis on edges and another on smoothing parts out. Therefore, the top layer puts more focus on the context by using a variety of channels. Whereas, the lower layer focuses more on the details and specifics by using a higher resolution.

The multiple feature layers interact with each other via a **BiFPN**, a weighted bi-directional feature pyramid. For example, the first detailed layer thinks it sees a car but is not sure. At the top of the feature pyramid, the context layer provides feedback that the object is at the end of the road. So yes it is probably a car, but it is blurred out due to the distance.

- **Steps/ Algorithm**

Installation of TensorFlow On Ubuntu:

1. Install the Python Development Environment:

You need to download Python, the PIP package, and a virtual environment. If these packages are already installed, you can skip this step.

You can download and install what is needed by visiting the following links: https://www.python.org/

https://pip.pypa.io/en/stable/installing/ https://docs.python.org/3/library/venv.ml

To install these packages, run the following commands in the terminal: sudo apt update

sudo apt install python3-dev python3-pip python3-venv

2. Create a Virtual Environment Navigate to the directory where you want to store your Python 3.0 virtual environment. It can be in your home directory, or any other directory where your user can read and write permissions.

mkdir tensorflow_files cd tensorflow_files

Now, you are inside the directory. Run the following command to create a virtual environment: python3 -m venv

virtualenv

The command above creates a directory named virtualenv. It contains a copy of the Python binary, the PIP package manager, the standard Python library, and other supporting files.

3. **Activate the Virtual Environment**

source virtualenv/bin/activate

Once the environment is activated, the virtual environment's bin directory will be added to the beginning of the $PATH variable. Your shell's prompt will alter, and it will show the name of the virtual environment you are currently using, i.e. virtualenv.

4. Update PIP

pip install --upgrade pip

5. Install TensorFlow

The virtual environment is activated, and it's up and running. Now, it's time to install the TensorFlow package. pip install -- upgrade TensorFlow

- **Installation of Keras on Ubuntu :**

Prerequisite: Python version 3.5 or above.

STEP 1: Install and Update Python3 and Pip

Skip this step if you already have Python3 and Pip on your machine**.**

sudo apt install python3 python3.pip

sudo pip3 install —upgrade pip

STEP 2: Upgrade Setup tools pip3 install —upgrade setup tools

STEP 3: Install TensorFlow pip3 install TensorFlow

Verify the installation was successful by checking the software package information: pip3 show TensorFlow
STEP 4: Install Keras pip3 install Keras

Verify the installation by displaying the package information:

pip3 show Keras

- **Installation of Theano on Ubuntu:**

Step 1: First of all, we will install Python3 on our Linux Machine. Use the following command in the terminal to install Python3.
sudo apt-get install python3

Step 2: Now, install the pip module

sudo apt install python3-pip

Step 3: Now, install the Theano
Verifying Theano package Installation on Linux using PIP python3 -m pip show theano

Installation of PyTorch

First, check if you are using python's latest version or not.Because PyGame requires python
3.7 or a higher version python3 –version

pip3 –version
pip3 install torch==1.8.1+cpu torchvision==0.9.1+cpu torchaudio==0.8.1 -f https://download.pytorch.org/whl/torch_stable.html

First, check if you are using python's latest version or not.Because PyGame requires python
3.7 or a higher version python3 –version

pip3 –version
pip3 install torch==1.8.1+cpu torchvision==0.9.1+cpu torchaudio==0.8.1 -f https://download.pytorch.org/whl/torch_stable.html

Python Libraries and functions required

1. Tensorflow,keras

NumPy : NumPy is a Python library used for working with arrays. It also has functions for working in the domain of linear algebra, Fourier transform, and matrices. NumPy stands for Numerical Python. To import NumPy use

import NumPy as np
pandas: pandas is a fast, powerful, flexible and easy to use open source data analysis and manipulation tool, built on top of the Python programming language. To import pandas use

import pandas as pd

sklearn : Scikit-learn (Sklearn) is the most useful and robust library for machine learning in
Python. It provides a selection of efficient tools for machine learning and statistical modeling including classification, regression, clustering and dimensionality reduction via a consistence interface in Python. This library, which is largely written in Python, is built upon NumPy,

SciPy and Matplotlib. For importing train_test_ split use

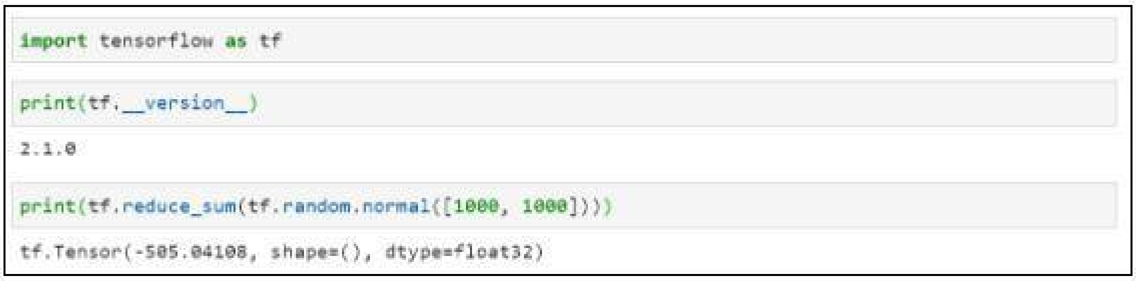
from sklearn.model_selection import train_test_split

2. For Theano Requirements:

- · Python3
- · Python3-pip
- · NumPy
- · SciPy
- · BLAS

Sample Code with comments

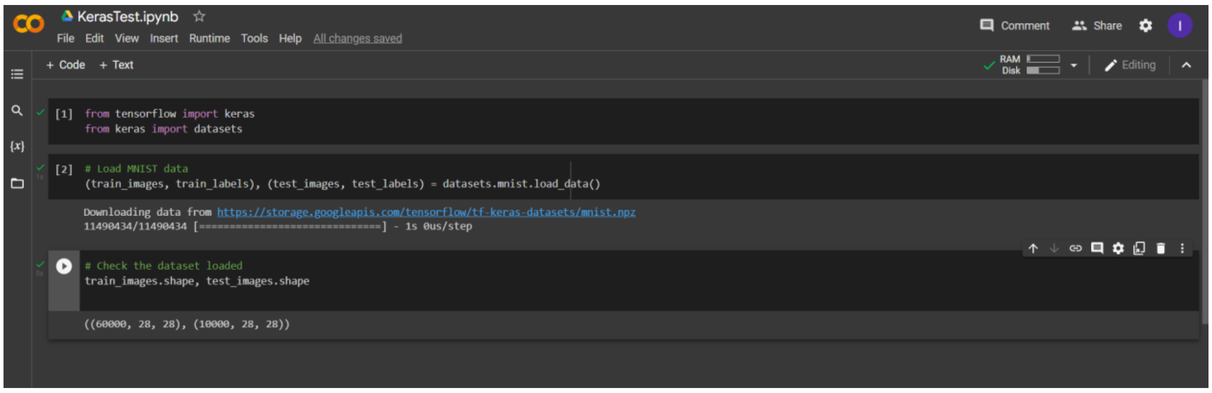
1. TensorFlow Test program:



```
import tensorflow as tf

print(tf.__version__)

2.1.0

print(tf.reduce_sum(tf.random.normal([1000, 1000])))

tf.Tensor(-505.04108, shape=(), dtype=float32)
```

2. Keras Test Program:

from tensorflow import keras
from keras **import** datasets
Load MNIST data
#(train_images, train_labels), (test_images, test_labels) = datasets.mnist.load_data()
Check the dataset loaded # train_images.shape, test_images.shape



```
[1] from tensorflow import keras
    from keras import datasets

[2] # Load MNIST data
    (train_images, train_labels), (test_images, test_labels) = datasets.mnist.load_data()

    Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
    11490434/11490434 [==============================] - 1s 0us/step

    # Check the dataset loaded
    train_images.shape, test_images.shape

    ((60000, 28, 28), (10000, 28, 28))
```

3. Theano test program

import theano
from theano import tensor

# Declaring variables
a = tensor.dscalar()
b = tensor.dscalar()

# Subtracting
res = a + b

# Converting it to a callable object
# so that it takes matrix as parameters
func = theano.function([a, b], res)

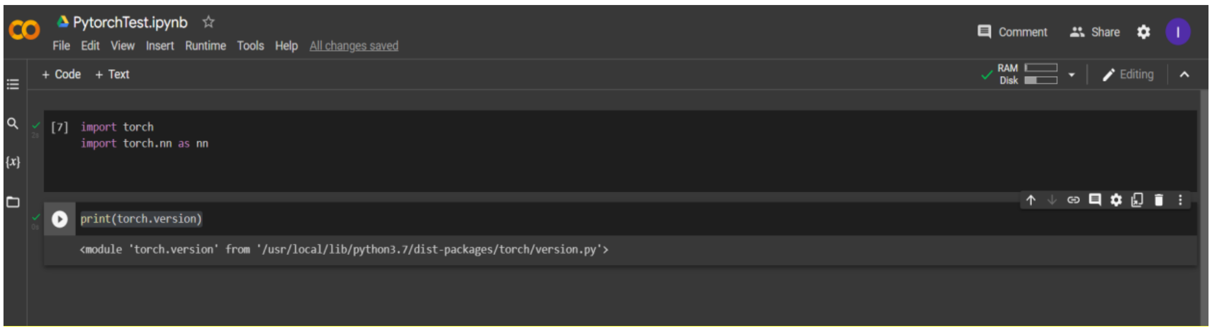
# Calling function
#assert 20.0 == func(30.5, 10.5)

fun = theano.function([x, y], z)
fun(8, 7)

4. Test program for PyTorch

## The usual imports import torch
import torch.nn as nn

## print out the pytorch version used print(torch._version_)



```
[7] import torch
    import torch.nn as nn
```

```
print(torch.version)
```

```
<module 'torch.version' from '/usr/local/lib/python3.7/dist-packages/torch/version.py'>
```

Conclusion:

TensorFlow, PyTorch, Keras and Theano all these packages are installed and ready for Deep learning applications. As per application domain and dataset we can choose the appropriate package and build the required type of Neural Network.

FAQ:

1)What is Deep learning?

2) What are various packages in python for supporting Machine Learning libraries and which are mainly used for Deep Learning?

3) Compare TensorFlow / Keras/Theano and PyTorch on following points (make a table)

i. Available Functionality

ii. GUI status

iii. Versions.

iv. Features

v. Compatibility with other environments.

vi. Specific Application domains

4) Enlist the Models Datasets and pretrained models, Libraries and Extensions, Tools related to TensorFlow also discuss any two case studies like (PayPal, Intel, Etc.) related to TensorFlow.

5) Explain the Keras Ecosystem. (Keras tuner, kerasNLP,keras,Autokaras and Model Optimization.) Also explain following concepts related to Keras: 1. Developing sequential Model 2. Training and validation using the inbuilt functions 3. Parameter Optimization.

6) Explain a simple Theano program.

7) Explain PyTorch Tensors. And explain Uber's Pyro, Tesla Autopilot

ASSIGNMENT NO.3

Title: Implementing Feedforward neural networks

Aim: Implementing Feedforward neural networks with Keras and TensorFlow

a. Import the necessary packages

b. Load the training and testing data (MNIST/CIFAR10)

c. Define the network architecture using Keras

d. Train the model using SGD

e. Evaluate the network

f. Plot the training loss and accuracy

Theory:

Feedforward Neural Network

A Feed Forward Neural Network is an artificial neural network in which the connections between nodes do not form a cycle. The opposite of a feed forward neural network is a recurrent neural network, in which certain pathways are cycled. The feed forward model is the simplest form of neural network as information is only processed in one direction. While the data may pass through multiple hidden nodes, it always moves in one direction and never backwards.

Working of Feedforward Neural Network

A Feed Forward Neural Network is commonly seen in its simplest form as a single layer perceptron. In this model, a series of inputs enter the layer and are multiplied by the weights. Each value is then added together to get a sum of the weighted input values. If the sum of the values is above a specific threshold, usually set at zero, the value produced is often 1, whereas if the sum falls below the threshold, the output value is -1. The single layer perceptron is an important model of feed forward neural networks and is often used in classification tasks. Furthermore, single layer perceptron's can incorporate aspects of machine learning. Using a property known as the delta rule, the neural network can compare the outputs of its nodes with the intended values, thus allowing the network to adjust its weights through training in order to produce more accurate output values. This process of training and learning produces a form of a gradient descent. In multi-layered perceptron's, the process of updating weights is nearly analogous, however the process is defined more specifically as back-propagation. In such cases, each hidden layer within the network is adjusted according to the output values produced by the final layer.

Real time scenarios of Feedforward Neural Network

1. Physiological feedforward system: during this, the feedforward management is epitomized by the conventional prevenient regulation of heartbeat prior to work out by the central involuntary

2. Gene regulation and feedforward: during this, a motif preponderantly seems altogether the illustrious networks and this motif has been shown to be a feedforward system for the detection of the non-temporary modification of atmosphere.

3. Automation and machine management: feedforward control may be discipline among the sphere of automation controls utilized in

4. Parallel feedforward compensation with derivative: This a rather new technique that changes the part of AN open-loop transfer operation of a non-minimum part system into the minimum part.

Components of Feedforward Neural Network.

The following are the components of a feedforward neural network:

Layer of input: It contains the neurons that receive input. The data is subsequently passed on to the next tier. The input layer's total number of neurons is equal to the number of variables in the dataset.

Hidden layer: This is the intermediate layer, which is concealed between the input and output layers. This layer has many neurons that perform alterations on the inputs. They then communicate with the output layer.

Output layer: It is the last layer and is depending on the model's construction. Additionally, the output layer is the expected feature, as you are aware of the desired outcome.

Neuron weights: Weights are used to describe the strength of a connection between neurons. The range of a weight's value is from 0 to 1.

Cost Function in Feedforward Neural Network

The cost function is an important factor of a feedforward neural network. Generally, minor adjustments to weights and biases have little effect on the categorized data points. Thus, to determine a method for improving performance by making minor adjustments to weights and biases using a smooth cost function.

Mean square error cost function.

The mean square error cost function is defined as follows:

$$C(w, b) \equiv \frac{1}{2n} \sum_x \|y(x) - a\|^2.$$

Where,

w = weights collected in the network

b = biases

n = number of training inputs

a = output vectors

x = input

‖v‖ = usual length of vector v

Loss function in Feedforward Neural Network

A neural network's loss function is used to identify if the learning process needs to be adjusted.
As many neurons as there are classes in the output layer. To show the difference between the predicted and actual distributions of probabilities.

Cross Entropy Loss

The cross-entropy loss for binary classification is as follows.

Cross Entropy Loss:

$$L(\Theta) = \begin{cases} -log(\hat{y}) & \text{if } y = 1 \\ -log(1 - \hat{y}) & \text{if } y = 0 \end{cases}$$

The cross-entropy loss associated with multi-class categorization is as follows:

Cross Entropy Loss:

$$L(\Theta) = -\sum_{i=1}^{k} y_i \log(\hat{y}_i)$$

Kernel Concept in Feedforward Neural Network

Kernels (weights) used to scale input and hidden node values. Each connection typically holds a different weight. Biases used to adjust scaled values before passing them through an activation function.

MNIST and CIFAR 10 Dataset.

MNIST: The MNIST dataset is an acronym that stands for the Modified National Institute of Standards and Technology dataset. It is a dataset of 60,000 small square 28×28-pixel grayscale images of handwritten single digits between 0 and 9.

CIFAR 10: A dataset of 50,000 32x32 color training images and 10,000 test images, labeled over 10 categories.

Use and parameter setting related to feedforward network implementation for libraries :

SKlearn :

i) LabelBinarizer (sklearn.preprocessing) : Class used to wrap the functionality of label_binarize and allow for fitting to classes independently of the transform operation.

Parameters:

y:*array-like:* Sequence of integer labels or multilabel data to encode.

classes:*array-like of shape (n_classes,):* Uniquely holds the label for each class.

neg_label:*int, default=0 :*Value with which negative labels must be encoded.

pos_label:*int, default=1:* Value with which positive labels must be encoded.

sparse_output:*bool, default=False:* Set to true if output binary array is desired in CSR sparse format.

ii) classification_report (sklearn.metrics) : Build a text report showing the main classification metrics.

Parameters:

y_true:*1d array-like, or label indicator array / sparse matrix*
Ground truth (correct) target values.

y_pred:*1d array-like, or label indicator array / sparse matrix*
Estimated targets as returned by a classifier.

labels:*array-like of shape (n_labels,), default=None*
Optional list of label indices to include in the report.

target_names:*list of str of shape (n_labels,), default=None*
Optional display names matching the labels (same order).

sample_weight:*array-like of shape (n_samples,), default=None*
Sample weights.

digits:*int, default=2*
Number of digits for formatting output floating point values. When output_dict is True, this will be ignored, and the returned values will not be rounded.

output_dict:*bool, default=False*
If True, return output as dict.

zero_division: ***"warn", 0 or 1, default="warn"***
Sets the value to return when there is a zero division. If set to "warn", this acts as 0, but warnings are also raised.

Tensorflow.keras:

1. **models :** Model groups layers into an object with training and inference features.

Parameters:

input: The input(s) of the model: a keras.Input object or list of keras.Input objects.

output: The output(s) of the model. See Functional API example below.

name: String, the name of the model.

2. **layers:** This is the class from which all layers inherit.

Parameters:

trainable: Boolean, whether the layer's variables should be trainable.

name: String name of the layer.

dtype: The dtype of the layer's computations and weights. Can also be a tf.keras.mixed_precision.Policy, which allows the computation and weight dtype to differ. Default of None means to use tf.keras.mixed_precision.global_policy(), which is a float32 policy unless set to a different value.

dynamic: Set this to True if your layer should only be run eagerly, and should not be used to generate a static computation graph. This would be the case for a Tree-RNN or a recursive network, for example, or generally for any layer that manipulates tensors using Python control flow. If False, we assume that the layer can safely be used to generate a static computation graph.

a) optimizers: Built-in optimizer classes.
b) experimental module: Public API for tf.keras.optimizers.experimental namespace.
c) legacy module: Public API for tf.keras.optimizers.legacy namespace.
d) schedules module: Public API for tf.keras.optimizers.schedules namespace.
e) datasets: Small NumPy datasets for debugging/testing.
f) backend: Keras backend API.
g) experimental module: Public API for tf.keras.backend.experimental namespace.

Need for flattening the dataset in standard neural network implementation.

Rectangular or cubic shapes can't be direct inputs. And this is why we need flattening and fully connected layers. Flattening is converting the data into a 1-dimensional array for inputting it to the next layer. We flatten the output of the convolutional layers to create a single long feature vector.

Sigmoid and Softmax activation function.

Sr. No.	Softmax Function	Sigmoid Function
1	Used for multi-classification in logistic regression model.	Used for binary classification in logistic regression model.
2	The probabilities sum will be 1	The probabilities sum need not be 1.
3	Used in the different layers of neural networks.	Used as activation function while building neural networks.
4	The high value will have the higher probability than other values.	The high value will have the high probability but not the higher probability.

Significance of an optimizer in a training model

An optimizer is a function or an algorithm that modifies the attributes of the neural network, such as weights and learning rate. Thus, it helps in reducing the overall loss and improving the accuracy.

Epochs in fit command in training

An epoch is a single iteration through the training data. Each sample from your training dataset will be used once per epoch, whether it is for training or validation.

Steps/ Algorithm
1. Dataset link and libraries:
Dataset: MNIST or CIFAR 10: kaggel.com
You can download the dataset from the above-mentioned website.

Libraries required:

Pandas and NumPy for data manipulation TensorFlow/Keras for Neural Networks

Scikit-learn library for splitting the data into train-test samples, and for some basic model evaluation

a) Import following libraries from SKlearn : i) LabelBinarizer (sklearn.preprocessing) ii) classification_report (sklearn.metrics) .

b) Import Following libraries from tensorflow.keras : models , layers,optimizers,datasets
, backend and set to respective values

c) Grab the MNIST dataset or required dataset.

d) Flatten the dataset.

e) If required, do the normalization of data.

f) Convert the labels from integers to vectors. (specially for one hot coding)

g) Decide the Neural Network Architecture: i) Select model (Sequential recommended)

ii) Activation function (sigmoid recommended) iii) Select the input shape iv) see the weights in the output layer

h) Train the model: i) Select optimizer (SGD recommended) ii) use model that .fit to start training ii) Set Epochs and batch size

i) Call model.predict for class prediction.

j) Plot training and loss accuracy

k) Calculate Precision, Recall, F1-score, Support

l) Repeat for CIFAR dataset.

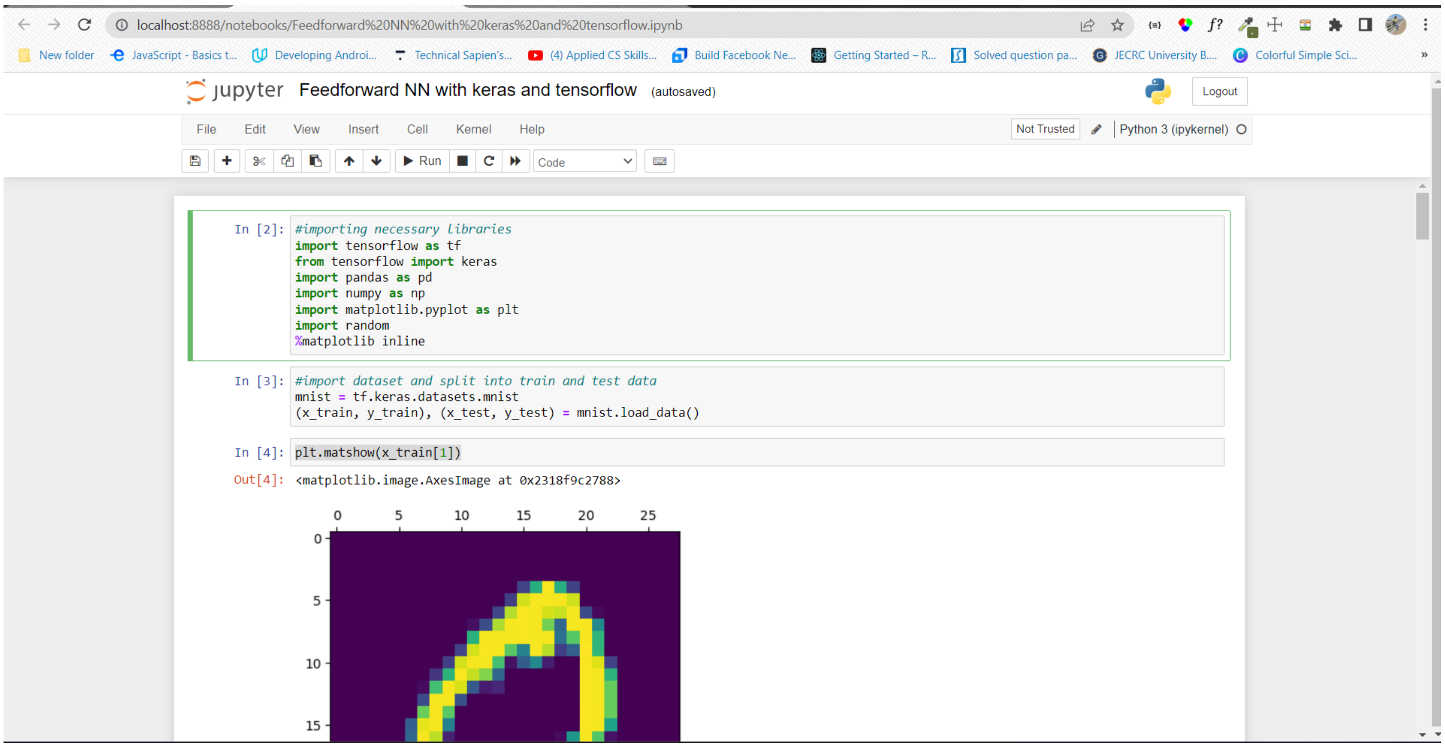
CODE

```
#importing necessary libraries
import tensorflow as tf
from tensorflow import keras
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import random
%matplotlib inline
#import dataset and split into train and test data
mnist = tf.keras.datasets.mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()
plt.matshow(x_train[1])
plt.imshow(-x_train[0], cmap="gray")
x_train = x_train / 255
x_test = x_test / 255
model = keras.Sequential([
keras.layers.Flatten(input_shape=(28, 28)),
keras.layers.Dense(128, activation="relu"),
keras.layers.Dense(10, activation="softmax")
])

model.summary()
model.compile(optimizer="sgd",
loss="sparse_categorical_crossentropy",
metrics=['accuracy'])
history=model.fit(x_train,
y_train,validation_data=(x_test,y_test),epochs=10)
test_loss,test_acc=model.evaluate(x_test,y_test)
print("Loss=%.3f" %test_loss)
print("Accuracy=%.3f" %test_acc)
n=random.randint(0,9999)
plt.imshow(x_test[n])
plt.show()
x_train
x_test
predicted_value=model.predict(x_test)
plt.imshow(x_test[n])
plt.show()
# history.history()
history.history.keys()
# dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])

plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()
print(predicted_value[n])
# history.history()
history.history.keys()
# dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

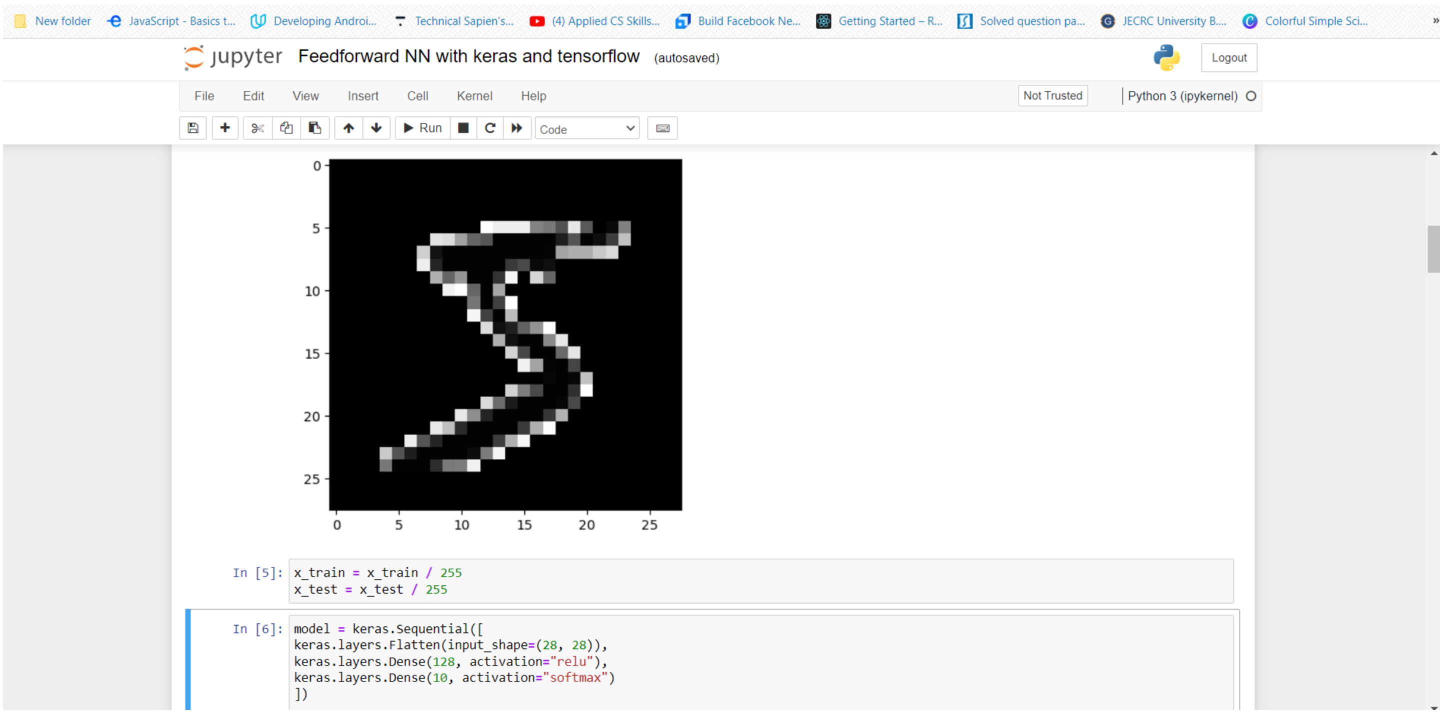
```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()
```

OUTPUT

Import the necessary packages and load the training and testing data (MNIST/CIFAR10)

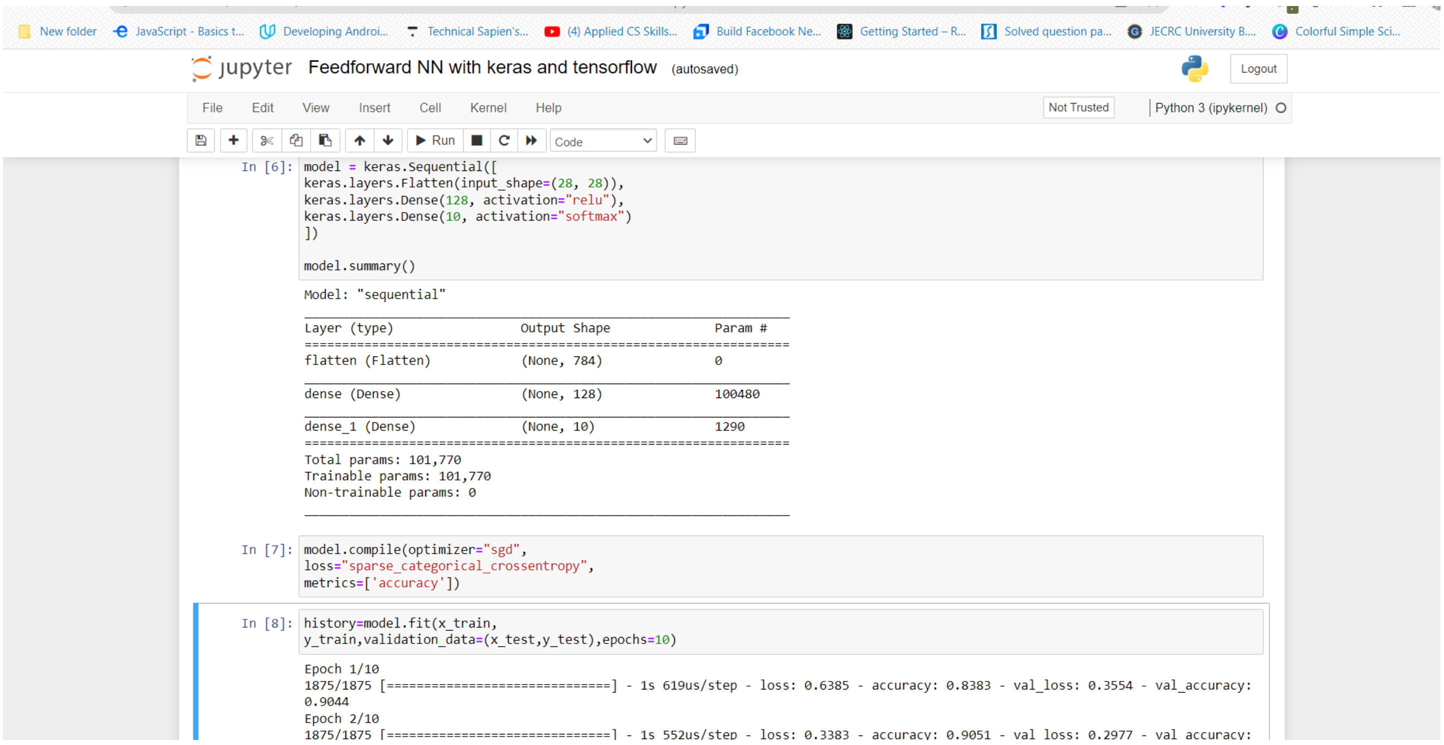


```
In [2]: #importing necessary libraries
        import tensorflow as tf
        from tensorflow import keras
        import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
        import random
        %matplotlib inline

In [3]: #import dataset and split into train and test data
        mnist = tf.keras.datasets.mnist
        (x_train, y_train), (x_test, y_test) = mnist.load_data()

In [4]: plt.matshow(x_train[1])
Out[4]: <matplotlib.image.AxesImage at 0x2318f9c2788>
```

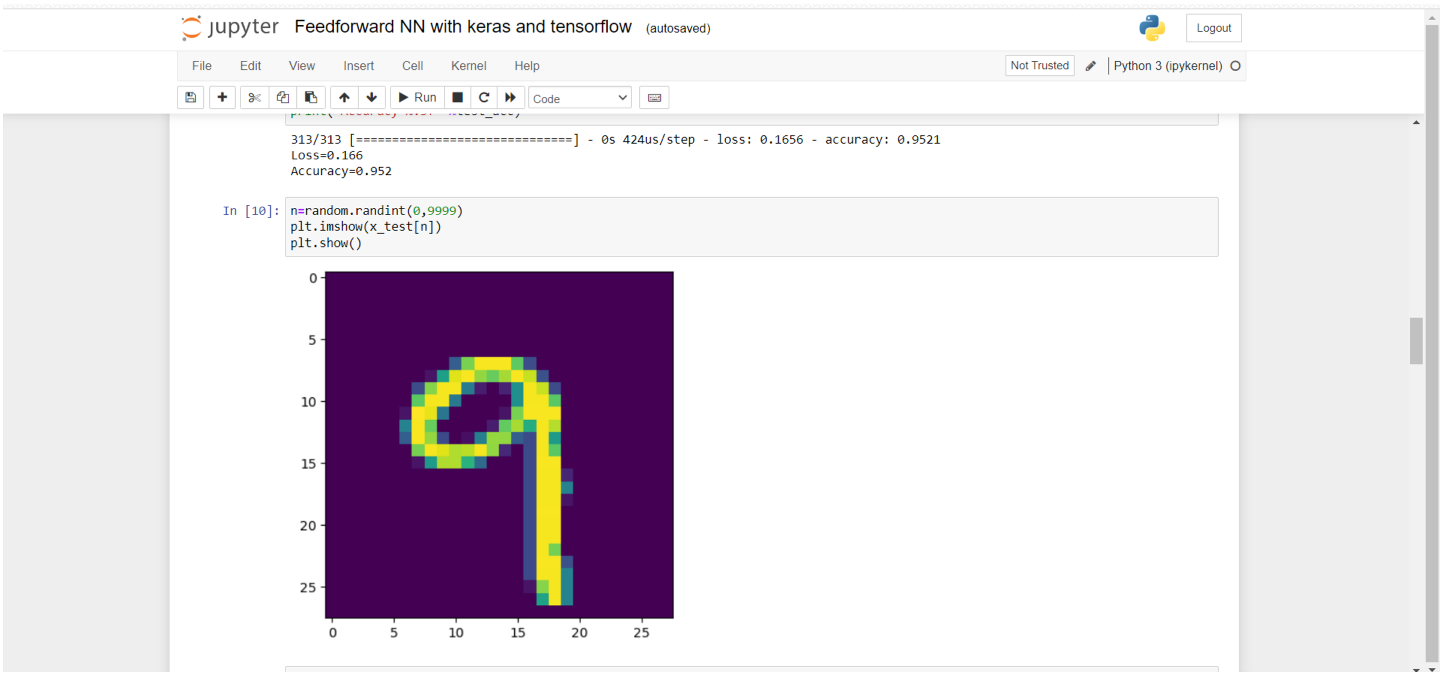


```
In [5]: x_train = x_train / 255
        x_test = x_test / 255

In [6]: model = keras.Sequential([
        keras.layers.Flatten(input_shape=(28, 28)),
        keras.layers.Dense(128, activation="relu"),
        keras.layers.Dense(10, activation="softmax")
        ])
```

Define the network architecture using Keras

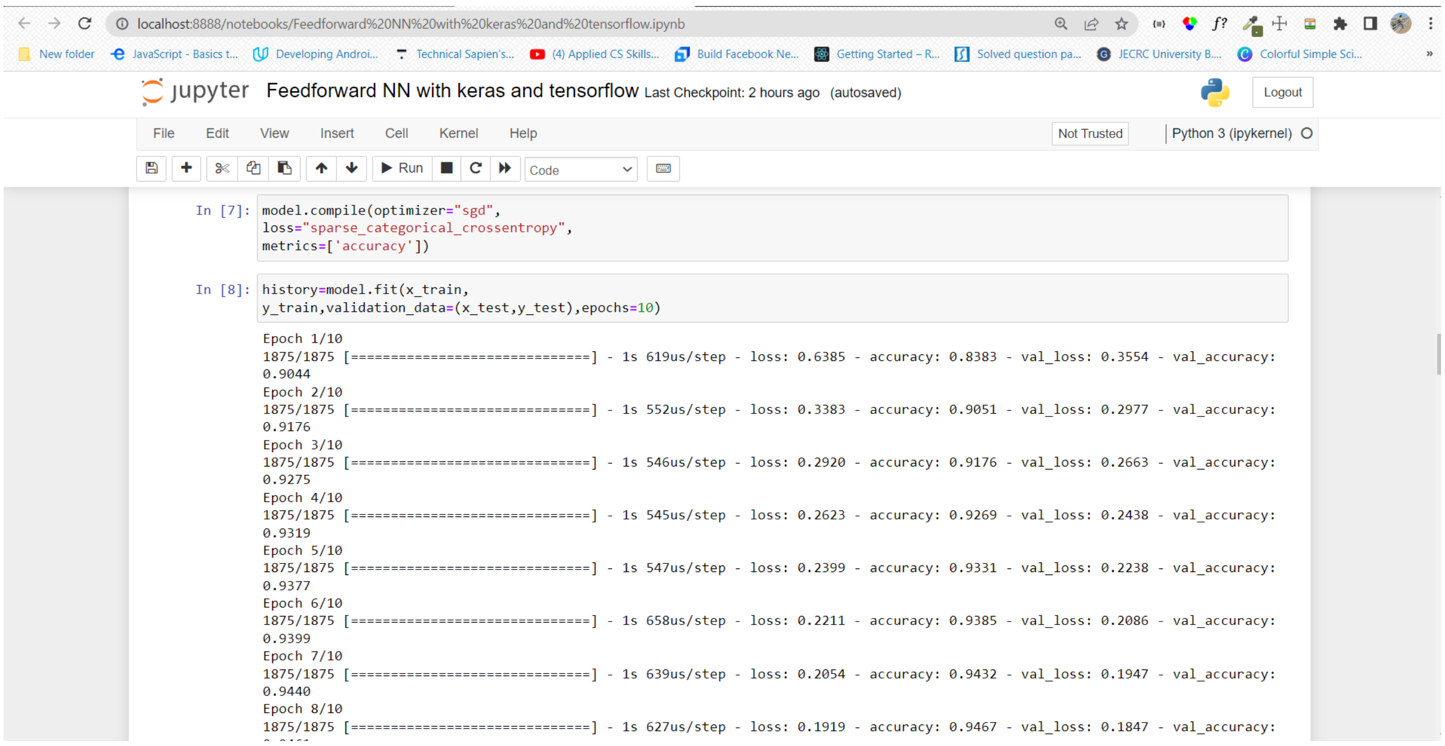

jupyter Feedforward NN with keras and tensorflow (autosaved)
Logout
File Edit View Insert Cell Kernel Help
Not Trusted
Python 3 (ipykernel)
Run
Code

```
In [6]: model = keras.Sequential([
        keras.layers.Flatten(input_shape=(28, 28)),
        keras.layers.Dense(128, activation="relu"),
        keras.layers.Dense(10, activation="softmax")
        ])

        model.summary()

        Model: "sequential"
        _________________________________________________________________
        Layer (type)                Output Shape              Param #
        =================================================================
        flatten (Flatten)           (None, 784)               0

        dense (Dense)               (None, 128)               100480

        dense_1 (Dense)             (None, 10)                1290
        =================================================================
        Total params: 101,770
        Trainable params: 101,770
        Non-trainable params: 0
        _________________________________________________________________

In [7]: model.compile(optimizer="sgd",
        loss="sparse_categorical_crossentropy",
        metrics=['accuracy'])

In [8]: history=model.fit(x_train,
        y_train,validation_data=(x_test,y_test),epochs=10)

        Epoch 1/10
        1875/1875 [==============================] - 1s 619us/step - loss: 0.6385 - accuracy: 0.8383 - val_loss: 0.3554 - val_accuracy:
        0.9044
        Epoch 2/10
        1875/1875 [==============================] - 1s 552us/step - loss: 0.3383 - accuracy: 0.9051 - val_loss: 0.2977 - val_accuracy:
```


jupyter Feedforward NN with keras and tensorflow (autosaved)
Logout
File Edit View Insert Cell Kernel Help
Not Trusted
Python 3 (ipykernel)
Run
Code
0
5
10
15
20
25
0
5
10
15
20
25

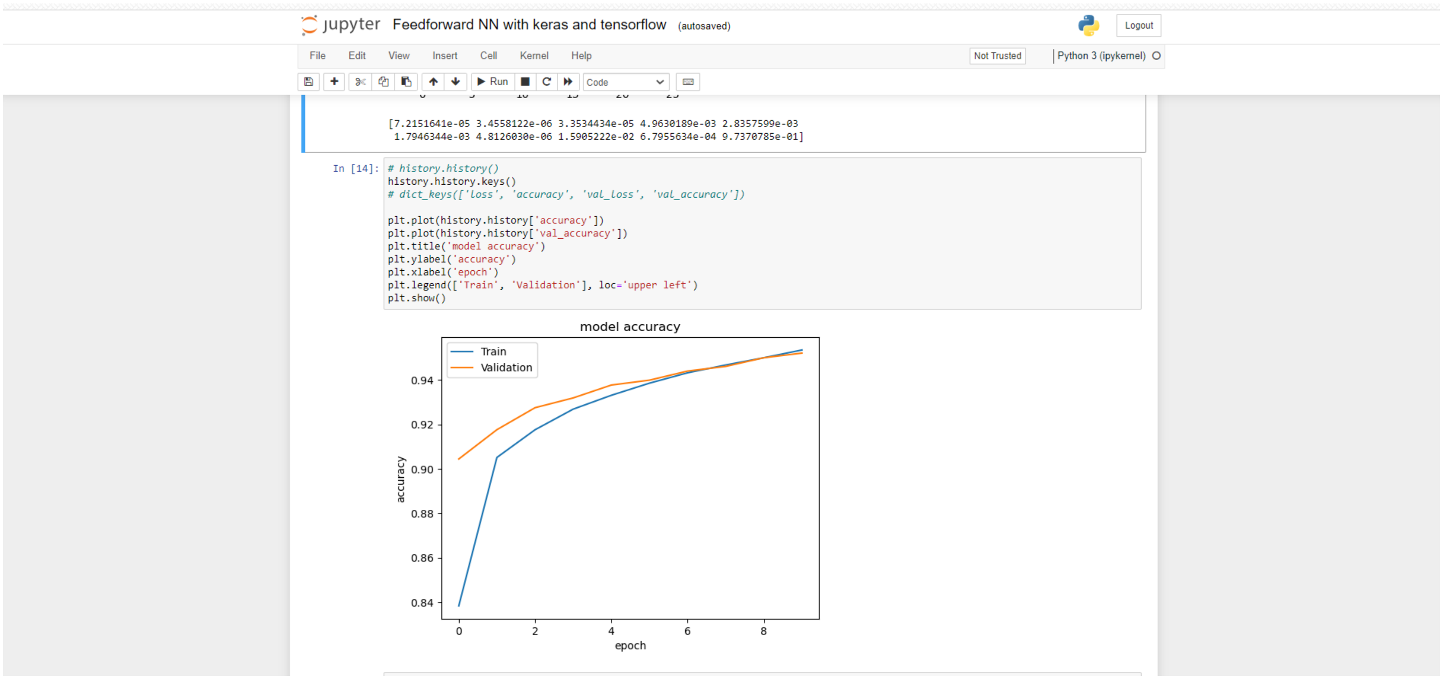
```
313/313 [==============================] - 0s 424us/step - loss: 0.1656 - accuracy: 0.9521
Loss=0.166
Accuracy=0.952

In [10]: n=random.randint(0,9999)
         plt.imshow(x_test[n])
         plt.show()
```

Train the model using SGD


localhost:8888/notebooks/Feedforward%20NN%20with%20keras%20and%20tensorflow.ipynb
jupyter Feedforward NN with keras and tensorflow Last Checkpoint: 2 hours ago (autosaved)
Logout
File Edit View Insert Cell Kernel Help
Not Trusted
Python 3 (ipykernel)
Run
Code

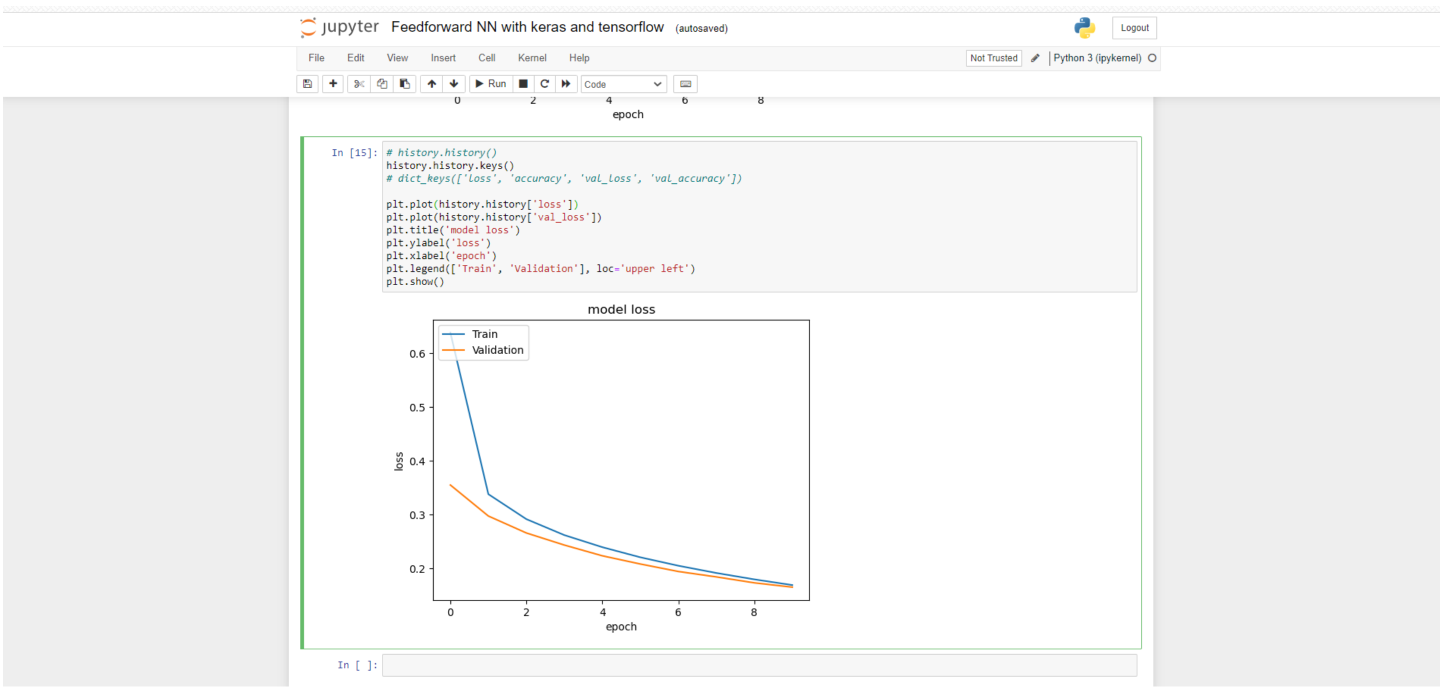
```
In [7]: model.compile(optimizer="sgd",
        loss="sparse_categorical_crossentropy",
        metrics=['accuracy'])

In [8]: history=model.fit(x_train,
        y_train,validation_data=(x_test,y_test),epochs=10)

        Epoch 1/10
        1875/1875 [==============================] - 1s 619us/step - loss: 0.6385 - accuracy: 0.8383 - val_loss: 0.3554 - val_accuracy:
        0.9044
        Epoch 2/10
        1875/1875 [==============================] - 1s 552us/step - loss: 0.3383 - accuracy: 0.9051 - val_loss: 0.2977 - val_accuracy:
        0.9176
        Epoch 3/10
        1875/1875 [==============================] - 1s 546us/step - loss: 0.2920 - accuracy: 0.9176 - val_loss: 0.2663 - val_accuracy:
        0.9275
        Epoch 4/10
        1875/1875 [==============================] - 1s 545us/step - loss: 0.2623 - accuracy: 0.9269 - val_loss: 0.2438 - val_accuracy:
        0.9319
        Epoch 5/10
        1875/1875 [==============================] - 1s 547us/step - loss: 0.2399 - accuracy: 0.9331 - val_loss: 0.2238 - val_accuracy:
        0.9377
        Epoch 6/10
        1875/1875 [==============================] - 1s 658us/step - loss: 0.2211 - accuracy: 0.9385 - val_loss: 0.2086 - val_accuracy:
        0.9399
        Epoch 7/10
        1875/1875 [==============================] - 1s 639us/step - loss: 0.2054 - accuracy: 0.9432 - val_loss: 0.1947 - val_accuracy:
        0.9440
        Epoch 8/10
        1875/1875 [==============================] - 1s 627us/step - loss: 0.1919 - accuracy: 0.9467 - val_loss: 0.1847 - val_accuracy:
```

Plot the training loss and accuracy



```
[7.2151641e-05 3.4558122e-06 3.3534434e-05 4.9630189e-03 2.8357599e-03
 1.7946344e-03 4.8126030e-06 1.5905222e-02 6.7955634e-04 9.7370785e-01]

In [14]: # history.history()
         history.history.keys()
         # dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])

         plt.plot(history.history['accuracy'])
         plt.plot(history.history['val_accuracy'])
         plt.title('model accuracy')
         plt.ylabel('accuracy')
         plt.xlabel('epoch')
         plt.legend(['Train', 'Validation'], loc='upper left')
         plt.show()
```

Evaluate the network



```
In [15]: # history.history()
         history.history.keys()
         # dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])

         plt.plot(history.history['loss'])
         plt.plot(history.history['val_loss'])
         plt.title('model loss')
         plt.ylabel('loss')
         plt.xlabel('epoch')
         plt.legend(['Train', 'Validation'], loc='upper left')
         plt.show()
```

Conclusion:
Accuracy=0.952 is achieved by Feedforward Neural Network **w**ith a Loss of 0.166

FAQs:

1)What is a Feedforward Neural Network?

2) How the Feedforward Neural Network Works?

3) Enlist at least three Real time scenarios where Feedforward Neural Network is used.

4) Explain the components of Feedforward Neural Network.

5) What is the cost function in Feedforward Neural Network?

6) Define mean square error cost function.

7) What is the Loss function in Feedforward Neural Network?

8) What is cross entropy loss?

9) What is the kernel concept related to Feedforward Neural Network?

10) Describe MNIST and CIFAR 10 Dataset.

11) Explain use and parameter setting related to feedforward network implementation for following libraries : SKlearn : i) LabelBinarizer (sklearn.preprocessing) ii) classification_report (sklearn.metrics) and tensorflow.keras : models , layers,optimizers,datasets ,backend and set to respective values.

12) What is meant by flattening the dataset and why it is needed is related to standard neural network implementation.

13) Explain the difference between Sigmoid and Softmax activation function.

14) What is the significance of an optimizer in a training model?

15) What are Epochs in fit command in training?

ASSIGNMENT NO.4

Title: Build the Image classification model

Aim: Build the Image classification model by dividing the model into following 4 stages:

a. Loading and preprocessing the image data

b. Defining the model's architecture

c. Training the model

d. Estimating the model's performance

Theory:

Problems in Image Classification

There are following main challenges in image classification:

1. Intra-Class Variation
2. Scale Variation
3. View-Point Variation
4. Occlusion
5. Illumination
6. Background Clutter

Use of Deep learning for Image classification and state and compare different types

Deep learning is good at identifying patterns in images. It uses the same neural net approach for many different problems like Support Vector Machines, Linear Classifier, Regression, Bayesian, Decision Trees, Clustering and Association Rules.

Supervised Models

1. Classic Neural Networks (Multilayer Perceptron's)
2. Convolutional Neural Networks (CNNs)
3. Recurrent Neural Networks (RNNs)

Unsupervised Models

1. Self-Organizing Maps (SOMs)
2. Boltzmann Machines
3. Autoencoders

Supervised vs Unsupervised Models

There are a number of features that distinguish the two, but the most integral point of difference is in how these models are trained. While supervised models are trained through examples of a particular set of data, unsupervised models are only given input data and don't have a set outcome they can learn from. So that y-column that we're always trying to predict is not there in an unsupervised model. While supervised models have tasks such as regression and classification and will produce a formula, unsupervised models have clustering and association rule learning.

Convolutional Neural Network (CNN) :

A convolutional neural network, or CNN, is a deep learning neural network sketched for processing structured arrays of data such as portrayals.

CNNs are very satisfactory at picking up on design in the input image, such as lines, gradients, circles, or even eyes and faces.

This characteristic makes convolutional neural networks so robust for computer vision.

CNN can run directly on an underdone image and do not need any preprocessing.

A convolutional neural network is a feed forward neural network, seldom with up to 20.

The strength of a convolutional neural network comes from a particular kind of layer called the convolutional layer.

CNN contains many convolutional layers assembled on top of each other, each one competent at recognizing more sophisticated shapes.

With three or four convolutional layers it is viable to recognize handwritten digits and with 25 layers it is possible to differentiate human faces.

The agenda for this sphere is to activate machines to view the world as humans do, perceive it in a similar fashion and even use the knowledge for a multitude of duty such as image and video recognition, image inspection and classification, media recreation, recommendation systems, natural language processing, etc.

Convolution operation and Convolution kernel related to Deep learning

The 2D convolution is a simple operation at heart: you start with a kernel, which is simply a small matrix of weights. This kernel "slides" over the 2D input data, performing an elementwise multiplication with the part of the input it is currently on, and then summing up the results into a single output pixel.

The kernel repeats this process for every location it slides over, converting a 2D matrix of features into yet another 2D matrix of features. The output features are essentially, the weighted sums (with the weights being the values of the kernel itself) of the input features located roughly in the same location of the output pixel on the input layer.

Whether or not an input feature falls within this “roughly same location”, gets determined directly by whether it’s in the kernel that produced the output or not. This means the size of the kernel directly determines how many (or few) input features get combined in the production of a new output feature.

Kernel operation on the Input image by taking a sample matrix.

A kernel is in fact a matrix with an M x N dimension that is smaller than the image matrix. The kernel is also known as the convolution matrix which is well suited for tasks like blurring, sharpening, edge-detection, and similar image processing tasks.

In the below image we have a 5 x 5 grayscale image matrix which is in yellow and the matrix which is in red (3 x 3) is the Kernel matrix to sharpen the overall image. For different image processing tasks, the kernel matrix will have varying values.

The kernel matrix will convolve through the big matrix (i.e., image matrix) from left to right and top to bottom, and at each step of convolution, it returns a single pixel value. That single pixel value is the average of the neighborhood pixels in a 3 x 3 matrix grid. Finally, the pixel value returned at each step will result in an output image matrix. For example, if we want to sharpen the given image then we can use the defined kernel matrix to achieve that task. and the resultant matrix/image would be a sharpened image.

Types of convolution and convolution layers related to CNN.

Convolution Layers

There are three types of layers that make up the CNN which are the convolutional layers, pooling layers, and fully connected (FC) layers. When these layers are stacked, a CNN architecture will be formed. In addition to these three layers, there are two more important parameters which are the dropout layer and the activation function which are defined below.

1. Convolutional Layer

This layer is the first layer that is used to extract the various features from the input images. In this layer, the mathematical operation of convolution is performed between the input image and a filter of a particular size MxM. By sliding the filter over the input image, the dot product is taken between the filter and the parts of the input image with respect to the size of the filter (MxM).

The output is termed as the Feature map which gives us information about the image such as the corners and edges. Later, this feature map is fed to other layers to learn several other features of the input image.

The convolution layer in CNN passes the result to the next layer once applying the convolution operation in the input. Convolutional layers in CNN benefit a lot as they ensure the spatial relationship between the pixels is intact.

2. Pooling Layer

In most cases, a Convolutional Layer is followed by a Pooling Layer. The primary aim of this layer is to decrease the size of the convolved feature map to reduce the computational costs. This is performed by decreasing the connections between layers and independently operates on each feature map. Depending upon the method used, there are several types of Pooling operations. It basically summarizes the features generated by a convolution layer.

In Max Pooling, the largest element is taken from the feature map. Average Pooling calculates the average of the elements in a predefined size Image section. The total sum of the elements in the predefined section is computed in Sum Pooling. The Pooling Layer usually serves as a bridge between the Convolutional Layer and the FC Layer. This CNN model generalizes the features extracted by the convolution layer and helps the networks to recognize the features independently. With the help of this, the computations are also reduced in a network.

3. Fully Connected Layer

The Fully Connected (FC) layer consists of the weights and biases along with the neurons and is used to connect the neurons between two different layers. These layers are usually placed before the output layer and form the last few layers of a CNN Architecture.

In this, the input image from the previous layers are flattened and fed to the FC layer. The flattened vector then undergoes few more FC layers where the mathematical functions operations usually take place. In this stage, the classification process begins to take place. The reason two layers are connected is that two fully connected layers will perform better than a single connected layer. These layers in CNN reduce the human supervision

4. Dropout

Usually, when all the features are connected to the FC layer, it can cause overfitting in the training dataset. Overfitting occurs when a particular model works so well on the training data causing a negative impact in the model’s performance when used on new data.

To overcome this problem, a dropout layer is utilized wherein a few neurons are dropped from the neural network during the training process resulting in reduced size of the model. On passing a dropout of 0.3, 30% of the nodes are dropped out randomly from the neural network.

Dropout results in improving the performance of a machine learning model as it prevents overfitting by making the network simpler. It drops neurons from the neural networks during training.

5. Activation Functions

Finally, one of the most important parameters of the CNN model is the activation function. They are used to learn and approximate any kind of continuous and complex relationship between variables of the network. In simple words, it decides which information of the model should fire in the forward direction and which ones should not at the end of the network.

Feature extraction with convolution layers

Fully Connected Networks

In the sparse autoencoder, one design choice that we had made was to “fully connect” all the hidden units to all the input units. On the relatively small images that we were working with (e.g., 8x8 patches for the sparse autoencoder assignment, 28x28 images for the MNIST dataset), it was computationally feasible to learn features on the entire image. However, with larger images (e.g., 96x96 images) learning features that span the entire image (fully connected networks) is very computationally expensive–you would have about 104 input units, and assuming you want to learn 100 features, you would have on the order of 106 parameters to learn. The feedforward and backpropagation computations would also be about 102 times slower, compared to 28x28 images.

Locally Connected Networks

One simple solution to this problem is to restrict the connections between the hidden units and the input units, allowing each hidden unit to connect to only a small subset of the input units. Specifically, each hidden unit will connect to only a small contiguous region of pixels in the input. (For input modalities different than images, there is often also a natural way to select “contiguous groups” of input units to connect to a single hidden unit as well; for example, for audio, a hidden unit might be connected to only the input units corresponding to a certain time span of the input audio clip.)

This idea of having locally connected networks also draws inspiration from how the early visual system is wired up in biology. Specifically, neurons in the visual cortex have localized receptive fields (i.e., they respond only to stimuli in a certain location).

Convolutions

Natural images have the property of being” ‘stationary”’, meaning that the statistics of one part of the image are the same as any other part. This suggests that the features that we learn at one part of the image can also be applied to other parts of the image, and we can use the same features at all locations.

More precisely, having learned features over small (say 8x8) patches sampled randomly from the larger image, we can then apply this learned 8x8 feature detector anywhere in the image. Specifically, we can take the learned 8x8 features and ”‘convolve”’ them with the larger image, thus obtaining a different feature activation value at each location in the image.

To give a concrete example, suppose you have learned features on 8x8 patches sampled from a 96x96 image. Suppose further this was done with an autoencoder that has 100 hidden units. To get the convolved features, for every 8x8 region of the 96x96 image, that is, the 8x8 regions starting at (1,1),(1,2),…(89,89), you would extract the 8x8 patch, and run it through your trained sparse autoencoder to get the feature activations. This would result in 100 sets of 89x89 convolved features.

Steps/ Algorithm

1. Choose a dataset of your interest or you can also create your own image dataset (Ref: https://www.kaggle.com/datasets/) Import all necessary files.

Libraries and functions required

Tensorflow,keras

NumPy : NumPy is a Python library used for working with arrays. It also has functions for working in the domain of linear algebra, fourier transform, and matrices. NumPy stands for Numerical Python. To import numpy use

import NumPy as np

pandas: pandas is a fast, powerful, flexible and easy to use open source data analysis and manipulation tool, built on top of the Python programming language. To import pandas use

import pandas as pd

sklearn : Scikit-learn (Sklearn) is the most useful and robust library for machine learning in Python. It provides a selection of efficient tools for machine learning and statistical modeling including classification, regression, clustering, and dimensionality reduction via a consistent interface in Python.

This library, which is largely written in Python, is built upon NumPy, SciPy and Matplotlib. For importing train_test_ split use

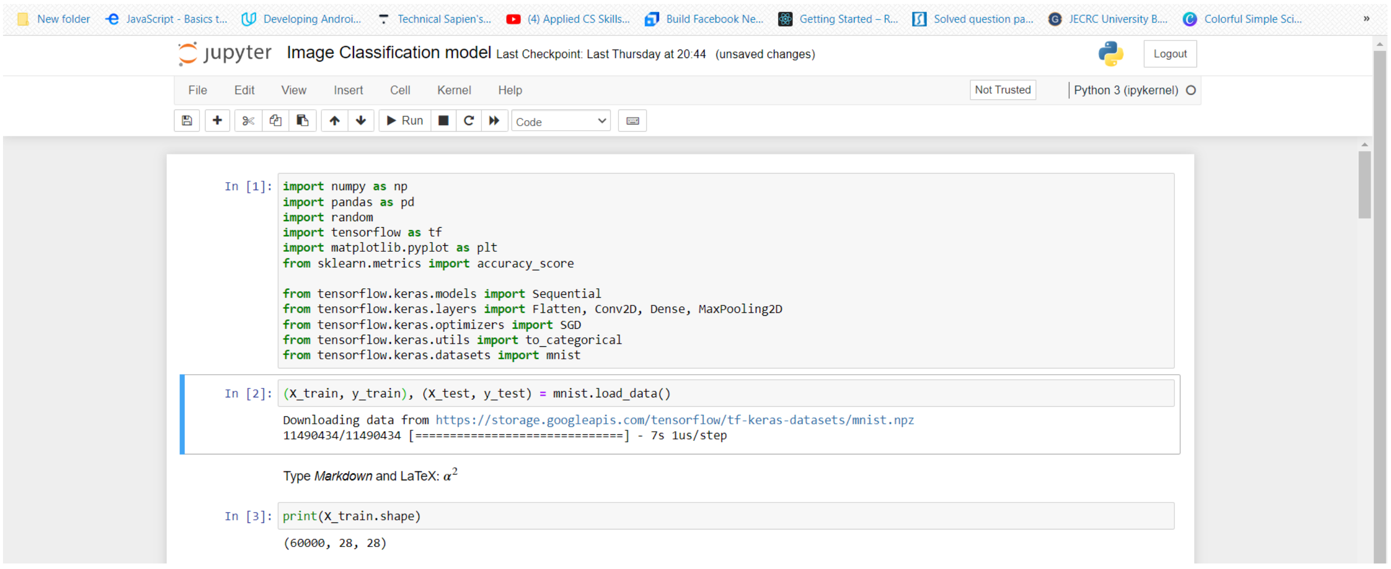
1. Prepare Dataset for Training: //Preparing our dataset for training will involve assigning paths and creating categories(labels), resizing our images.
2. Create a Training a Data: // Training is an array that will contain image pixel values and the index at which the image in the CATEGORIES list.
3. Shuffle the Dataset
4. Assigning Labels and Features
5. Normalizing X and converting labels to categorical data
6. Split X and Y for use in CNN
7. Define, compile, and train the CNN Model
8. Accuracy and Score of models

CODE

```
import numpy as np
import pandas as pd
import random
import tensorflow as tf
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Flatten, Conv2D, Dense, MaxPooling2D
from tensorflow.keras.optimizers import SGD
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.datasets import mnist

(X_train, y_train), (X_test, y_test) = mnist.load_data()

print(X_train.shape)

X_train[0].min(), X_train[0].max()

X_train = (X_train - 0.0) / (255.0 - 0.0)
X_test = (X_test - 0.0) / (255.0 - 0.0)
X_train[0].min(), X_train[0].max()

def plot_digit(image, digit, plt, i):
    plt.subplot(4, 5, i + 1)
    plt.imshow(image, cmap=plt.get_cmap('gray'))
    plt.title(f"Digit: {digit}")
    plt.xticks([])
    plt.yticks([])
plt.figure(figsize=(16, 10))
for i in range(20):
    plot_digit(X_train[i], y_train[i], plt, i)
plt.show()

y_train[0:20]

model = Sequential([
    Conv2D(32, (3, 3), activation="relu", input_shape=(28, 28, 1)),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(100, activation="relu"),
    Dense(10, activation="softmax")
])

optimizer = SGD(learning_rate=0.01, momentum=0.9)
model.compile(
    optimizer=optimizer,
    loss="sparse_categorical_crossentropy",
    metrics=["accuracy"]
)



model.summary()
model.fit(X_train, y_train, epochs=10, batch_size=32)

plt.figure(figsize=(16, 10))
for i in range(20):
    image = random.choice(X_test).squeeze()
    digit = np.argmax(model.predict(image.reshape((1, 28, 28, 1)))[0], axis=-1)
    plot_digit(image, digit, plt, i)
plt.show()

predictions = np.argmax(model.predict(X_test), axis=-1)
accuracy_score(y_test, predictions)

n=random.randint(0,9999)
plt.imshow(X_test[n])
plt.show()

predicted_value=model.predict(X_test)
print("Handwritten number in the image is= %d" %np.argmax(predicted_value[n]))

score = model.evaluate(X_test, y_test, verbose=0)
print('Test loss:', score[0]) #Test loss: 0.0296396646054
print('Test accuracy:', score[1])
```

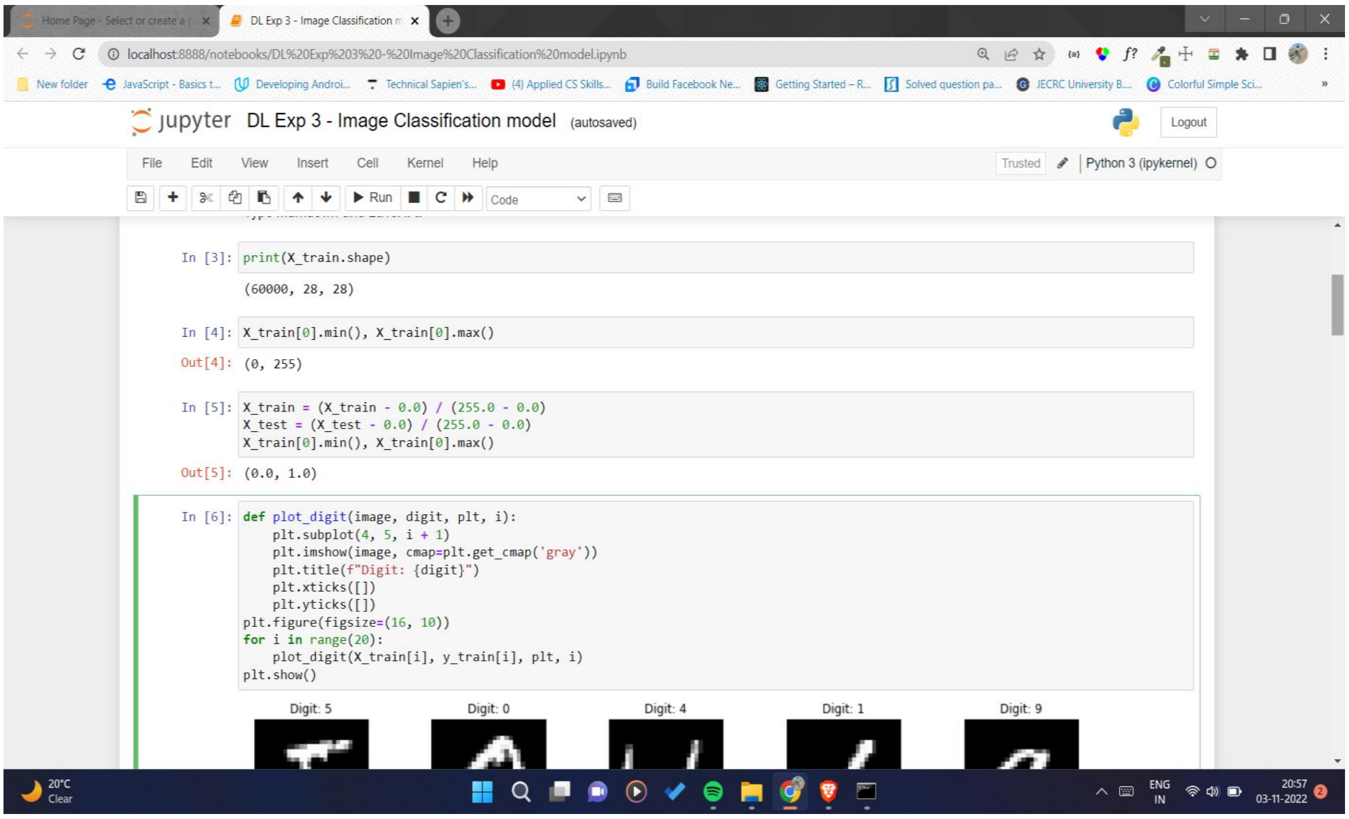
OUTPUT

Loading and preprocessing the image data

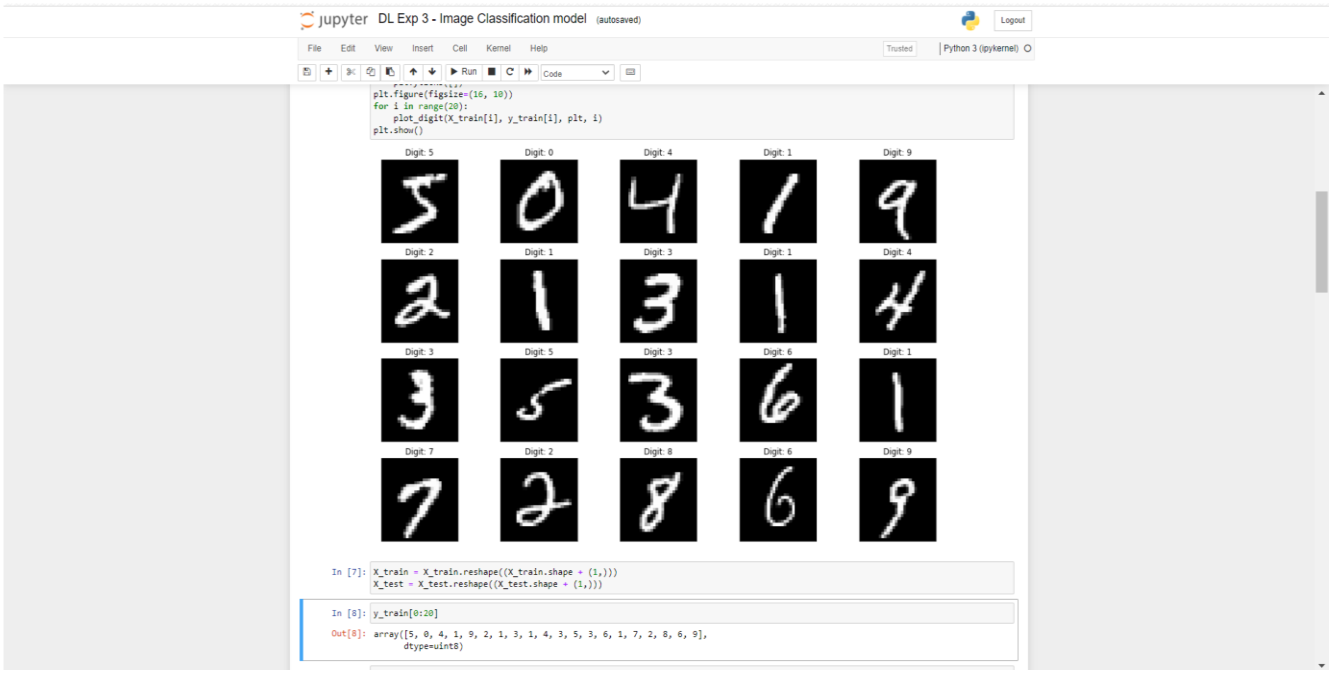

jupyter Image Classification model Last Checkpoint: Last Thursday at 20:44 (unsaved changes)
Logout
File Edit View Insert Cell Kernel Help
Not Trusted
Python 3 (ipykernel)

```
In [1]: import numpy as np
        import pandas as pd
        import random
        import tensorflow as tf
        import matplotlib.pyplot as plt
        from sklearn.metrics import accuracy_score

        from tensorflow.keras.models import Sequential
        from tensorflow.keras.layers import Flatten, Conv2D, Dense, MaxPooling2D
        from tensorflow.keras.optimizers import SGD
        from tensorflow.keras.utils import to_categorical
        from tensorflow.keras.datasets import mnist

In [2]: (X_train, y_train), (X_test, y_test) = mnist.load_data()

        Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
        11490434/11490434 [==============================] - 7s 1us/step

        Type Markdown and LaTeX: α^2

In [3]: print(X_train.shape)

        (60000, 28, 28)
```

Defining the model's architecture


Home Page - Select or create a
DL Exp 3 - Image Classification m
localhost:8888/notebooks/DL%20Exp%203%20-%20Image%20Classification%20model.ipynb
jupyter DL Exp 3 - Image Classification model (autosaved)
Logout
File Edit View Insert Cell Kernel Help
Trusted
Python 3 (ipykernel)
Digit: 5
Digit: 0
Digit: 4
Digit: 1
Digit: 9
20°C Clear
ENG IN
20:57 03-11-2022

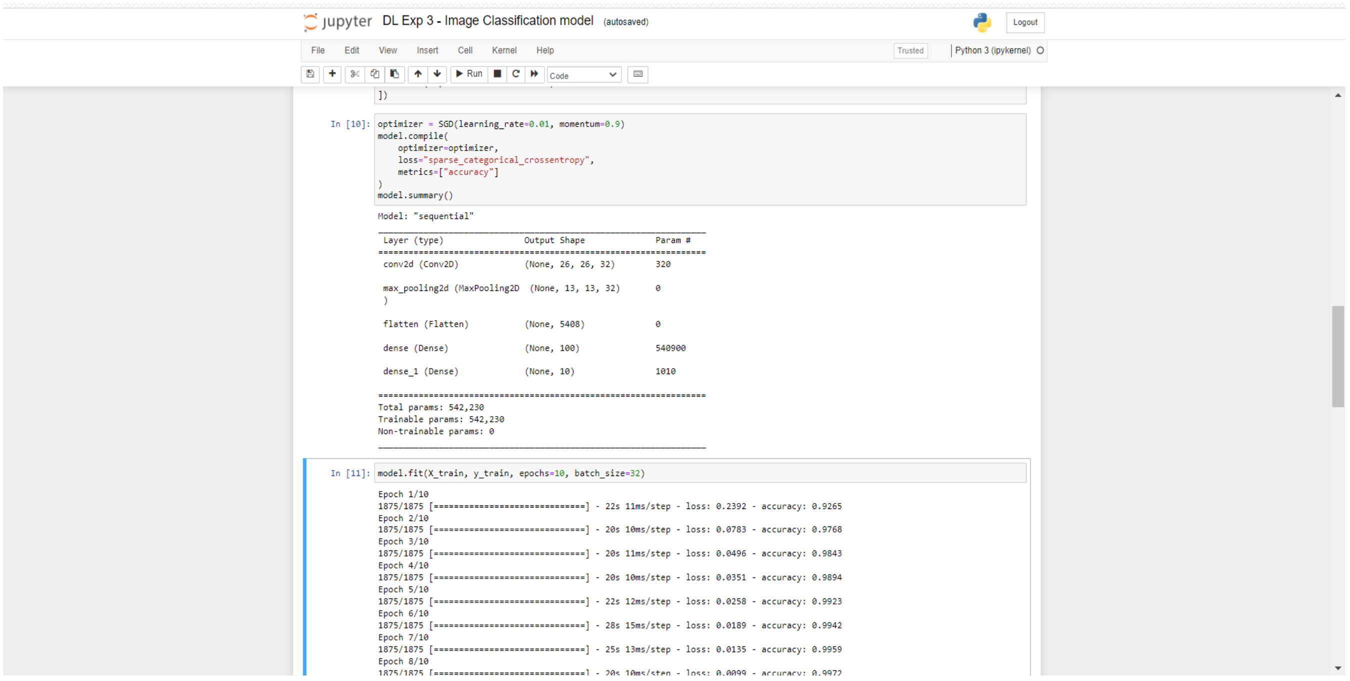
```
In [3]: print(X_train.shape)

        (60000, 28, 28)

In [4]: X_train[0].min(), X_train[0].max()

Out[4]: (0, 255)

In [5]: X_train = (X_train - 0.0) / (255.0 - 0.0)
        X_test = (X_test - 0.0) / (255.0 - 0.0)
        X_train[0].min(), X_train[0].max()

Out[5]: (0.0, 1.0)

In [6]: def plot_digit(image, digit, plt, i):
            plt.subplot(4, 5, i + 1)
            plt.imshow(image, cmap=plt.get_cmap('gray'))
            plt.title(f"Digit: {digit}")
            plt.xticks([])
            plt.yticks([])
        plt.figure(figsize=(16, 10))
        for i in range(20):
            plot_digit(X_train[i], y_train[i], plt, i)
        plt.show()
```



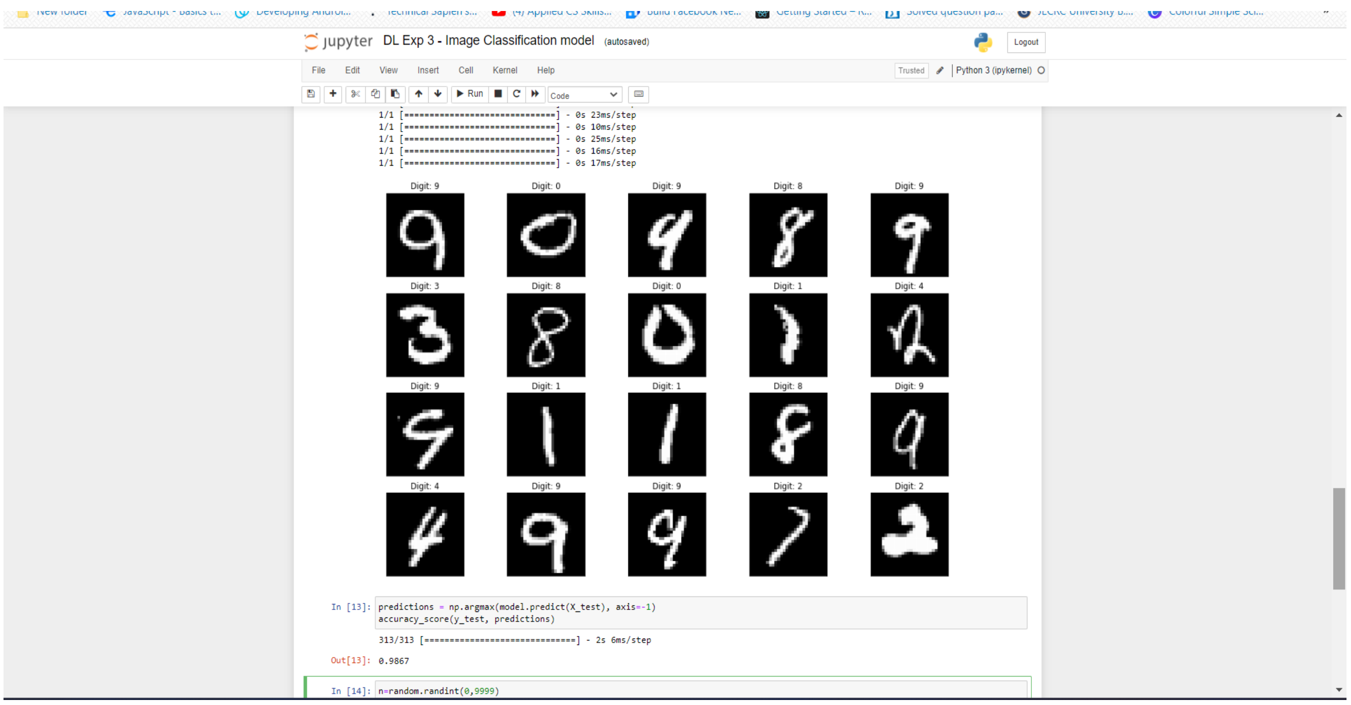
```
plt.figure(figsize=(16, 10))
for i in range(20):
    plot_digit(X_train[i], y_train[i], plt, i)
plt.show()
```

```
In [7]: X_train = X_train.reshape((X_train.shape + (1,)))
        X_test = X_test.reshape((X_test.shape + (1,)))

In [8]: y_train[0:20]

Out[8]: array([5, 0, 4, 1, 9, 2, 1, 3, 1, 4, 3, 5, 3, 6, 1, 7, 2, 8, 6, 9],
              dtype=uint8)
```

Training the model


jupyter DL Exp 3 - Image Classification model (autosaved)
Logout
File Edit View Insert Cell Kernel Help
Trusted
Python 3 (ipykernel)
Run
Code

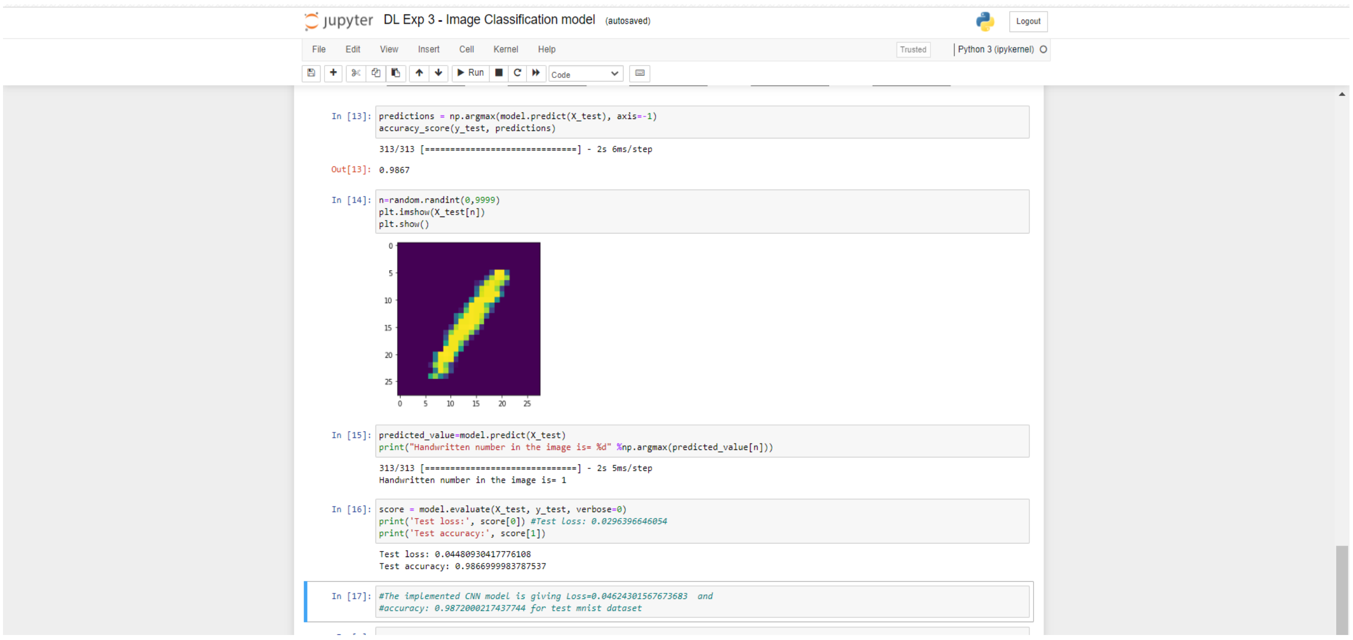
```
])

In [10]: optimizer = SGD(learning_rate=0.01, momentum=0.9)
         model.compile(
             optimizer=optimizer,
             loss="sparse_categorical_crossentropy",
             metrics=["accuracy"]
         )
         model.summary()

         Model: "sequential"
         _________________________________________________________________
          Layer (type)                Output Shape              Param #
         =================================================================
          conv2d (Conv2D)             (None, 26, 26, 32)        320

          max_pooling2d (MaxPooling2D  (None, 13, 13, 32)       0
          )

          flatten (Flatten)           (None, 5408)              0

          dense (Dense)               (None, 100)               540900

          dense_1 (Dense)             (None, 10)                1010

         =================================================================
         Total params: 542,230
         Trainable params: 542,230
         Non-trainable params: 0
         _________________________________________________________________

In [11]: model.fit(X_train, y_train, epochs=10, batch_size=32)

         Epoch 1/10
         1875/1875 [==============================] - 22s 11ms/step - loss: 0.2392 - accuracy: 0.9265
         Epoch 2/10
         1875/1875 [==============================] - 20s 10ms/step - loss: 0.0783 - accuracy: 0.9768
         Epoch 3/10
         1875/1875 [==============================] - 20s 11ms/step - loss: 0.0496 - accuracy: 0.9843
         Epoch 4/10
         1875/1875 [==============================] - 20s 10ms/step - loss: 0.0351 - accuracy: 0.9894
         Epoch 5/10
         1875/1875 [==============================] - 22s 12ms/step - loss: 0.0258 - accuracy: 0.9923
         Epoch 6/10
         1875/1875 [==============================] - 28s 15ms/step - loss: 0.0189 - accuracy: 0.9942
         Epoch 7/10
         1875/1875 [==============================] - 25s 13ms/step - loss: 0.0135 - accuracy: 0.9959
         Epoch 8/10
         1875/1875 [==============================] - 20s 10ms/step - loss: 0.0099 - accuracy: 0.9972
```



```
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 10ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 16ms/step
1/1 [==============================] - 0s 17ms/step
```

```
In [13]: predictions = np.argmax(model.predict(X_test), axis=-1)
         accuracy_score(y_test, predictions)

         313/313 [==============================] - 2s 6ms/step

Out[13]: 0.9867

In [14]: n=random.randint(0,9999)
```

Estimating the model's performance



```
In [13]: predictions = np.argmax(model.predict(X_test), axis=-1)
         accuracy_score(y_test, predictions)

         313/313 [==============================] - 2s 6ms/step

Out[13]: 0.9867

In [14]: n=random.randint(0,9999)
         plt.imshow(X_test[n])
         plt.show()
```

```
In [15]: predicted_value=model.predict(X_test)
         print("Handwritten number in the image is= %d" %np.argmax(predicted_value[n]))

         313/313 [==============================] - 2s 5ms/step
         Handwritten number in the image is= 1

In [16]: score = model.evaluate(X_test, y_test, verbose=0)
         print('Test loss:', score[0]) #Test loss: 0.0296396646054
         print('Test accuracy:', score[1])

         Test loss: 0.04480930417776108
         Test accuracy: 0.9866999983787537

In [17]: #The implemented CNN model is giving Loss=0.04624301567673683  and
         #accuracy: 0.9872000217437744 for test mnist dataset
```

Conclusion:

The implemented CNN model is giving Loss=0.04624301567673683 and accuracy: 0.9872000217437744 for test MNIST dataset

FAQ:

1. What is the Image classification problem?
2. Why use Deep learning for Image classification? State and compare different Type of Neural Networks used for the Image classification?
3. What is CNN?
4. Explain Convolution operation and Convolution kernel related to Deep learning.
5. Explain how kernels operate on the Input image by taking a sample matrix.
6. Explain the types of convolution and convolution layers related to CNN.
7. Explain how the feature extraction is done with convolution layers?

ASSIGNMENT NO.5

Title: ECG Anomaly detection using Autoencoders

Aim: Use Autoencoder to implement anomaly detection. Build the model by using:

a. Import required libraries
b. Upload / access the dataset
c. Encoder converts it into latent representation
d. Decoder networks convert it back to the original input
e. Compile the models with Optimizer, Loss, and Evaluation Metrics

Theory:

Anomaly Detection

Anomaly detection is the identification of data points, items, observations, or events that do not conform to the expected pattern of a given group. These anomalies occur very infrequently but may signify a large and significant threat such as cyber intrusions or fraud.

Anomaly detection is heavily used in behavioral analysis and other forms of analysis to aid in learning about the detection, identification, and prediction of the occurrence of these anomalies.

Autoencoders in Deep Learning

An autoencoder (AE) is an unsupervised artificial neural network that provides compression and other functions in machine learning. The Autoencoder's primary function is to reconstruct an output from the input using a feedforward approach. The input is compressed before being decompressed as output, which is generally identical to the original input. It's like an autoencoder to measure and compare similar inputs and outputs for execution results.

An autoencoder is also known as a diabolo network or an auto associator.

An encoder, a code, and a decoder are the three main components of an autoencoder. The initial data is transformed into a coded result, then expanded into a finished output by the network's succeeding layers. Examining a "denoising" autoencoder is one approach to learning about autoencoders. The denoising autoencoder refines the production by combining it with a noisy input to recreate something that represents the original set of inputs. Image processing, classification, and other parts of machine learning benefit from autoencoders.

Different applications with Autoencoders in DL.

Compression of data

Even though autoencoders are meant to compress data, they are rarely employed for this reason in practice. The following are the reasons −

Lossy compression − The Autoencoder's output is not identical to the input, but it is a near but degraded representation. They are not the best option for lossless compression.

Data-specific − Autoencoders can only compress data identical to the data on which they were trained. They differ from traditional data compression algorithms like jpeg or gzip in that they learn features relevant to the provided training data. As a result, we can't anticipate a landscape photo to be compressed by an autoencoder trained on handwritten digits.

Reduction of Dimensionality

The autoencoders reduce the input to a reduced representation stored in the middle layer called code. By separating this layer from the model, the information from the input has been compressed, and each node can now be handled as a variable. As a result, we may determine that by deleting the decoder, an autoencoder with the coding layer as the output can be used for dimensionality reduction.

Extraction of Features

Autoencoders' encoding segment aids in the learning of critical hidden features present in the input data, reducing the reconstruction error. A new set of unique feature combinations is formed during the encoding process.

Image Production

The VAE (Variational Autoencoder) is a generative model used to produce images that the model has not yet seen. The concept is that the system will generate similar images based on input photographs such as faces or scenery. The purpose is to:

- Create new animated characters
- Create fictitious human images
- Colorization of an image

Different types of anomaly detection Algorithms

1. K-nearest neighbor: k-NN

k-NN is one of the simplest supervised learning algorithms and methods in machine learning. It stores all of the available examples and then classifies the new ones based on similarities in distance metrics.

k-NN is a famous classification algorithm and a lazy learner. What does a lazy learner mean?

K-nearest neighbor mainly stores the training data. It doesn’t do anything else during the training process.

2. Local Outlier Factor (LOF)

The LOF is a key anomaly detection algorithm based on a concept of a local density. It uses the distance between the k nearest neighbors to estimate the density.

LOF compares the local density of an item to the local densities of its neighbors. Thus, one can determine areas of similar density and items that have a significantly lower density than their neighbors. These are the outliers.

To put it in other words, the density around an outlier item is seriously different from the density around its neighbors.

That is why LOF is called a density-based outlier detection algorithm. In addition, as you see, LOF is the nearest neighbor technique as k-NN.
LOF is computed on the base of the average ratio of the local reachability density of an item and its k-nearest neighbors.

3. K-means
K-means is a very popular clustering algorithm in the data mining area. It creates k groups from a set of items so that the elements of a group are more similar.
Just to recall that cluster algorithms are designed to make groups where the members are more similar. In this term, clusters and groups are synonymous.
In K-means technique, data items are clustered depending on feature similarity.
One of the greatest benefits of k-means is that it is very easy to implement. K-means is successfully implemented in most of the usual programming languages that data science uses.

4. Support Vector Machine (SVM)
A support vector machine is also one of the most effective anomaly detection algorithms. SVM is a supervised machine learning technique mostly used in classification problems.
It uses a hyperplane to classify data into 2 different groups.
Just to recall that a hyperplane is a function such as a formula for a line (e.g. y = nx + b).
SVM determines the best hyperplane that separates data into 2 classes.
To say it in another way, given labeled learning data, the algorithm produces an optimal hyperplane that categorizes the new examples.
When it comes to anomaly detection, the SVM algorithm clusters the normal data behavior using a learning area. Then, using the testing example, it identifies the abnormalities that go out of the learned area.

5. Neural Networks Based Anomaly Detection
When it comes to modern anomaly detection algorithms, we should start with neural networks.
Artificial neural networks are quite popular algorithms initially designed to mimic biological neurons.

The primary goal of creating a system of artificial neurons is to get systems that can be trained to learn some data patterns and execute functions like classification, regression, prediction, etc.

Difference between Anomaly detection and Novelty Detection.
Anomaly detection and novelty detection or noise removal are similar, but distinct. Novelty detection identifies patterns in data that were previously unobserved so users can determine whether they are anomalous. Noise removal is the process of removing noise or unneeded observations from a signal that is otherwise meaningful.

Different blocks and working of Autoencoders.

Autoencoders are the models in a dataset that find low-dimensional representations by exploiting the extreme non-linearity of neural networks. An autoencoder is made up of two parts:

Encoder – This transforms the input (high-dimensional into a code that is crisp and short.

Decoder – This transforms the short code into a high-dimensional input.

Assume that from a data-generating process, pdata(x), if X is a set of samples drawn. Suppose xi >>n; however, do not keep any restrictions on the support structure. An example of this is, for RGB images, xi >> n×m×3.

Reconstruction and Reconstruction errors

Reconstruction is prerequisite whenever a discrete signal needs to be resampled as a result of transformation such as texture mapping, image manipulation, volume slicing.and rendering.

Reconstruction error or loss is the sum of eigen values of the ignored subspace. Let's say you have 10-Dimensional data, and you are selecting first 4 principal components, what this means is your principal subspace has 4 dimensions and corresponds to 4 largest eigen values and respective vectors, So reconstruction error is sum of 6 eigen values of the ignored subspace, (the smallest 6).

Minimizing the reconstruction error means minimizing the contribution of ignored eigenvalues which depends on the distribution of the data and how many components we are selecting.

Minmaxscaler from sklearn.

There is another way of data scaling, where the minimum of feature is made equal to zero and the maximum of features equal to one. MinMax Scaler shrinks the data within the given range, usually of 0 to 1. It transforms data by scaling features to a given range. It scales the values to a specific value range without changing the shape of the original distribution.

train_test_split from sklearn.

The train_test_split() method is used to split our data into train and test sets.

First, we need to divide our data into features (X) and labels (y). The dataframe gets divided into X_train,X_test , y_train and y_test. X_train and y_train sets are used for training and fitting the model.

The X_test and y_test sets are used for testing the model if it's predicting the right outputs/labels. We can explicitly test the size of the train and test sets. It is suggested to keep our train sets larger than the test sets.

Anomaly scores

The anomaly score (severity) is a value from 0 to 100, which indicates the significance of the observed anomaly compared to previously seen anomalies. Highly anomalous values are shown in red.

TensorFlow dataset

TensorFlow Datasets is a collection of datasets ready to use, with TensorFlow or other Python ML frameworks, such as Jax. All datasets are exposed as tf.data.Datasets , enabling easy-to-use and high-performance input pipelines. To get started see the guide and our list of datasets.

ECG Dataset.

An electrocardiogram (ECG) is a simple test that measures the heart's rhythm and corresponding electrical activity. Sensors which are attached to the skin are used to detect the electrical signals produced by the heart each time it beats.

Keras Optimizers

Optimizers are not integral to Keras but a general concept used in Neural Network and Keras has out of the box implementations for Optimizers.

Keras layers dense and dropouts

Dense layer is the regular deeply connected neural network layer. It is the most common and frequently used layer. Dense layer does the below operation on the input and returns the output.

output = activation(dot(input, kernel) + bias)

where,

- input represent the input data
- kernel represent the weight data
- dot represent numpy dot product of all input and its corresponding weights
- bias represent a biased value used in machine learning to optimize the model
- activation represent theDropout Layer
- The dropout layer is an important layer for reducing overfitting in neural network models.
- Intuitively, the main purpose of the dropout layer is to remove the noise that may be present in the input of neurons. This consequently prevents overfitting of models.
- Dropout has three arguments, and they are as follows-
- rate − This parameter tells the layer how much of the input data has to be dropped. The range of the value is from 0 to 1.
- noise_shape – This parameter will specify the dimension of shape for applying dropout layer.

- seed − The seed parameter helps in providing random seed i.e. value to the layer. activation function

Keras losses and mean squared logarithmic error

Loss Functions, also known as cost functions, are used for computing the error with the aim that the model should minimize it during training.

Loss Functions also help in finding out the slope i.e. gradient w.r.t. weights used in the model and then these weights are updated after each epoch with the help of backpropagation.

Mean squared logarithmic error (MSLE) can be interpreted as a measure of the ratio between the true and predicted values.

Mean squared logarithmic error is, as the name suggests, a variation of the Mean Squared Error.

MSLE only care about the percentage difference

The introduction of the logarithm makes MSLE only care about the relative difference between the true and the predicted value, or in other words, it only cares about the percentual difference between them.

This means that MSLE will treat small differences between small true and predicted values approximately the same as big differences between large true and predicted values.

Relu activation function

The ReLU function is the Rectified linear unit. It is the most widely used activation function. It is defined as:

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

The main advantage of using the ReLU function over other activation functions is that it does not activate all the neurons at the same time. What does this mean ? If you look at the ReLU function, if the input is negative it will convert it to zero and the neuron does not get activated.

Steps/ Algorithm

1. Dataset link and libraries:

Dataset: http://storage.googleapis.com/download.tensorflow.org/data/ecg.csv

Libraries required :

Pandas and Numpy for data manipulation

Tensorflow/Keras for Neural Networks

Scikit-learn library for splitting the data into train-test samples, and for some basic model evaluation

For Model building and evaluation following libraries:

sklearn.metrics import accuracy_score

tensorflow.keras.optimizers import Adam

sklearn.preprocessing import MinMaxScaler

tensorflow.keras import Model, Sequential

tensorflow.keras.layers import Dense, Dropout

tensorflow.keras.losses

import MeanSquaredLogarithmicError

a) Import following libraries from SKlearn : i) MinMaxscaler (sklearn.preprocessing) ii) Accuracy(sklearn.metrics) . iii) train_test_split (model_selection)

b) Import Following libraries from tensorflow.keras : models , layers,optimizers,datasets , and set to respective values.

c) Grab to ECG.csv required dataset

d) Find shape of dataset

e) Use train_test_split from sklearn to build model (e.g. train_test_split(features,

target, test_size=0.2, stratify=target)

f) Take use case Novelty detection hence select training data set as Target class is 1 i.e. Normal class

g) Scale the data using MinMaxScaler.

h) Create Autoencoder Subclass by extending model class from keras.

i) Select parameters as i)Encoder : 4 layers ii) Decoder : 4 layers iii) Activation Function : Relu iv) Model : sequential.

j) Configure model with following parameters : epoch = 20 , batch size =512 and compile with Mean Squared Logarithmic loss and Adam optimizer.

e.g. model = AutoEncoder(output_units=x_train_scaled.shape[1]) #

configurations of model

model.compile(loss='msle', metrics=['mse'], optimizer='adam')

history = model.fit(

x_train_scaled, x_train_scaled,

epochs=20, batch_size=512,

validation_data=(x_test_scaled, x_test_scaled)

k) Plot loss,Val_loss, Epochs and msle loss

l) Find threshold for anomaly and do predictions :

e.g.: find_threshold(model, x_train_scaled): reconstructions =

model.predict(x_train_scaled) # provides losses of individual

instances

reconstruction_errors = tf.keras.losses.msle(reconstructions, x_train_scaled) # threshold

for anomaly scores

threshold = np.mean(reconstruction_errors.numpy()) \
m) + np.std(reconstruction_errors.numpy()) return threshold ‘
n) Get accuracy score

CODE

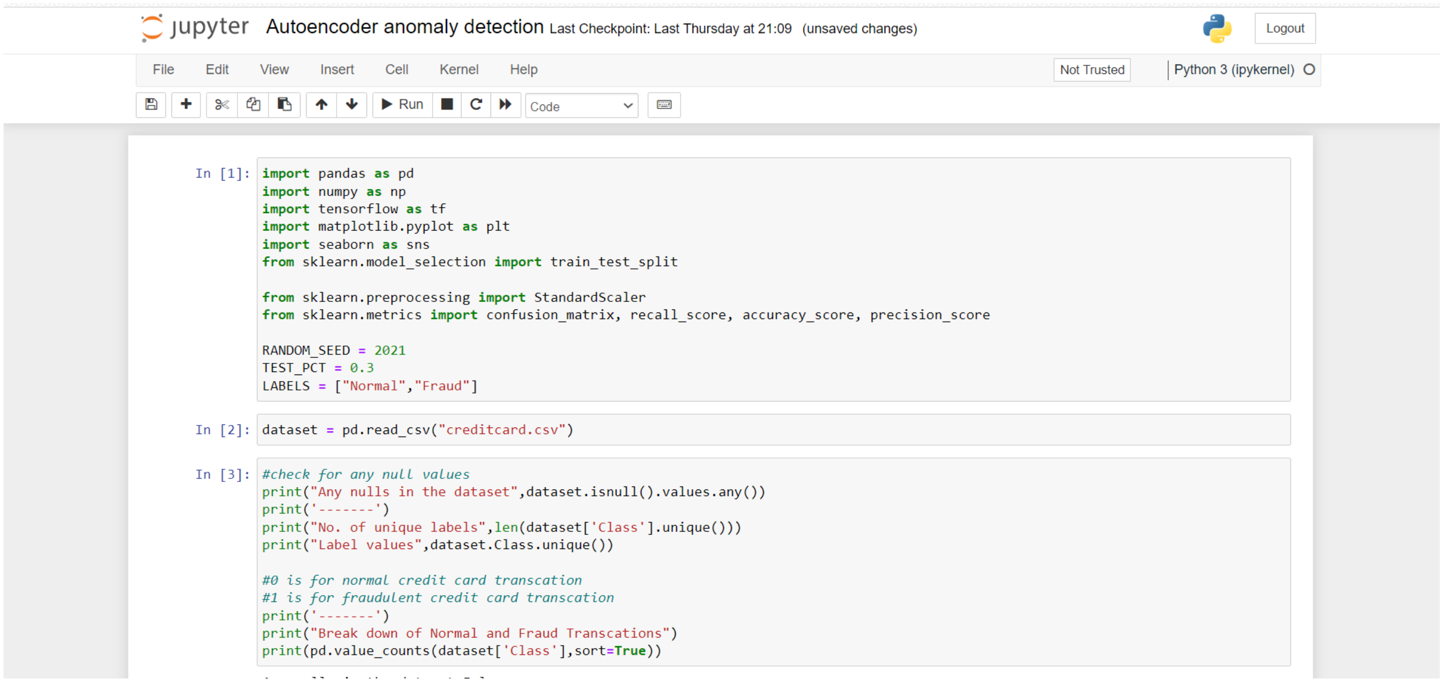
```
import pandas as pd
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix, recall_score, accuracy_score, precision_score

RANDOM_SEED = 2021
TEST_PCT = 0.3
LABELS = ["Normal","Fraud"]

dataset = pd.read_csv("creditcard.csv")

#check for any null values
print("Any nulls in the dataset",dataset.isnull().values.any())
print('-------')
print("No. of unique labels",len(dataset['Class'].unique()))
print("Label values",dataset.Class.unique())

#0 is for normal credit card transcation
#1 is for fraudulent credit card transcation
print('-------')
print("Break down of Normal and Fraud Transcations")
print(pd.value_counts(dataset['Class'],sort=True))

#visualizing the imbalanced dataset
count_classes = pd.value_counts(dataset['Class'],sort=True)
count_classes.plot(kind='bar',rot=0)
plt.xticks(range(len(dataset['Class'].unique())),dataset.Class.unique())
plt.title("Frequency by observation number")
plt.xlabel("Class")
plt.ylabel("Number of Observations")

#Save the normal and fraudulent transactions in separate data frame
normal_dataset = dataset[dataset.Class == 0]
fraud_dataset = dataset[dataset.Class == 1]

#Visualize transaction amounts for normal and fraudulent transactions
bins = np.linspace(200,2500,100)
plt.hist(normal_dataset.Amount,bins=bins,alpha=1,density=True,label='Normal')
plt.hist(fraud_dataset.Amount,bins=bins,alpha=0.5,density=True,label='Fraud')
plt.legend(loc='upper right')
plt.title("Transaction Amount vs Percentage of Transactions")
plt.xlabel("Transaction Amount (USD)")

plt.ylabel("Percentage of Transactions")
plt.show()

dataset
```

```
sc = StandardScaler()
dataset['Time'] = sc.fit_transform(dataset['Time'].values.reshape(-1,1))
dataset['Amount'] = sc.fit_transform(dataset['Amount'].values.reshape(-1,1))

raw_data = dataset.values
#The last element contains if the transaction is normal which is represented by 0 and if fraud then 1
labels = raw_data[:,-1]

#The other data points are the electrocardiogram data
data = raw_data[:,0:-1]

train_data,test_data,train_labels,test_labels = train_test_split(data,labels,test_size = 0.2,random_state =2021)

min_val = tf.reduce_min(train_data)
max_val = tf.reduce_max(train_data)

train_data = (train_data - min_val) / (max_val - min_val)
test_data = (test_data - min_val) / (max_val - min_val)

train_data = tf.cast(train_data,tf.float32)
test_data = tf.cast(test_data,tf.float32)

train_labels = train_labels.astype(bool)
test_labels = test_labels.astype(bool)

#Creating normal and fraud datasets
normal_train_data = train_data[~train_labels]
normal_test_data = test_data[~test_labels]

fraud_train_data = train_data[train_labels]
fraud_test_data = test_data[test_labels]
print("No. of records in Fraud Train Data=",len(fraud_train_data))
print("No. of records in Normal Train Data=",len(normal_train_data))
print("No. of records in Fraud Test Data=",len(fraud_test_data))
print("No. of records in Normal Test Data=",len(normal_test_data))

nb_epoch = 50
batch_size = 64
input_dim = normal_train_data.shape[1]
#num of columns,30
encoding_dim = 14
hidden_dim1 = int(encoding_dim / 2)
hidden_dim2 = 4
learning_rate = 1e-7

          #input layer
input_layer = tf.keras.layers.Input(shape=(input_dim,))

#Encoder
encoder = tf.keras.layers.Dense(encoding_dim,activation="tanh",activity_regularizer =


tf.keras.regularizers.l2(learning_rate))(input_layer)
encoder = tf.keras.layers.Dropout(0.2)(encoder)
encoder = tf.keras.layers.Dense(hidden_dim1,activation='relu')(encoder)
encoder = tf.keras.layers.Dense(hidden_dim2,activation=tf.nn.leaky_relu)(encoder)

#Decoder
decoder = tf.keras.layers.Dense(hidden_dim1,activation='relu')(encoder)
decoder = tf.keras.layers.Dropout(0.2)(decoder)
decoder = tf.keras.layers.Dense(encoding_dim,activation='relu')(decoder)
decoder = tf.keras.layers.Dense(input_dim,activation='tanh')(decoder)

#Autoencoder
autoencoder = tf.keras.Model(inputs = input_layer,outputs = decoder)
autoencoder.summary()

cp =
tf.keras.callbacks.ModelCheckpoint(filepath="autoencoder_fraud.h5",mode='min',monitor='val_loss',verbose=2,sav
e_best_only=True)
#Define our early stopping
early_stop = tf.keras.callbacks.EarlyStopping(
            monitor='val_loss',
            min_delta=0.0001,
            patience=10,
            verbose=11,
            mode='min',
            restore_best_weights=True
)

autoencoder.compile(metrics=['accuracy'],loss= 'mean_squared_error',optimizer='adam')

history = autoencoder.fit(normal_train_data,normal_train_data,epochs = nb_epoch,
batch_size = batch_size,shuffle = True,
validation_data = (test_data,test_data),
verbose=1,
callbacks = [cp,early_stop]).history

plt.plot(history['loss'],linewidth = 2,label = 'Train')
plt.plot(history['val_loss'],linewidth = 2,label = 'Test')
plt.legend(loc='upper right')
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')

#plt.ylim(ymin=0.70,ymax=1)

plt.show()

test_x_predictions = autoencoder.predict(test_data)
mse = np.mean(np.power(test_data - test_x_predictions, 2),axis = 1)
error_df = pd.DataFrame({'Reconstruction_error':mse,
                'True_class':test_labels})

threshold_fixed = 50
groups = error_df.groupby('True_class')
fig,ax = plt.subplots()
```

```
for name,group in groups:
        ax.plot(group.index,group.Reconstruction_error,marker='o',ms = 3.5,linestyle='',
                label = "Fraud" if  name==1 else "Normal")
ax.hlines(threshold_fixed,ax.get_xlim()[0],ax.get_xlim()[1],colors="r",zorder=100,label="Threshold")
ax.legend()
plt.title("Reconstructions error for normal and fraud data")
plt.ylabel("Reconstruction error")
plt.xlabel("Data point index")
plt.show()

threshold_fixed = 52
pred_y = [1 if e > threshold_fixed else 0
          for e in
        error_df.Reconstruction_error.values]
error_df['pred'] = pred_y
conf_matrix = confusion_matrix(error_df.True_class,pred_y)

plt.figure(figsize = (4,4))
sns.heatmap(conf_matrix,xticklabels = LABELS,yticklabels = LABELS,annot = True,fmt="d")
plt.title("Confusion matrix")
plt.ylabel("True class")
plt.xlabel("Predicted class")
plt.show()

#Print Accuracy,Precision and Recall
print("Accuracy :",accuracy_score(error_df['True_class'],error_df['pred']))
print("Recall :",recall_score(error_df['True_class'],error_df['pred']))
print("Precision :",precision_score(error_df['True_class'],error_df['pred']))
```

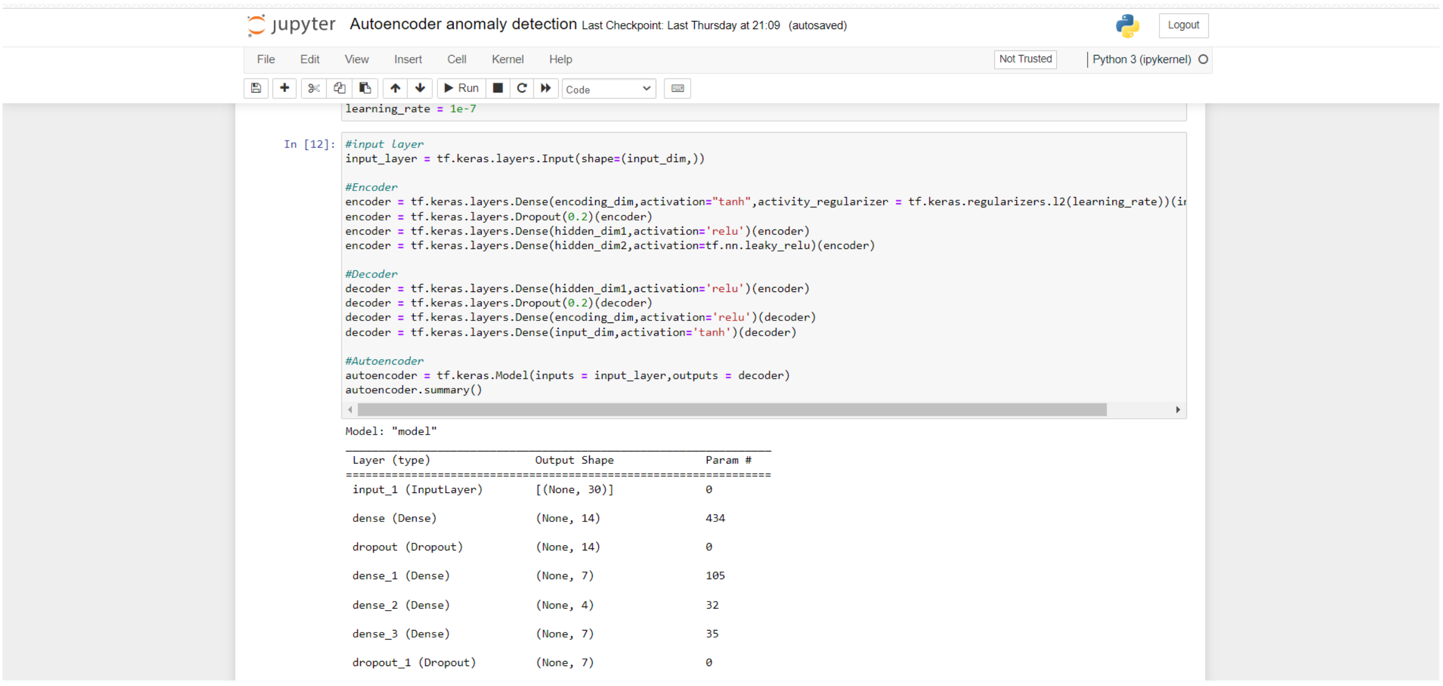
OUTPUT

Upload / access the dataset


jupyter
Autoencoder anomaly detection Last Checkpoint: Last Thursday at 21:09 (unsaved changes)
Logout
File Edit View Insert Cell Kernel Help
Not Trusted
Python 3 (ipykernel)
Run
Code

```
In [1]: import pandas as pd
        import numpy as np
        import tensorflow as tf
        import matplotlib.pyplot as plt
        import seaborn as sns
        from sklearn.model_selection import train_test_split

        from sklearn.preprocessing import StandardScaler
        from sklearn.metrics import confusion_matrix, recall_score, accuracy_score, precision_score

        RANDOM_SEED = 2021
        TEST_PCT = 0.3
        LABELS = ["Normal","Fraud"]

In [2]: dataset = pd.read_csv("creditcard.csv")

In [3]: #check for any null values
        print("Any nulls in the dataset",dataset.isnull().values.any())
        print('-------')
        print("No. of unique labels",len(dataset['Class'].unique()))
        print("Label values",dataset.Class.unique())

        #0 is for normal credit card transcation
        #1 is for fraudulent credit card transcation
        print('-------')
        print("Break down of Normal and Fraud Transcations")
        print(pd.value_counts(dataset['Class'],sort=True))
```

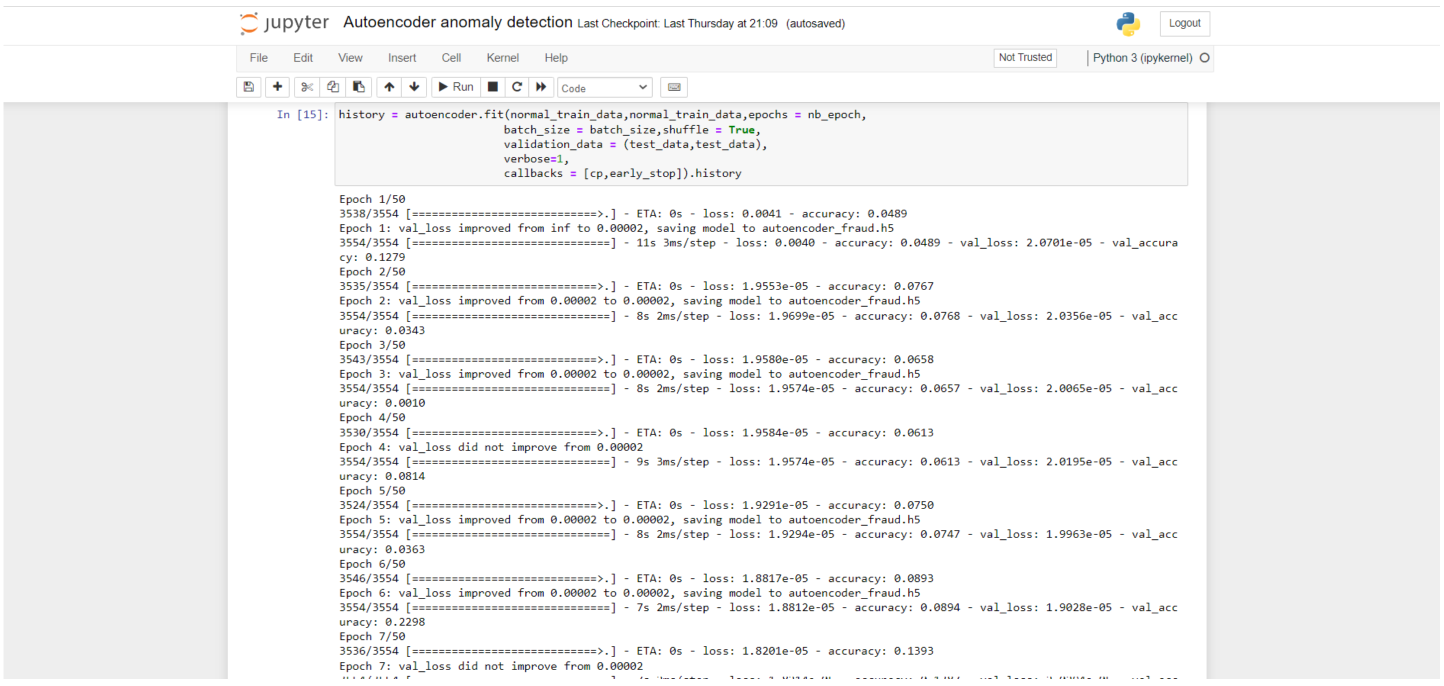
Encoder converts it into latent representation
Decoder networks convert it back to the original input


jupyter
Autoencoder anomaly detection Last Checkpoint: Last Thursday at 21:09 (autosaved)
Logout
File Edit View Insert Cell Kernel Help
Not Trusted
Python 3 (ipykernel)
Run
Code

```
         learning_rate = 1e-7

In [12]: #input layer
         input_layer = tf.keras.layers.Input(shape=(input_dim,))

         #Encoder
         encoder = tf.keras.layers.Dense(encoding_dim,activation="tanh",activity_regularizer = tf.keras.regularizers.l2(learning_rate))(ir
         encoder = tf.keras.layers.Dropout(0.2)(encoder)
         encoder = tf.keras.layers.Dense(hidden_dim1,activation='relu')(encoder)
         encoder = tf.keras.layers.Dense(hidden_dim2,activation=tf.nn.leaky_relu)(encoder)

         #Decoder
         decoder = tf.keras.layers.Dense(hidden_dim1,activation='relu')(encoder)
         decoder = tf.keras.layers.Dropout(0.2)(decoder)
         decoder = tf.keras.layers.Dense(encoding_dim,activation='relu')(decoder)
         decoder = tf.keras.layers.Dense(input_dim,activation='tanh')(decoder)

         #Autoencoder
         autoencoder = tf.keras.Model(inputs = input_layer,outputs = decoder)
         autoencoder.summary()

         Model: "model"
         _________________________________________________________________
          Layer (type)                Output Shape              Param #
         =================================================================
          input_1 (InputLayer)        [(None, 30)]              0

          dense (Dense)               (None, 14)                434

          dropout (Dropout)           (None, 14)                0

          dense_1 (Dense)             (None, 7)                 105

          dense_2 (Dense)             (None, 4)                 32

          dense_3 (Dense)             (None, 7)                 35

          dropout_1 (Dropout)         (None, 7)                 0
```

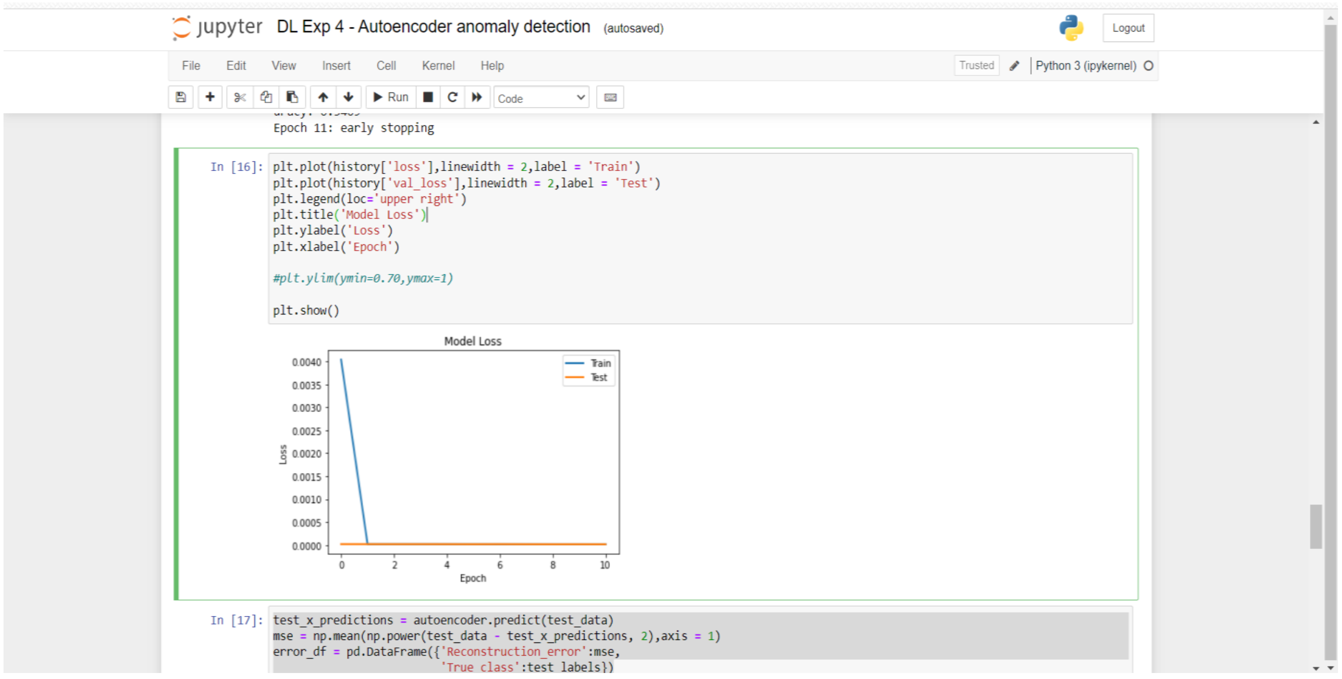
Compile the models with Optimizer, Loss, and Evaluation Metrics

```
In [11]: nb_epoch = 50
         batch_size = 64
         input_dim = normal_train_data.shape[1]
         #num of columns,30
         encoding_dim = 14
         hidden_dim1 = int(encoding_dim / 2)
         hidden_dim2 = 4
         learning_rate = 1e-7
```

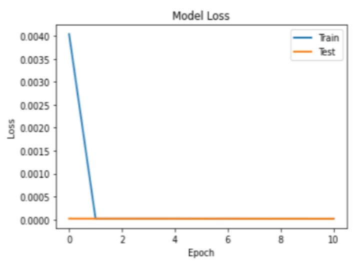

jupyter Autoencoder anomaly detection Last Checkpoint: Last Thursday at 21:09 (autosaved)
Logout
File Edit View Insert Cell Kernel Help
Not Trusted
Python 3 (ipykernel)
Run
Code

```
In [15]: history = autoencoder.fit(normal_train_data,normal_train_data,epochs = nb_epoch,
                                   batch_size = batch_size,shuffle = True,
                                   validation_data = (test_data,test_data),
                                   verbose=1,
                                   callbacks = [cp,early_stop]).history

Epoch 1/50
3538/3554 [============================>.] - ETA: 0s - loss: 0.0041 - accuracy: 0.0489
Epoch 1: val_loss improved from inf to 0.00002, saving model to autoencoder_fraud.h5
3554/3554 [==============================] - 11s 3ms/step - loss: 0.0040 - accuracy: 0.0489 - val_loss: 2.0701e-05 - val_accura
cy: 0.1279
Epoch 2/50
3535/3554 [============================>.] - ETA: 0s - loss: 1.9553e-05 - accuracy: 0.0767
Epoch 2: val_loss improved from 0.00002 to 0.00002, saving model to autoencoder_fraud.h5
3554/3554 [==============================] - 8s 2ms/step - loss: 1.9699e-05 - accuracy: 0.0768 - val_loss: 2.0356e-05 - val_acc
uracy: 0.0343
Epoch 3/50
3543/3554 [============================>.] - ETA: 0s - loss: 1.9580e-05 - accuracy: 0.0658
Epoch 3: val_loss improved from 0.00002 to 0.00002, saving model to autoencoder_fraud.h5
3554/3554 [==============================] - 8s 2ms/step - loss: 1.9574e-05 - accuracy: 0.0657 - val_loss: 2.0065e-05 - val_acc
uracy: 0.0010
Epoch 4/50
3530/3554 [============================>.] - ETA: 0s - loss: 1.9584e-05 - accuracy: 0.0613
Epoch 4: val_loss did not improve from 0.00002
3554/3554 [==============================] - 9s 3ms/step - loss: 1.9574e-05 - accuracy: 0.0613 - val_loss: 2.0195e-05 - val_acc
uracy: 0.0814
Epoch 5/50
3524/3554 [============================>.] - ETA: 0s - loss: 1.9291e-05 - accuracy: 0.0750
Epoch 5: val_loss improved from 0.00002 to 0.00002, saving model to autoencoder_fraud.h5
3554/3554 [==============================] - 8s 2ms/step - loss: 1.9294e-05 - accuracy: 0.0747 - val_loss: 1.9963e-05 - val_acc
uracy: 0.0363
Epoch 6/50
3546/3554 [============================>.] - ETA: 0s - loss: 1.8817e-05 - accuracy: 0.0893
Epoch 6: val_loss improved from 0.00002 to 0.00002, saving model to autoencoder_fraud.h5
3554/3554 [==============================] - 7s 2ms/step - loss: 1.8812e-05 - accuracy: 0.0894 - val_loss: 1.9028e-05 - val_acc
uracy: 0.2298
Epoch 7/50
3536/3554 [============================>.] - ETA: 0s - loss: 1.8201e-05 - accuracy: 0.1393
Epoch 7: val_loss did not improve from 0.00002
```

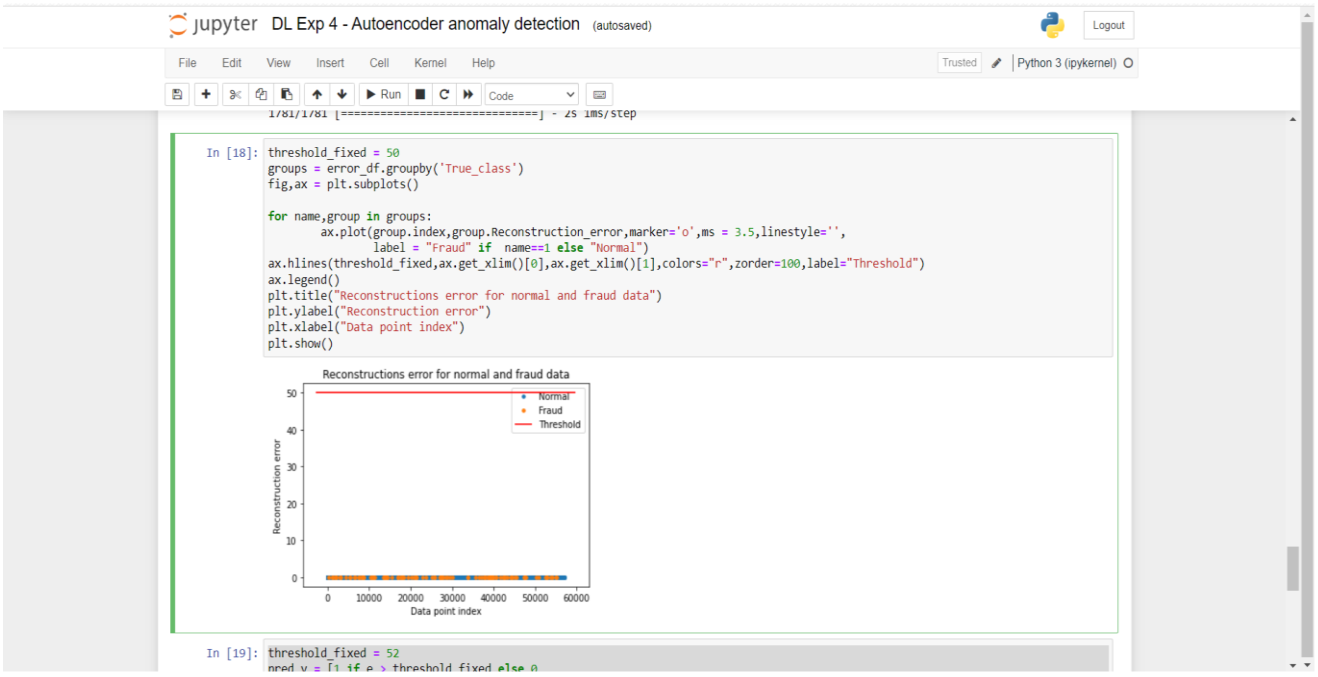
Result


jupyter DL Exp 4 - Autoencoder anomaly detection (autosaved)
Logout
File Edit View Insert Cell Kernel Help
Trusted
Python 3 (ipykernel)
Run
Code

```
Epoch 11: early stopping

In [16]: plt.plot(history['loss'],linewidth = 2,label = 'Train')
         plt.plot(history['val_loss'],linewidth = 2,label = 'Test')
         plt.legend(loc='upper right')
         plt.title('Model Loss')
         plt.ylabel('Loss')
         plt.xlabel('Epoch')

         #plt.ylim(ymin=0.70,ymax=1)

         plt.show()
```



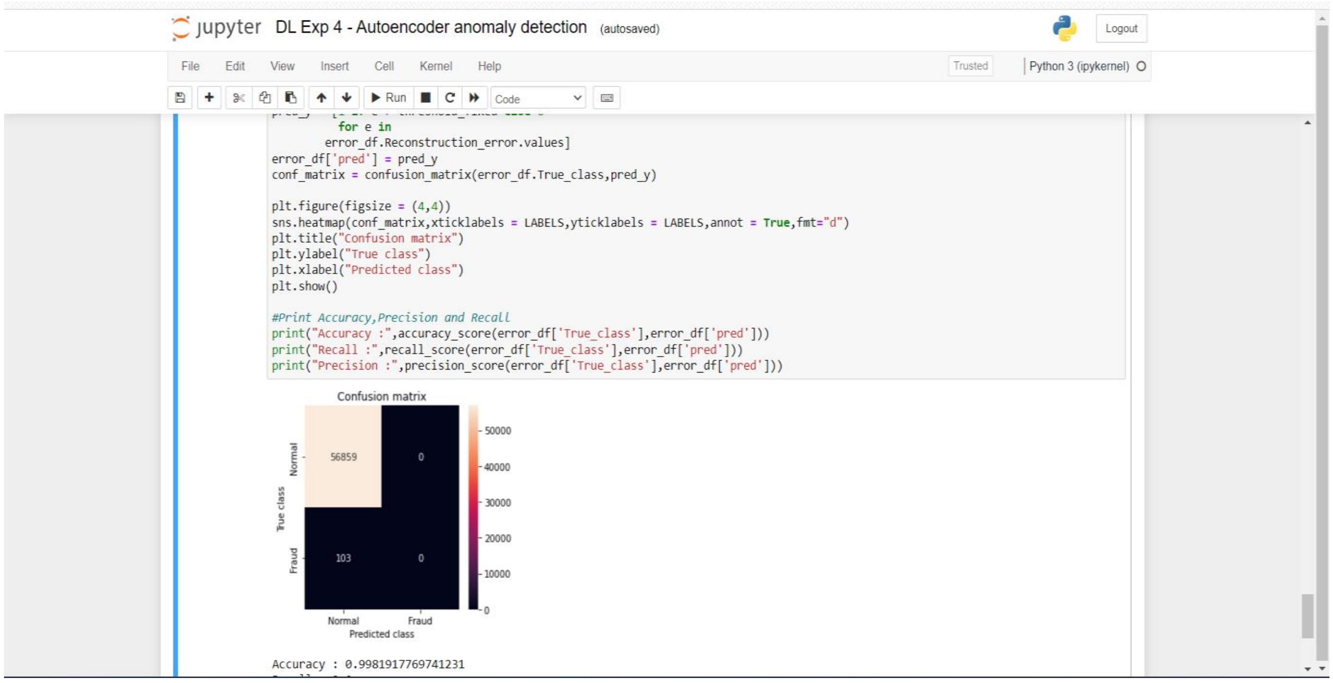
```
In [17]: test_x_predictions = autoencoder.predict(test_data)
         mse = np.mean(np.power(test_data - test_x_predictions, 2),axis = 1)
         error_df = pd.DataFrame({'Reconstruction_error':mse,
                                  'True_class':test_labels})
```


jupyter DL Exp 4 - Autoencoder anomaly detection (autosaved)
Logout
File Edit View Insert Cell Kernel Help
Trusted
Python 3 (ipykernel)
Run
Code
Reconstructions error for normal and fraud data
Normal
Fraud
Threshold
Reconstruction error
Data point index

```
1781/1781 [==============================] - 2s 1ms/step
```

```
In [18]: threshold_fixed = 50
         groups = error_df.groupby('True_class')
         fig,ax = plt.subplots()

         for name,group in groups:
                 ax.plot(group.index,group.Reconstruction_error,marker='o',ms = 3.5,linestyle='',
                         label = "Fraud" if  name==1 else "Normal")
         ax.hlines(threshold_fixed,ax.get_xlim()[0],ax.get_xlim()[1],colors="r",zorder=100,label="Threshold")
         ax.legend()
         plt.title("Reconstructions error for normal and fraud data")
         plt.ylabel("Reconstruction error")
         plt.xlabel("Data point index")
         plt.show()
```

```
In [19]: threshold_fixed = 52
         pred_y = [1 if e > threshold_fixed else 0
```



```
                 for e in
                error_df.Reconstruction_error.values]
error_df['pred'] = pred_y
conf_matrix = confusion_matrix(error_df.True_class,pred_y)

plt.figure(figsize = (4,4))
sns.heatmap(conf_matrix,xticklabels = LABELS,yticklabels = LABELS,annot = True,fmt="d")
plt.title("Confusion matrix")
plt.ylabel("True class")
plt.xlabel("Predicted class")
plt.show()

#Print Accuracy,Precision and Recall
print("Accuracy :",accuracy_score(error_df['True_class'],error_df['pred']))
print("Recall :",recall_score(error_df['True_class'],error_df['pred']))
print("Precision :",precision_score(error_df['True_class'],error_df['pred']))
```

```
Accuracy : 0.9981917769741231
```

Conclusion:

The detection Accuracy: 0.9981917769741231 is achieved using autoencoder and Recall: 0.0 and Precision: 0.0 respectively.

FAQ:

1) What is Anomaly Detection?
2) What are Autoencoders in Deep Learning?
3) Enlist different applications with Autoencoders in DL.
4) Enlist different types of anomaly detection Algorithms.
5) What is the difference between Anomaly detection and Novelty Detection.
6) Explain different blocks and working of Autoencoders.
7) What are reconstruction and Reconstruction errors?
8) What is Minmaxscaler from sklearn.
8) Explain. train_test_split from sklearn.
9) What are anomaly scores?
10) Explain TensorFlow dataset.
11) Describe the ECG Dataset.
12) Explain Keras Optimizers
13) Explain Keras layers dense and dropouts
14) Explain Keras losses and mean squared logarithmic error
15) Explain Relu activation function

ASSIGNMENT NO.6

Title: Implement the Continuous Bag of Words (CBOW) Model.

Aim: Implement the Continuous Bag of Words (CBOW) Model. Stages can be:

a. Data preparation

b. Generate training data

c. Train model

d. Output

Theory:

NLP

Natural language processing (NLP) refers to the branch of computer science—and more specifically, the branch of artificial intelligence or AI—concerned with giving computers the ability to understand text and spoken words in much the same way human beings can.

NLP combines computational linguistics—rule-based modeling of human language—with statistical, machine learning, and deep learning models. Together, these technologies enable computers to process human language in the form of text or voice data and to 'understand' its full meaning, complete with the speaker or writer's intent and sentiment.

NLP drives computer programs that translate text from one language to another, respond to spoken commands, and summarize large volumes of text rapidly—even in real time. There's a good chance you've interacted with NLP in the form of voice-operated GPS systems, digital assistants, speech-to-text dictation software, customer service chatbots, and other consumer conveniences. But NLP also plays a growing role in enterprise solutions that help streamline business operations, increase employee productivity, and simplify mission-critical business processes.

Word Embeddings

It is an approach for representing words and documents. Word Embedding or Word Vector is a numeric vector input that represents a word in a lower-dimensional space. It allows words with similar meaning to have a similar representation. They can also approximate meaning. A word vector with 50 values can represent 50 unique features.

Features: Anything that relates words to one another. E.g.: Age, Sports, Fitness, Employed etc. Each word vector has values corresponding to these features.

- Goal of Word Embeddings
- To reduce dimensionality
- To use a word to predict the words around
- Inter word semantics must be captured
- They are used as input to machine learning models.

Take the words —-> Give their numeric representation —-> Use in training or inference
To represent or visualize any underlying patterns of usage in the corpus that was used to train them.

Word2Vec techniques.
Word2Vec creates vectors of the words that are distributed numerical representations of word features – these word features could comprise of words that represent the context of the individual words present in our vocabulary. Word embeddings eventually help in establishing the association of a word with another similar meaning word through the created vectors.

As seen in the image below where word embeddings are plotted, similar meaning words are closer in space, indicating their semantic similarity.

Two different model architectures that can be used by Word2Vec to create the word embeddings are the Continuous Bag of Words (CBOW) model & the Skip-Gram model.

Applications of Word embedding in NLP
i. Text summarization: extractive or abstractive text summarization.
ii. Sentiment Analysis.
iii. Translating from one language to another: neural machine translation.
iv. Chatbots.

CBOW architecture.
Even though Word2Vec is an unsupervised model where you can give a corpus without any label information and the model can create dense word embeddings, Word2Vec internally leverages a supervised classification model to get these embeddings from the corpus.
The CBOW architecture comprises a deep learning classification model in which we take in context words as input, X, and try to predict our target word, Y.
For example, if we consider the sentence – "Word2Vec has a deep learning model working in the backend.", there can be pairs of context words and target (center) words. If we consider a context window size of 2, we will have pairs like ([deep, model], learning), ([model, in], working), ([a, learning), deep) etc. The deep learning model would try to predict these target words based on the context words.

The word embeddings are then passed to a lambda layer where we average out the word embeddings.
We then pass these embeddings to a dense SoftMax layer that predicts our target word. We match this with our target word and compute the loss and then we perform backpropagation with each epoch to update the embedding layer in the process.

We can extract out the embeddings of the needed words from our embedding layer, once the training is completed.

Input to CBOW model and Output to CBW model.
Input will contain the sum of one-hot encoded vectors of the context words within the window size. The size of the input will be n x 1.
The output shows the words that are most similar to the word 'virus' along with the sequence or degree of similarity. The words like symptoms and incubation are contextually very accurate with the word virus which proves that CBOW model successfully understands the context of the data.

Tokenizer
Tokenization is the first step in any NLP pipeline. It has an important effect on the rest of your pipeline. A tokenizer breaks unstructured data and natural language text into chunks of information that can be considered as discrete elements.

Window size parameters for the CBOW model.
The output shows the words that are most similar to the word 'virus' along with the sequence or degree of similarity. The words like symptoms and incubation are contextually very accurate with the word virus which proves that CBOW model successfully understands the context of the data.
So, the output of the hidden layer in the CBOW architecture is the average of all the context word vectors.

Embedding and Lambda layer from Keras
Embedding layer is one of the available layers in Keras. This is mainly used in Natural Language Processing related applications such as language modeling, but it can also be used with other tasks that involve neural networks. While dealing with NLP problems, we can use pre-trained word embeddings such as GloVe. Alternatively, we can also train our own embeddings using Keras embedding layer.
Lambda is used to transform the input data using an expression or function. For example, if Lambda with expression lambda x: x ** 2 is applied to a layer, then its input data will be squared before processing.

yield ()
Yield represents the income an investment generates and is usually expressed as a percentage. But be careful not to confuse yield with return. Return on investment (ROI) is typically considered profit and loss, such as capital gains.

Steps/ Algorithm

1. Dataset link and libraries:

Create any English 5 to 10 sententece paragraph as input Import following data from keras :

keras.models import Sequential

keras.layers import Dense, Embedding, Lambda

keras.utils import np_utils

keras.preprocessing import sequence

keras.preprocessing.text import Tokenizer

Import Gensim for NLP operations : requirements :

Gensim runs on Linux, Windows and Mac OS X, and should run on any other platform that supports Python 3.6+ and NumPy. Gensim depends on the following software: Python, tested with versions 3.6, 3.7 and 3.8. NumPy for number crunching.

a) Import following libraries gemsim and numpy set i.e. text file created . It should be preprocessed.

b) Tokenize every word from the paragraph . You can call in built tokenizer present in Gensim

c) Fit the data to tokenizer

d) Find total no of words and total no of sentences.

e) Generate the pairs of Context words and target words :

e.g. cbow_model(data, window_size, total_vocab): total_length = window_size*2

for text in data: text_len = len(text)

for idx, word in enumerate(text): context_word = []

target = []

begin = idx - window_size end = idx + window_size + 1

context_word.append([text[i] for i in range(begin, end) if 0 <= i < text_len and i
!= idx])

target.append(word)

contextual = sequence.pad_sequences(context_word, total_length=total_length) final_target =

np_utils.to_categorical(target, total_vocab)

yield(contextual, final_target)

f) Create a Neural Network model with the following parameters . Model type : sequential

Layers : Dense , Lambda , embedding. Compile Options : (loss='categorical_crossentropy', optimizer='adam')

g) Create vector file of some word for testing e.g.:dimensions=100

vect_file = open('/content/gdrive/My Drive/vectors.txt' ,'w') vect_file.write('{}

{}\n'.format(total_vocab,dimensions)

h) Assign weights to your trained model
e.g. weights = model.get_weights()[0] for text, i in vectorize.word_index.items():

final_vec = ' '.join(map(str, list(weights[i, :]))) vect_file.write('{} {}\n'.format(text, final_vec)

Close()

i) Use the vectors created in Gensim
e.g. cbow_output = gensim.models.KeyedVectors.load_word2vec_format('/content/gdrive/My Drive/vectors.txt', binary=False)

j) choose the word to get similar type of words: cbow_output.most_similar(positive=['Your word'])

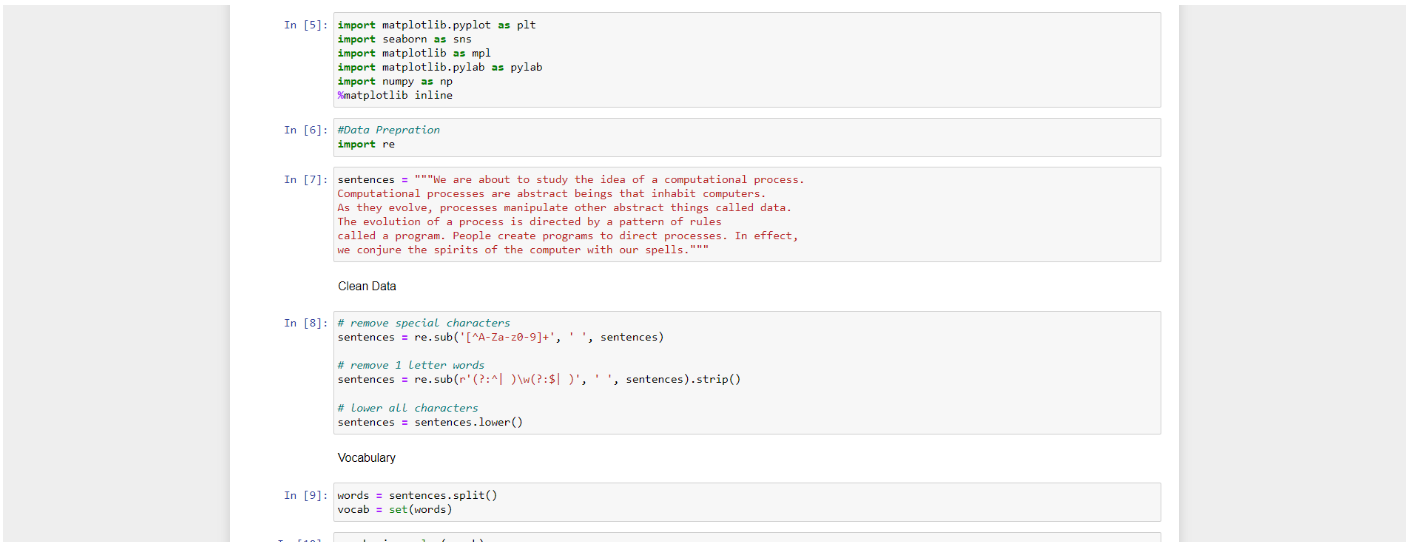
CODE

```
import matplotlib.pyplot as plt
import seaborn as sns
import matplotlib as mpl
import matplotlib.pylab as pylab
import numpy as np
%matplotlib inline
#Data Prepration
import re
sentences = """We are about to study the idea of a computational process.
Computational processes are abstract beings that inhabit computers.
As they evolve, processes manipulate other abstract things called data.
The evolution of a process is directed by a pattern of rules
called a program. People create programs to direct processes. In effect,
we conjure the spirits of the computer with our spells."""

# remove special characters
sentences = re.sub('[^A-Za-z0-9]+', ' ', sentences)

# remove 1 letter words
sentences = re.sub(r'(?:^| )\w(?:$| )', ' ', sentences).strip()

# lower all characters
sentences = sentences.lower()

words = sentences.split()
vocab = set(words)

vocab_size = len(vocab)
embed_dim = 10
context_size = 2

word_to_ix = {word: i for i, word in enumerate(vocab)}
ix_to_word = {i: word for i, word in enumerate(vocab)}

# data - [(context), target]

data = []
for i in range(2, len(words) - 2):
    context = [words[i - 2], words[i - 1], words[i + 1], words[i + 2]]
    target = words[i]
    data.append((context, target))
print(data[:5])

embeddings =  np.random.random_sample((vocab_size, embed_dim))

def linear(m, theta):
    w = theta
    return m.dot(w)

def log_softmax(x):
    e_x = np.exp(x - np.max(x))
    return np.log(e_x / e_x.sum())
```

```
def NLLLoss(logs, targets):
    out = logs[range(len(targets)), targets]
    return -out.sum()/len(out)

def log_softmax_crossentropy_with_logits(logits,target):

    out = np.zeros_like(logits)
    out[np.arange(len(logits)),target] = 1

    softmax = np.exp(logits) / np.exp(logits).sum(axis=-1,keepdims=True)

    return (- out + softmax) / logits.shape[0]

def forward(context_idxs, theta):
    m = embeddings[context_idxs].reshape(1, -1)
    n = linear(m, theta)
    o = log_softmax(n)

    return m, n, o

def backward(preds, theta, target_idxs):
    m, n, o = preds

    dlog = log_softmax_crossentropy_with_logits(n, target_idxs)
    dw = m.T.dot(dlog)

    return dw

def optimize(theta, grad, lr=0.03):
    theta -= grad * lr
    return theta

#Genrate training data

theta = np.random.uniform(-1, 1, (2 * context_size * embed_dim, vocab_size))

epoch_losses = {}

for epoch in range(80):

    losses =  []

    for context, target in data:
        context_idxs = np.array([word_to_ix[w] for w in context])
        preds = forward(context_idxs, theta)

        target_idxs = np.array([word_to_ix[target]])
        loss = NLLLoss(preds[-1], target_idxs)

        losses.append(loss)

        grad = backward(preds, theta, target_idxs)
        theta = optimize(theta, grad, lr=0.03)
    epoch_losses[epoch] = losses

ix = np.arange(0,80)

fig = plt.figure()
fig.suptitle('Epoch/Losses', fontsize=20)
plt.plot(ix,[epoch_losses[i][0] for i in ix])
plt.xlabel('Epochs', fontsize=12)
plt.ylabel('Losses', fontsize=12)

def predict(words):
    context_idxs = np.array([word_to_ix[w] for w in words])
    preds = forward(context_idxs, theta)
    word = ix_to_word[np.argmax(preds[-1])]

    return word

# (['we', 'are', 'to', 'study'], 'about')
predict(['we', 'are', 'to', 'study'])

def accuracy():
    wrong = 0

    for context, target in data:
        if(predict(context) != target):
            wrong += 1

    return (1 - (wrong / len(data)))

accuracy()

predict(['processes', 'manipulate', 'things', 'study'])
```

OUTPUT:

a. Data preparation



```
In [5]: import matplotlib.pyplot as plt
        import seaborn as sns
        import matplotlib as mpl
        import matplotlib.pylab as pylab
        import numpy as np
        %matplotlib inline

In [6]: #Data Prepration
        import re

In [7]: sentences = """We are about to study the idea of a computational process.
        Computational processes are abstract beings that inhabit computers.
        As they evolve, processes manipulate other abstract things called data.
        The evolution of a process is directed by a pattern of rules
        called a program. People create programs to direct processes. In effect,
        we conjure the spirits of the computer with our spells."""
```

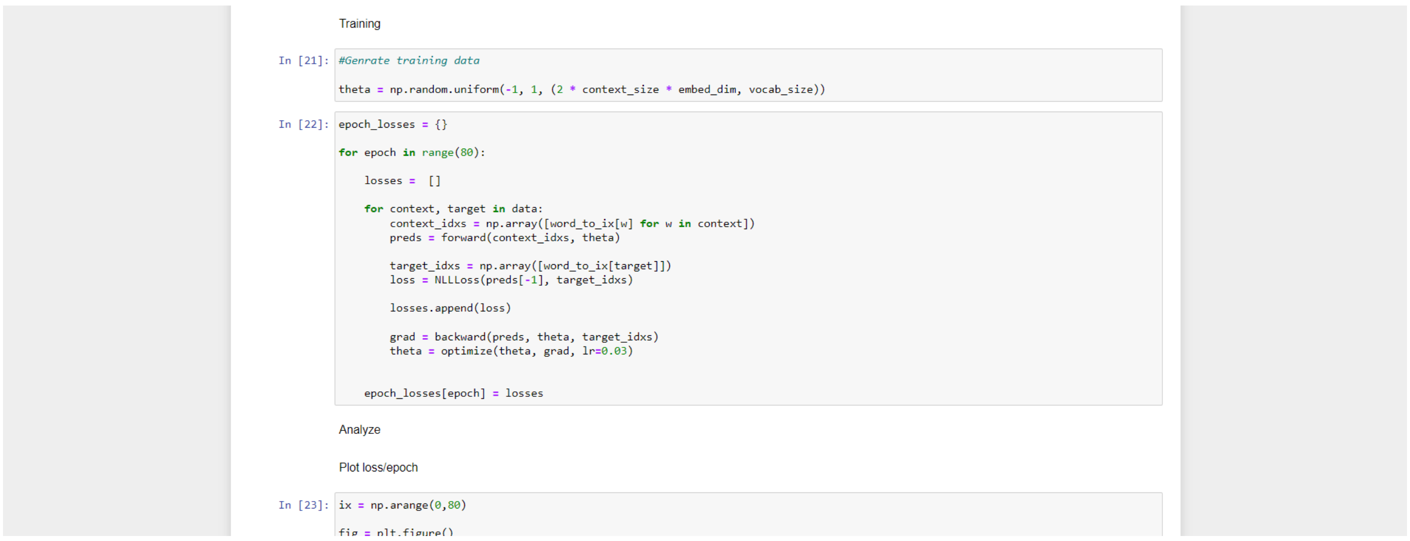
Clean Data

```
In [8]: # remove special characters
        sentences = re.sub('[^A-Za-z0-9]+', ' ', sentences)

        # remove 1 letter words
        sentences = re.sub(r'(?:^| )\w(?:$| )', ' ', sentences).strip()

        # lower all characters
        sentences = sentences.lower()
```

Vocabulary

```
In [9]: words = sentences.split()
        vocab = set(words)
```

b. Generate training data



Training

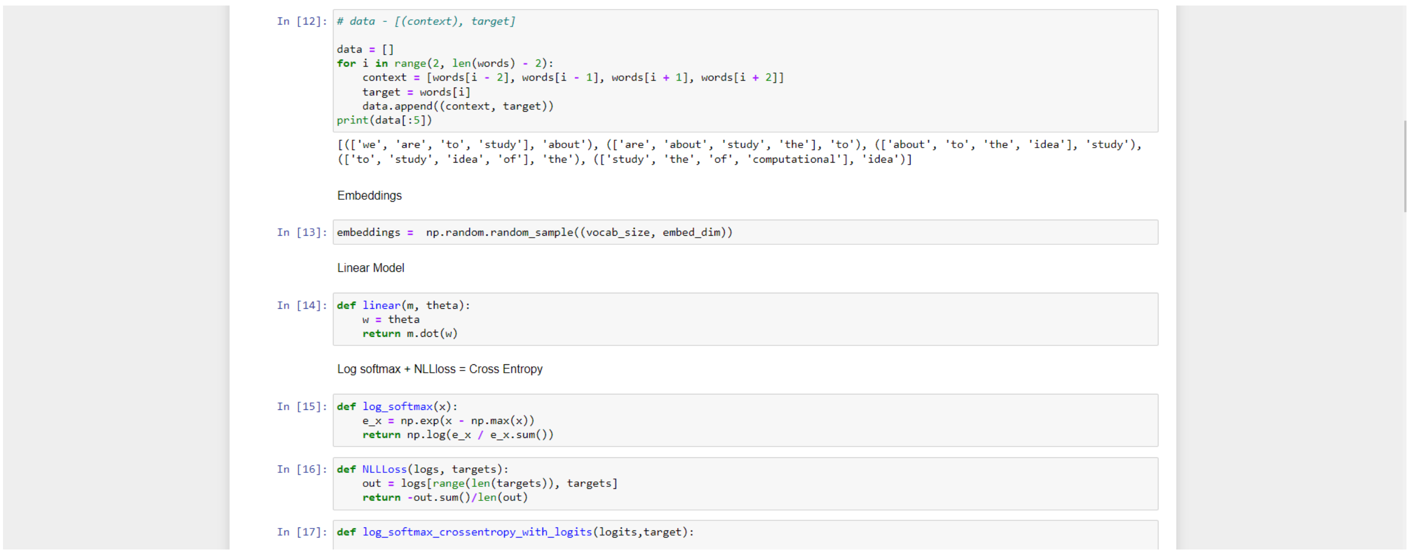
```
In [21]: #Genrate training data

         theta = np.random.uniform(-1, 1, (2 * context_size * embed_dim, vocab_size))

In [22]: epoch_losses = {}

         for epoch in range(80):

             losses =  []

             for context, target in data:
                 context_idxs = np.array([word_to_ix[w] for w in context])
                 preds = forward(context_idxs, theta)

                 target_idxs = np.array([word_to_ix[target]])
                 loss = NLLLoss(preds[-1], target_idxs)

                 losses.append(loss)

                 grad = backward(preds, theta, target_idxs)
                 theta = optimize(theta, grad, lr=0.03)


             epoch_losses[epoch] = losses
```

Analyze

Plot loss/epoch

```
In [23]: ix = np.arange(0,80)
```

c. Train model



```
In [12]: # data - [(context), target]

         data = []
         for i in range(2, len(words) - 2):
             context = [words[i - 2], words[i - 1], words[i + 1], words[i + 2]]
             target = words[i]
             data.append((context, target))
         print(data[:5])

         [(['we', 'are', 'to', 'study'], 'about'), (['are', 'about', 'study', 'the'], 'to'), (['about', 'to', 'the', 'idea'], 'study'),
         (['to', 'study', 'idea', 'of'], 'the'), (['study', 'the', 'of', 'computational'], 'idea')]
```

Embeddings

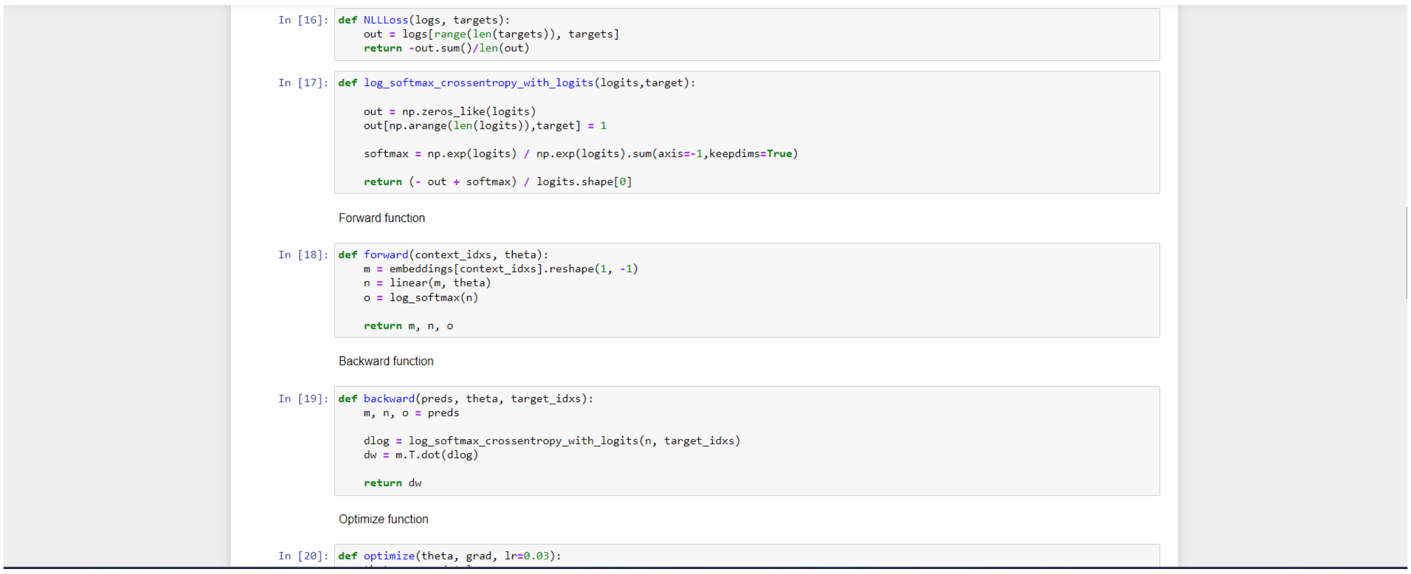
```
In [13]: embeddings =  np.random.random_sample((vocab_size, embed_dim))
```

Linear Model

```
In [14]: def linear(m, theta):
             w = theta
             return m.dot(w)
```

Log softmax + NLLloss = Cross Entropy

```
In [15]: def log_softmax(x):
             e_x = np.exp(x - np.max(x))
             return np.log(e_x / e_x.sum())

In [16]: def NLLLoss(logs, targets):
             out = logs[range(len(targets)), targets]
             return -out.sum()/len(out)

In [17]: def log_softmax_crossentropy_with_logits(logits,target):
```



```
In [16]: def NLLLoss(logs, targets):
             out = logs[range(len(targets)), targets]
             return -out.sum()/len(out)

In [17]: def log_softmax_crossentropy_with_logits(logits,target):

             out = np.zeros_like(logits)
             out[np.arange(len(logits)),target] = 1

             softmax = np.exp(logits) / np.exp(logits).sum(axis=-1,keepdims=True)

             return (- out + softmax) / logits.shape[0]
```

Forward function

```
In [18]: def forward(context_idxs, theta):
             m = embeddings[context_idxs].reshape(1, -1)
             n = linear(m, theta)
             o = log_softmax(n)

             return m, n, o
```

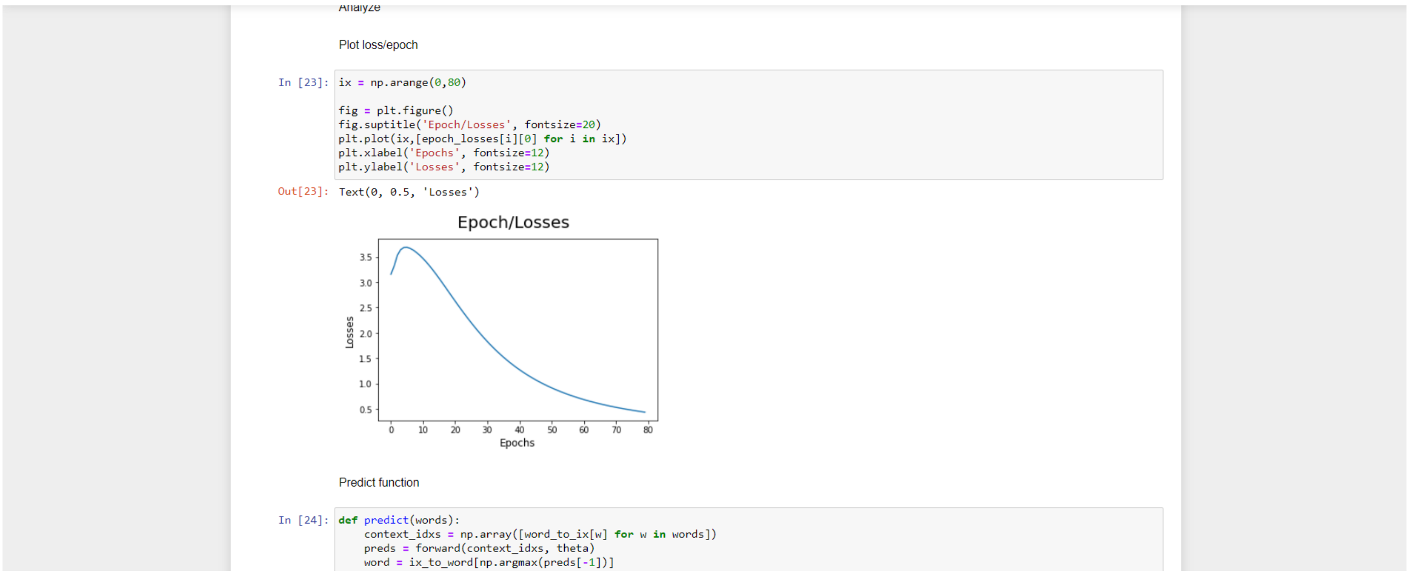
Backward function

```
In [19]: def backward(preds, theta, target_idxs):
             m, n, o = preds

             dlog = log_softmax_crossentropy_with_logits(n, target_idxs)
             dw = m.T.dot(dlog)

             return dw
```

Optimize function

```
In [20]: def optimize(theta, grad, lr=0.03):
```

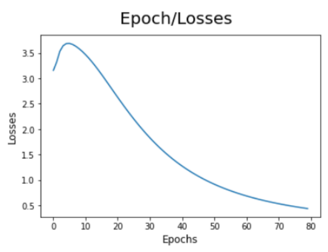
d. Output



Analyze

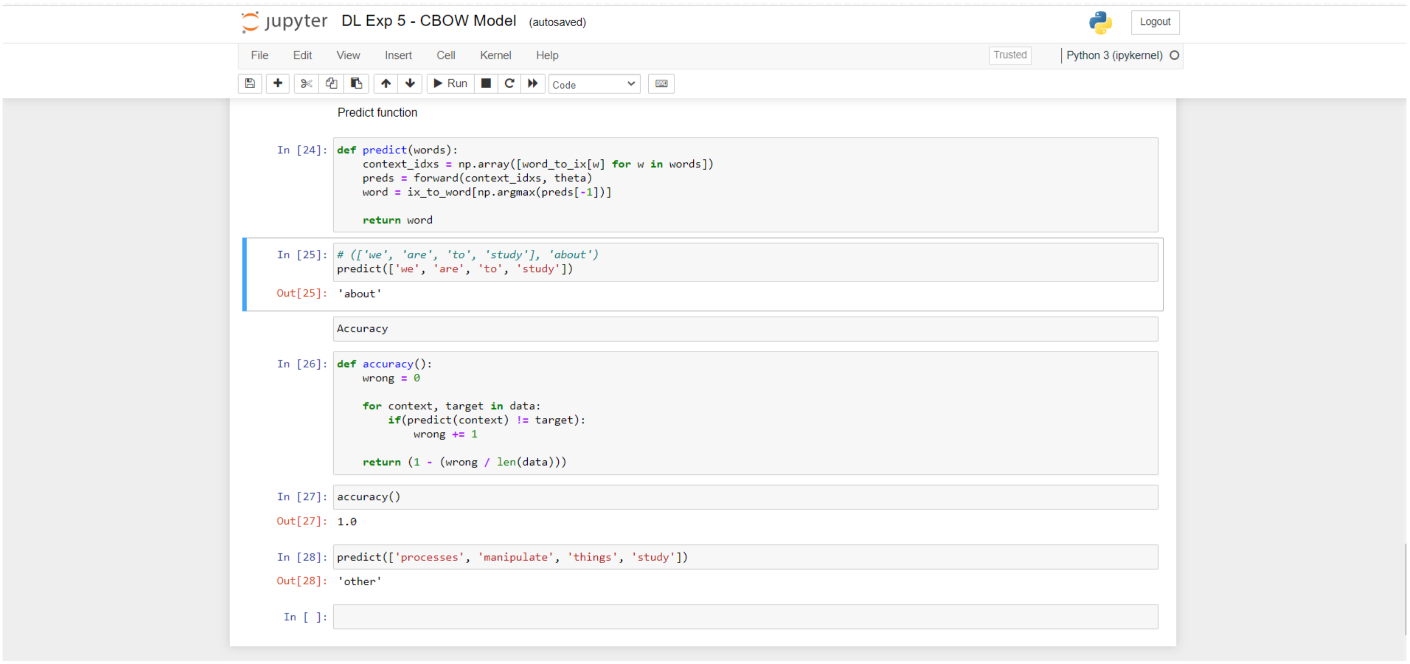
Plot loss/epoch

```
In [23]: ix = np.arange(0,80)

         fig = plt.figure()
         fig.suptitle('Epoch/Losses', fontsize=20)
         plt.plot(ix,[epoch_losses[i][0] for i in ix])
         plt.xlabel('Epochs', fontsize=12)
         plt.ylabel('Losses', fontsize=12)

Out[23]: Text(0, 0.5, 'Losses')
```



Predict function

```
In [24]: def predict(words):
             context_idxs = np.array([word_to_ix[w] for w in words])
             preds = forward(context_idxs, theta)
             word = ix_to_word[np.argmax(preds[-1])]
```



Predict function

```
In [24]: def predict(words):
             context_idxs = np.array([word_to_ix[w] for w in words])
             preds = forward(context_idxs, theta)
             word = ix_to_word[np.argmax(preds[-1])]

             return word

In [25]: # (['we', 'are', 'to', 'study'], 'about')
         predict(['we', 'are', 'to', 'study'])

Out[25]: 'about'
```

Accuracy

```
In [26]: def accuracy():
             wrong = 0

             for context, target in data:
                 if(predict(context) != target):
                     wrong += 1

             return (1 - (wrong / len(data)))

In [27]: accuracy()

Out[27]: 1.0

In [28]: predict(['processes', 'manipulate', 'things', 'study'])

Out[28]: 'other'

In [ ]:
```

Conclusion

The CBOW model tries to understand the context of the words and takes this as input. It then tries to predict words that are contextually accurate with accuracy 1.0

FAQs:

1. What is NLP?
2. What is Word embedding related to NLP?
3. Explain Word2Vec techniques.
4. Enlist applications of Word embedding in NLP.
5. Explain CBOW architecture.
6. What will be input to CBOW model and Output to CBW model.
7. What is Tokenizer?
8. Explain window size parameters in detail for the CBOW model.
9. Explain Embedding and Lambda layer from Keras
10. What is yield ()

ASSIGNMENT NO.7

Title: Object detection using Transfer Learning of CNN architectures

Aim: Object detection using Transfer Learning of CNN architectures

a. Load in a pre-trained CNN model trained on a large dataset
b. Freeze parameters (weights) in model's lower convolutional layers
c. Add custom classifier with several layers of trainable parameters to model
d. Train classifier layers on training data available for task
e. Fine-tune hyper parameters and unfreeze more layers as needed

Theory:

Transfer Learning:

Transfer learning, used in machine learning, is the reuse of a pre-trained model on a new problem. In transfer learning, a machine exploits the knowledge gained from a previous task to improve generalization about another. For example, in training a classifier to predict whether an image contains food, you could use the knowledge it gained during training to recognize drinks.

Pre-trained Neural Network models:

In Machine Learning, a pre-trained model falls under the category of transfer learning. Pre-Trained models are machine learning models that are trained, developed and made available by other developers. They are generally used to solve problems based on deep learning and are always trained on a very large dataset.

They are made available by developers who want to contribute to the machine learning community to solve a similar problem. The way so many developers contribute to the community by creating frameworks and packages the same way some developers also contribute with a machine learning model which is known as a Pre-Trained model. These models are generally made for solving very complex and common problems. LeNet-5, AlexNet, GoogLeNet are some of the popular pre-trained models.

Examples of a Pre-Trained Model:

Most of the pre-trained models that have been built and made available till now are based on convolutional neural networks. Below are some of the examples of such models that you can use:

LeNet-5: This is one of the most widely used convolutional neural network architectures. It was created by Yann LeCunn in 1998 and it is very much used for the task of recognizing handwritten digits.

AlexNet: It won the 2012 ImageNet challenge by a very large margin. It achieved a top-five error rate of 17% where the second-best achieved an error rate of 26%. It is very similar to LeNet-5.

GoogLeNet: It was developed by Christian Szegedy et al. from Google Research. This model has 10 times fewer parameters than AlexNet, roughly around 6 million instead of 60 million.

Pytorch Library:

PyTorch is an optimized tensor library primarily used for Deep Learning applications using GPUs and CPUs. It is an open-source machine learning library for Python, mainly developed by the Facebook AI Research team. It is one of the widely used Machine learning libraries, others being TensorFlow and Keras.

Advantages of Transfer Learning:

There are many advantages for using transfer learning beyond the potential savings of time and energy. One key advantage exists around the availability of a sufficient labeled training set for your problem domain. When insufficient training data exists, an existing model (from a related problem domain) can be used with additional training to support the new problem domain.

As discussed in feature transfer, a deep learning model implements feature extraction and classification with a smaller neural network topology. Depending upon the problem domain, the outputs (or classification) will commonly differ between two problems. For this reason, the classification layer is commonly replaced and reconstructed for the new problem domain. This requires significantly less resources to train and validate while exploiting the pre-trained feature extraction pipeline.

Applications of Transfer Learning:

The following list describes some of the applications of Transfer Learning:

i. Text and Image Classification
ii. Training the self-driving vehicles using simulations
iii. Robot training
iv. Medical Image Analysis
v. AI Games
vi. Sentiment Analysis
vii. Identifying and filtering spam emails
viii. Speech recognition

- **Caltech 101 images dataset:**

The Caltech 101 dataset contains 101 categories of objects for recognition algorithms. Each category has about 40 to 800 images. The dataset contains 50 categories on average per category. The images in this dataset are roughly 300 x 200 pixels.

Imagenet Dataset:

ImageNet is a large database or dataset of over 14 million images. It was designed by academics intended for computer vision research. It was the first of its kind in terms of scale. Images are organized and labelled in a hierarchy.

Basic steps for Transfer Learning:

The twelve key learning steps for transfer learning are as follows:

1. Import required libraries.
2. Load appropriate datasets.
3. Split the data in three datasets: training, testing and validation.
4. One-hot encoding the labels.
5. Data augmentation.
6. Create an instance for the base model.
7. Build the new model by defining and adding layers.
8. Train the new model with data.
9. Plot graphs like training accuracy and validation accuracy.
10. Make predictions.
11. Plot the confusion matrices.

Data Augmentation:

Data augmentation is a set of techniques used to increase the amount of data in a machine learning model by adding slightly modified copies of already existing data or newly created synthetic data from existing data. It helps smooth out the machine learning model and reduce the overfitting of data.

Data Augmentation Important

Collecting and labeling data is a tedious and costly process in machine learning models. Data augmentation can transform into datasets that help organizations to reduce operational costs. At the same time, it solves the problem of limited dataset size and limited data variation. This improves the overall performance of the model in various scenarios.

Working

Based on the type of dataset, different data augmentation techniques can be used. There are many data augmentation techniques available for the image/video, audio, and text data. We will explore the image/video data augmentation method in detail.

Need of preprocessing on input data in Transfer learning

Sometimes we are dealing with a dataset that is not used or tuned before with any machine learning model and It comes from the real world, where data and their insights are changing regularly. Here data scientists or data engineers will need to understand and find insights from data and make something that can integrate with real-world data.

Machine learning models do not know what real data is, they only know about machine language. So, in that case, the engineer will need to refine and clean data in a way, so that It can be trainable with machine learning models, and then easily usable in the real world.

PyTorch Transforms module.Explain following commands w.r.t
it : Compose([
RandomResizedCrop(size=256, scale=(0.8, 1.0)), RandomRotation(degrees=15),
ColorJitter(), RandomHorizontalFlip(),
CenterCrop(size=224), # Image net standards
.ToTensor(), Normalize

PyTorch uses modules to represent neural networks.

Modules are:

Building blocks of stateful computation. PyTorch provides a robust library of modules and makes it simple to define new custom modules, allowing for easy construction of elaborate, multi-layer neural networks.

Tightly integrated with PyTorch's autograd system. Modules make it simple to specify learnable parameters for PyTorch's Optimizers to update.

Easy to work with and transform. Modules are straightforward to save and restore, transfer between CPU / GPU / TPU devices, prune, quantize, and more.

Validation Transforms steps with Pytorch Transforms:

Training Neural Network with Validation

The training step in PyTorch is almost identical almost every time you train it. But before implementing that let's learn about 2 modes of the model object:-

Training Mode: Set by model.train(), it tells your model that you are training the model. So layers like dropout etc. which behave differently while training and testing can behave accordingly.

Evaluation Mode: Set by model.eval(), it tells your model that you are testing the model.

VGG-16 model from Pytorch

VGG-16 mainly has three parts: convolution, Pooling, and fully connected layers.

Convolution layer- In this layer, filters are applied to extract features from images. The most important parameters are the size of the kernel and stride.

Pooling layer- Its function is to reduce the spatial size to reduce the number of parameters and computation in a network.

Fully Connected- These are fully connected connections to the previous layers as in a simple neural network.

Steps/ Algorithm

1. Dataset link and libraries:

https://data.caltech.edu/records/mzrjq-6wc02

separate the data into training, validation, and testing sets with a 50%, 25%, 25% split and then structured the directories as follows:

/datadir

/train

/class1

/class2

.

.

/valid

/class1

/class2

.

.

/test

/class1

/class2

Libraries required :

PyTorch

torchvision import transforms

torchvision import datasets

torch.utils.data import DataLoader

torchvision import models torch.nn as nn

torch import optim

m) Prepare the dataset in splitting in three directories Train , validation and test with 50 25 25

n) Do pre-processing on data with transform from Pytorch Training

dataset transformation as follows : transforms.Compose([

transforms.RandomResizedCrop(size=256, scale=(0.8, 1.0)),

transforms.RandomRotation(degrees=15), transforms.ColorJitter(),

transforms.RandomHorizontalFlip(), transforms.CenterCrop(size=224), #

Image net standards transforms.ToTensor(),

transforms.Normalize([0.485, 0.456, 0.406],

[0.229, 0.224, 0.225]) # Imagenet standards Validation Dataset transform as follows : transforms.Compose([

transforms.Resize(size=256),

transforms.CenterCrop(size=224),

transforms.ToTensor(),

transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])

o) Create Datasets and Loaders : data = {

'train':(Our name given to train data set dir created) datasets.ImageFolder(root=traindir, transform=image_transforms['train']), 'valid':

datasets.ImageFolder(root=validdir, transform=image_transforms['valid']),

}

dataloaders = {

'train': DataLoader(data['train'], batch_size=batch_size, shuffle=True), 'val': DataLoader(data['valid'], batch_size=batch_size, shuffle=True)

}

p) Load Pretrained Model : from torchvision import models

model = model.vgg16(pretrained=True)

q) Freeze all the Models Weight

For param in model.parameters():

param.requires_grad = False

r) Add our own custom classifier with following parameters : Fully connected with ReLU activation, shape = (n_inputs, 256) Dropout with 40% chance of dropping

Fully connected with log softmax output, shape = (256, n_classes) import torch.nn as nn

# Add on classifier model.classifier[6] = nn.Sequential(

nn.Linear(n_inputs, 256), nn.ReLU(),

nn.Dropout(0.4), nn.Linear(256, n_classes),

nn.LogSoftmax(dim=1))

s) Only training the sixth layer of the classifier keep remaining layers off . Sequential(

(0): Linear(in_features=25088, out_features=4096, bias=True) (1): ReLU(inplace)

(2) : Dropout(p=0.5)

(3) : Linear(in_features=4096, out_features=4096, bias=True) (4): ReLU(inplace)

(5) : Dropout(p=0.5)

(6) : Sequential(

(0): Linear(in_features=4096, out_features=256, bias=True) (1): ReLU()

(2) : Dropout(p=0.4)

(3) : Linear(in_features=256, out_features=100, bias=True) (4): LogSoftmax()

)

)

t) Initialize the loss and optimizer criteration = nn.NLLLoss()

optimizer = optim.Adam(model.parameters())

u) Train the model using Pytorch for epoch in range(n_epochs): for data, targets in trainloader:

# Generate predictions out =

model(data)

# Calculate loss

loss = criterion(out, targets) #

Backpropagation

loss.backward()

# Update model parameters optimizer.step()

v) Perform Early stopping

w) Draw performance curve

x) Calculate Accuracy

pred = torch.max(ps, dim=1) equals = pred

== targets

# Calculate accuracy

accuracy = torch.mean(equals)

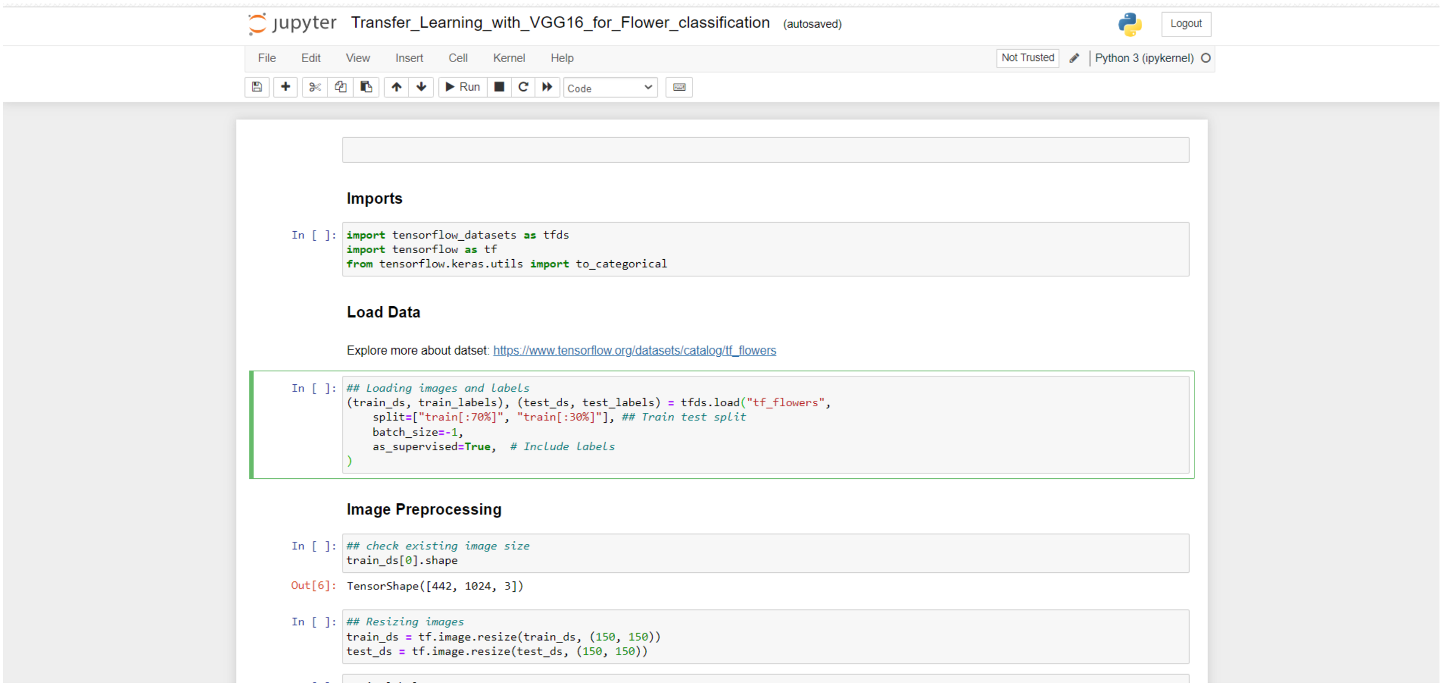
CODE

```
import tensorflow_datasets as tfds
import tensorflow as tf
from tensorflow.keras.utils import to_categorical
## Loading images and labels
(train_ds, train_labels), (test_ds, test_labels) = tfds.load("tf_flowers",
    split=["train[:70%]", "train[:30%]"], ## Train test split
    batch_size=-1,
    as_supervised=True,  # Include labels
)

## check existing image size
train_ds[0].shape

## Resizing images
train_ds = tf.image.resize(train_ds, (150, 150))
test_ds = tf.image.resize(test_ds, (150, 150))

train_labels

## Transforming labels to correct format
train_labels = to_categorical(train_labels, num_classes=5)
test_labels = to_categorical(test_labels, num_classes=5)

train_labels[0]

from tensorflow.keras.applications.vgg16 import VGG16
from tensorflow.keras.applications.vgg16 import preprocess_input

train_ds[0].shape

## Loading VGG16 model
base_model = VGG16(weights="imagenet", include_top=False, input_shape=train_ds[0].shape)

## will not train base mode
# Freeze Parameters in model's lower convolutional layers
base_model.trainable = False

## Preprocessing input
train_ds = preprocess_input(train_ds)
test_ds = preprocess_input(test_ds)

## model details
base_model.summary()

#add our layers on top of this model
from tensorflow.keras import layers, models

flatten_layer = layers.Flatten()
dense_layer_1 = layers.Dense(50, activation='relu')
dense_layer_2 = layers.Dense(20, activation='relu')
prediction_layer = layers.Dense(5, activation='softmax')
```

```
model = models.Sequential([
   base_model,
   flatten_layer,
   dense_layer_1,
   dense_layer_2,
   prediction_layer
])

from tensorflow.keras.callbacks import EarlyStopping

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

es = EarlyStopping(monitor='val_accuracy', mode='max', patience=5,  restore_best_weights=True)

history=model.fit(train_ds, train_labels, epochs=50, validation_split=0.2, batch_size=32, callbacks=[es])

los,accurac=model.evaluate(test_ds,test_labels)
print("Loss: ",los,"Accuracy: ", accurac)

import matplotlib.pyplot as plt
plt.plot(history.history['accuracy'])
plt.title('ACCURACY')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train'],loc='upper left')
plt.show()

import numpy as np
import pandas as pd
y_pred = model.predict(test_ds)
y_classes = [np.argmax(element) for element in y_pred]
#to_categorical(y_classes, num_classes=5)
#to_categorical(test_labels, num_classes=5)
print(y_classes[:10])
print("\nTest")
print(test_labels[:10])
```

OUTPUT

a. Load in a pre-trained CNN model trained on a large dataset



jupyter Transfer_Learning_with_VGG16_for_Flower_classification (autosaved)
Logout
File Edit View Insert Cell Kernel Help
Not Trusted
Python 3 (ipykernel)
Run
Code

Imports

```
In [ ]: import tensorflow_datasets as tfds
        import tensorflow as tf
        from tensorflow.keras.utils import to_categorical
```

Load Data

Explore more about datset: https://www.tensorflow.org/datasets/catalog/tf_flowers

```
In [ ]: ## Loading images and labels
        (train_ds, train_labels), (test_ds, test_labels) = tfds.load("tf_flowers",
            split=["train[:70%]", "train[:30%]"], ## Train test split
            batch_size=-1,
            as_supervised=True,  # Include labels
        )
```

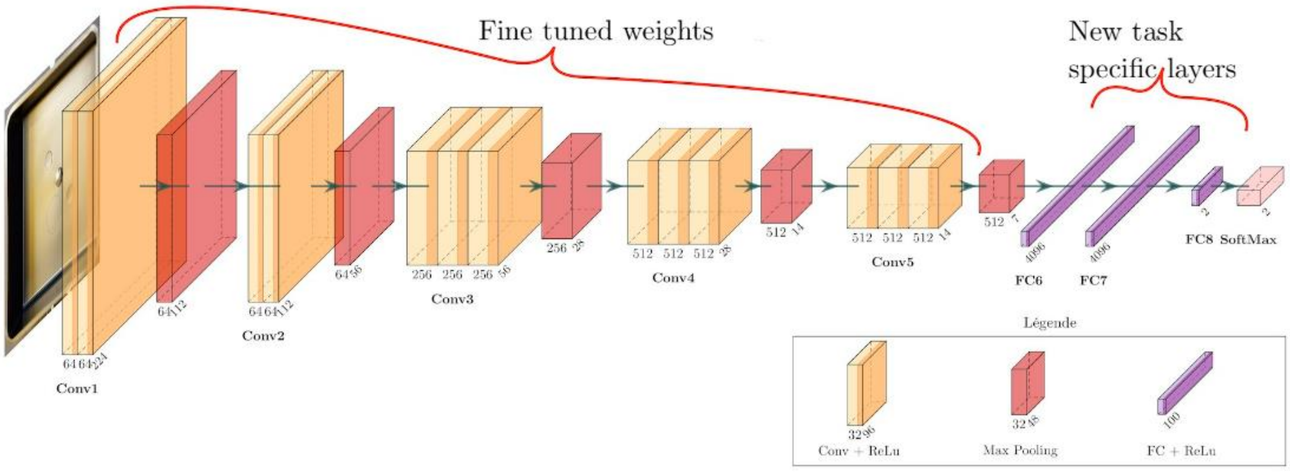
Image Preprocessing

```
In [ ]: ## check existing image size
        train_ds[0].shape

Out[6]: TensorShape([442, 1024, 3])

In [ ]: ## Resizing images
        train_ds = tf.image.resize(train_ds, (150, 150))
        test_ds = tf.image.resize(test_ds, (150, 150))
```

b. Freeze parameters (weights) in model's lower convolutional layers

```
In [ ]: ## will not train base mode
        # Freeze Parameters in model's lower convolutional layers
        base_model.trainable = False

In [ ]: ## Preprocessing input
        train_ds = preprocess_input(train_ds)
        test_ds = preprocess_input(test_ds)

In [ ]: ## model details
        base_model.summary()

        Model: "vgg16"
        _________________________________________________________________
         Layer (type)                Output Shape              Param #
        =================================================================
         input_1 (InputLayer)        [(None, 150, 150, 3)]     0

         block1_conv1 (Conv2D)       (None, 150, 150, 64)      1792

         block1_conv2 (Conv2D)       (None, 150, 150, 64)      36928

         block1_pool (MaxPooling2D)  (None, 75, 75, 64)        0

         block2_conv1 (Conv2D)       (None, 75, 75, 128)       73856

         block2_conv2 (Conv2D)       (None, 75, 75, 128)       147584

         block2_pool (MaxPooling2D)  (None, 37, 37, 128)       0

         block3_conv1 (Conv2D)       (None, 37, 37, 256)       295168

         block3_conv2 (Conv2D)       (None, 37, 37, 256)       590080

         block3_conv3 (Conv2D)       (None, 37, 37, 256)       590080

         block3_pool (MaxPooling2D)  (None, 18, 18, 256)       0
```

```
 block5_pool (MaxPooling2D)  (None, 4, 4, 512)        0

=================================================================
Total params: 14,714,688
Trainable params: 0
Non-trainable params: 14,714,688
_________________________________________________________________
```

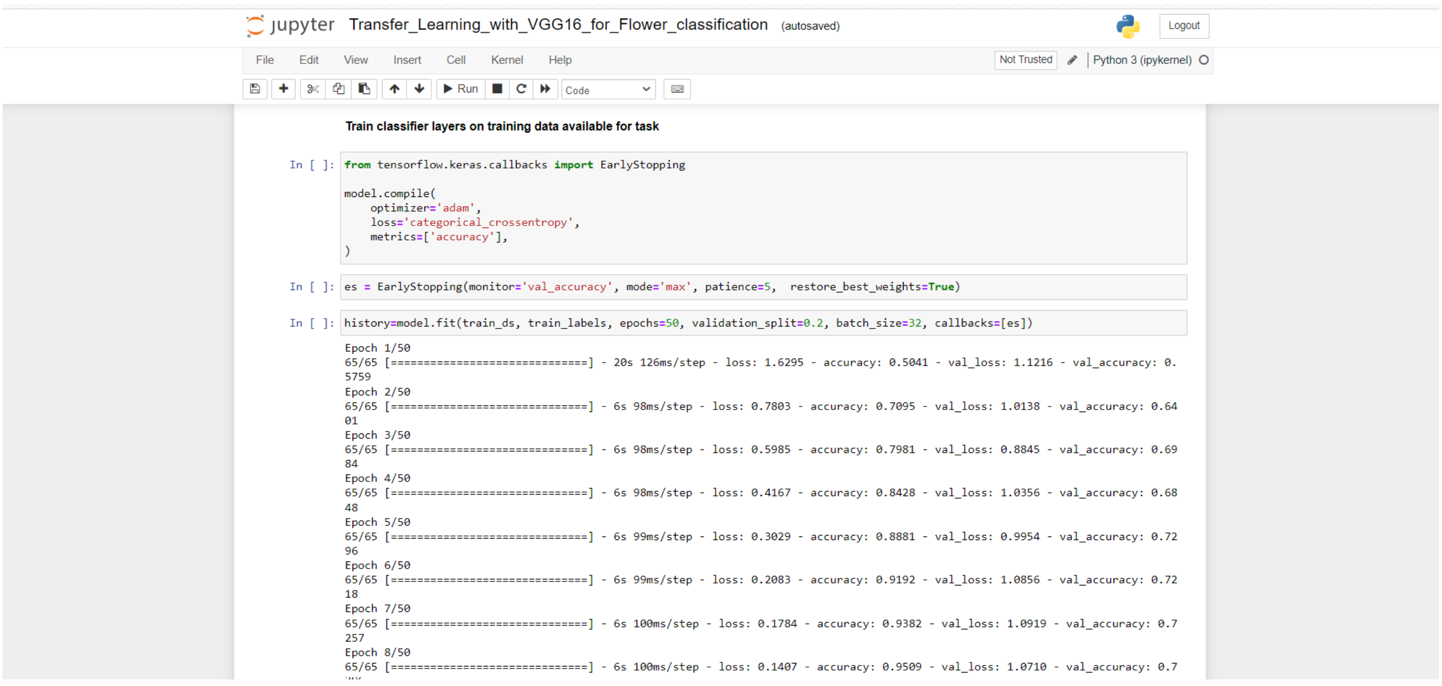


c. Add custom classifier with several layers of trainable parameters to model

Add custom classifier with two dense layers of trainable parameters to model

```
In [ ]: #add our layers on top of this model
        from tensorflow.keras import layers, models

        flatten_layer = layers.Flatten()
        dense_layer_1 = layers.Dense(50, activation='relu')
        dense_layer_2 = layers.Dense(20, activation='relu')
        prediction_layer = layers.Dense(5, activation='softmax')


        model = models.Sequential([
            base_model,
            flatten_layer,
            dense_layer_1,
            dense_layer_2,
            prediction_layer
        ])
```

d. Train classifier layers on training data available for task



Train classifier layers on training data available for task

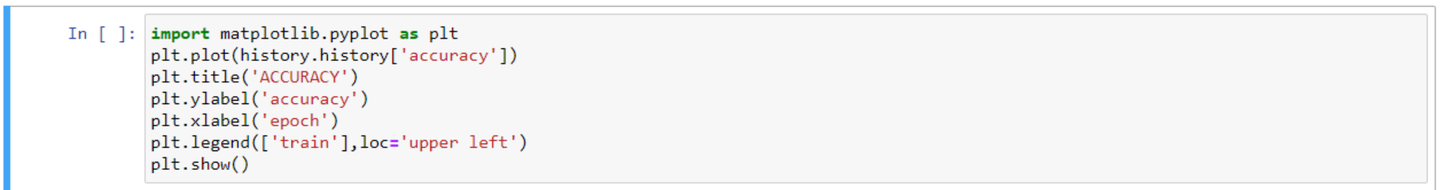
```
In [ ]: from tensorflow.keras.callbacks import EarlyStopping

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

```
In [ ]: es = EarlyStopping(monitor='val_accuracy', mode='max', patience=5,  restore_best_weights=True)
```

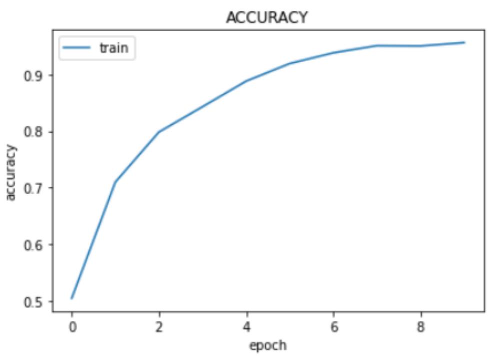
```
In [ ]: history=model.fit(train_ds, train_labels, epochs=50, validation_split=0.2, batch_size=32, callbacks=[es])
```

```
Epoch 1/50
65/65 [==============================] - 20s 126ms/step - loss: 1.6295 - accuracy: 0.5041 - val_loss: 1.1216 - val_accuracy: 0.
5759
Epoch 2/50
65/65 [==============================] - 6s 98ms/step - loss: 0.7803 - accuracy: 0.7095 - val_loss: 1.0138 - val_accuracy: 0.64
01
Epoch 3/50
65/65 [==============================] - 6s 98ms/step - loss: 0.5985 - accuracy: 0.7981 - val_loss: 0.8845 - val_accuracy: 0.69
84
Epoch 4/50
65/65 [==============================] - 6s 98ms/step - loss: 0.4167 - accuracy: 0.8428 - val_loss: 1.0356 - val_accuracy: 0.68
48
Epoch 5/50
65/65 [==============================] - 6s 99ms/step - loss: 0.3029 - accuracy: 0.8881 - val_loss: 0.9954 - val_accuracy: 0.72
96
Epoch 6/50
65/65 [==============================] - 6s 99ms/step - loss: 0.2083 - accuracy: 0.9192 - val_loss: 1.0856 - val_accuracy: 0.72
18
Epoch 7/50
65/65 [==============================] - 6s 100ms/step - loss: 0.1784 - accuracy: 0.9382 - val_loss: 1.0919 - val_accuracy: 0.7
257
Epoch 8/50
65/65 [==============================] - 6s 100ms/step - loss: 0.1407 - accuracy: 0.9509 - val_loss: 1.0710 - val_accuracy: 0.7
```

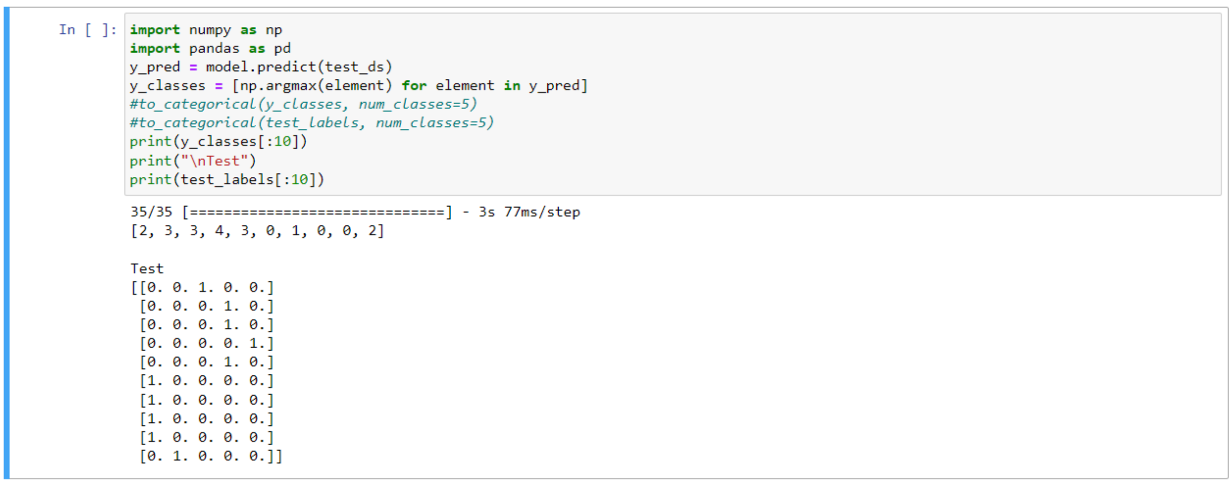
e. Result



```
In [ ]: import matplotlib.pyplot as plt
        plt.plot(history.history['accuracy'])
        plt.title('ACCURACY')
        plt.ylabel('accuracy')
        plt.xlabel('epoch')
        plt.legend(['train'],loc='upper left')
        plt.show()
```



epoch



```
In [ ]: import numpy as np
        import pandas as pd
        y_pred = model.predict(test_ds)
        y_classes = [np.argmax(element) for element in y_pred]
        #to_categorical(y_classes, num_classes=5)
        #to_categorical(test_labels, num_classes=5)
        print(y_classes[:10])
        print("\nTest")
        print(test_labels[:10])
```

```
35/35 [==============================] - 3s 77ms/step
[2, 3, 3, 4, 3, 0, 1, 0, 0, 2]

Test
[[0. 0. 1. 0. 0.]
 [0. 0. 0. 1. 0.]
 [0. 0. 0. 1. 0.]
 [0. 0. 0. 0. 1.]
 [0. 0. 0. 1. 0.]
 [1. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0.]]
```

Conclusion

In computer vision, transfer learning is usually expressed through the use of pre-trained models. A pre-trained model is a model that was trained on a large benchmark dataset to solve a problem similar to the one that we want to solve. Accordingly, due to the computational cost of training such models, it is common practice to import and use models from published literature (e.g. VGG, Inception, MobileNet). A comprehensive review of pre-trained models' performance on computer vision problems using data from the ImageNet (Deng et al. 2009) challenge is presented by Canziani et al. (2016).

FAQ:

1)What is Transfer learning?

2) What are pretrained Neural Network models?

3) Explain Pytorch library in short.

4) What are advantages of Transfer learning.

5) What are applications of Transfer learning.

6) Explain Caltech 101 images dataset.

7)Explain Imagenet dataset.

8) list down basic steps for transfer learning.

9) What is Data augmentation?

10)How and why Data augmentation is done related to transfer learning?

11)Why preprocessing is needed on inputdata in Transfer learning.

12)What is PyTorch Transforms module.Explain following commands w.r.t it :

Compose([

RandomResizedCrop(size=256, scale=(0.8, 1.0)),

RandomRotation(degrees=15),

ColorJitter(),

RandomHorizontalFlip(),

CenterCrop(size=224), # Image net standards

.ToTensor(),

Normalize

13) Explain the Validation Transforms steps with Pytorch Transforms .

14) Explain VGG-16 model from Pytorch