PRACTICAL NO-1

AIM: Introduction to Tensor flow /Keras - Importing Libraries and Modules.

Tensor flow

- **TensorFlow** is an open-source machine learning framework developed by **Google**.
- It is used to build and train **deep learning models** such as neural networks.
- It provides tools for:
  - Data preprocessing
  - Model building
  - Model training and evaluation
  - Deployment of AI models

Example uses: Image recognition, natural language processing, predictive analytics.

Keras

- **Keras** is a **high-level API** (Application Programming Interface) that runs on top of TensorFlow.
- It makes building and training deep learning models **simple and user-friendly**.
- You can define layers, activation functions, optimizers, and loss functions easily.

Think of **Keras** as the **"frontend"** for building models, and **TensorFlow** as the **"backend"** that runs them efficiently.

```
import numpy as np
import tensorflow as tf
from tensorflow import keras
# Everything is Tensor for Numpy
x = np.array(14)
x.ndim # A scalar is a 0-D tensor
```

```
0
```

```
x.shape # shape tuple is empty for a 0-D tensor
```

```
()
```

```
# A Vector is a 1-D Tensor
x = np.array([14, 21, 34, 78])
x.ndim
```

```
1
```

```
x.shape # Shape of a 1-D tensor is (no_of_elements,)
```

```
(4,)
```

```
# A Matrix or An array of array is a 2-D Tensor
x = np.array([[14,21,34,78],
[15,22,35,79],
[16,23,36,80]])
x.ndim # number of dimensions or rank of a 2-D Tensor
```

```
2
```

```
#Tensor Slicing
from keras.datasets import mnist
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
train_images.shape
```

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

```
my_slice = train_images[10]
my_slice.shape
```

```
(28, 28)
```

```
my_slice = train_images[10:100] # Selecting all images from 10th upto 100th␣
↪
(excl 100th)
my_slice.shape # we have selected 90 images
```

```
(90, 28, 28)
```

```
my_slice = train_images[:,14:28, 14:28]
my_slice.shape
```

```
(60000, 14, 14)
```

PRACTICAL NO-2

AIM: Loading the dataset, splitting dataset into training and testing data sets.

```
# Import required libraries
import pandas as pd
from sklearn.model_selection import train_test_split

# Load the dataset
data = pd.read_csv('/content/Salaries.csv')  # path in Colab
print("Dataset loaded successfully!\n")
print(data.head())
```

```
Dataset loaded successfully!

   Id       EmployeeName                                      JobTitle  \
0   1     NATHANIEL FORD  GENERAL MANAGER-METROPOLITAN TRANSIT AUTHORITY
1   2       GARY JIMENEZ                 CAPTAIN III (POLICE DEPARTMENT)
2   3     ALBERT PARDINI                 CAPTAIN III (POLICE DEPARTMENT)
3   4  CHRISTOPHER CHONG            WIRE ROPE CABLE MAINTENANCE MECHANIC
4   5    PATRICK GARDNER    DEPUTY CHIEF OF DEPARTMENT,(FIRE DEPARTMENT)

     BasePay  OvertimePay   OtherPay  Benefits   TotalPay  TotalPayBenefits  \
0  167411.18         0.00  400184.25       NaN  567595.43         567595.43
1  155966.02    245131.88  137811.38       NaN  538909.28         538909.28
2  212739.13    106088.18   16452.60       NaN  335279.91         335279.91
3   77916.00     56120.71  198306.90       NaN  332343.61         332343.61
4  134401.60      9737.00  182234.59       NaN  326373.19         326373.19

     Year  Notes         Agency  Status
0  2011.0    NaN  San Francisco     NaN
1  2011.0    NaN  San Francisco     NaN
2  2011.0    NaN  San Francisco     NaN
3  2011.0    NaN  San Francisco     NaN
4  2011.0    NaN  San Francisco     NaN
```

```
X = data.iloc[:, :-1]  # all columns except last
y = data.iloc[:, -1]   # last column as target
# Split dataset into training (80%) and testing (20%)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
print("\nTraining set size:", X_train.shape)
print("Testing set size:", X_test.shape)
```

```
Training set size: (7922, 12)
Testing set size: (1981, 12)
```

PRACTICAL NO-3

AIM: Implementation of Data preprocessing techniques.

```
import numpy as np # used for handling numbers
import pandas as pd # to handle datasets
from sklearn.impute import SimpleImputer # used for handling missing data
from sklearn.preprocessing import LabelEncoder, OneHotEncoder # used for
encoding categorical data
from sklearn.model_selection import train_test_split # used for splitting training and
testing data
from sklearn.preprocessing import StandardScaler

from google.colab import files
uploaded = files.upload()

import io
import pandas as pd
df = pd.read_csv(io.BytesIO(uploaded['DataPreprocessing  (2).csv']))
# Dimensions of the Dataframe
df.shape
```

```
(10, 4)
```

```
# Display the Dataframe
df
```

	Region	Age	Income	Online Shopper
0	India	49.0	86400.0	No
1	Brazil	32.0	57600.0	Yes
2	USA	35.0	64800.0	No
3	Brazil	43.0	73200.0	No
4	USA	45.0	NaN	Yes
5	India	40.0	69600.0	Yes
6	Brazil	NaN	62400.0	No
7	India	53.0	94800.0	Yes
8	USA	55.0	99600.0	No
9	India	42.0	80400.0	Yes

```
# Having a look at first 5 rows
```

```
df.head(5) # return only first 5 rows of the dataframe
```

	Region	Age	Income	Online Shopper
0	India	49.0	86400.0	No
1	Brazil	32.0	57600.0	Yes
2	USA	35.0	64800.0	No
3	Brazil	43.0	73200.0	No
4	USA	45.0	NaN	Yes

```
# Having a look at last 3 rows
df.tail(3) # return only last 3 rows of the dataframe
```

	Region	Age	Income	Online Shopper
7	India	53.0	94800.0	Yes
8	USA	55.0	99600.0	No
9	India	42.0	80400.0	Yes

```
df.columns # returns the names of all columns
```

```
Index(['Region', 'Age', 'Income', 'Online Shopper'], dtype='object')
```

```
df.info() # to check the datatypes of columns and the no of (non-null) entries in each column
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10 entries, 0 to 9
Data columns (total 4 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Region          10 non-null     object
 1   Age             9 non-null      float64
 2   Income          9 non-null      float64
 3   Online Shopper  10 non-null     object
dtypes: float64(2), object(2)
memory usage: 452.0+ bytes
```

```
# All non-numeric columns are by default omitted before summarising descriptive statistics for numerical columns
```

```
df.describe()
```

	Age	Income
count	9.000000	9.000000
mean	43.777778	76533.333333
std	7.693793	14718.695594
min	32.000000	57600.000000
25%	40.000000	64800.000000
50%	43.000000	73200.000000
75%	49.000000	86400.000000
max	55.000000	99600.000000

```
# Summarizing categorical variables
# Step 1 : to get only categorical columns extracted out
categorical = df.dtypes[df.dtypes == "object"].index
# Step 2: describe by indexing dataframe for only categorical values and applying describe() for those
df[categorical].describe()
```

	Region	Online Shopper
count	10	10
unique	3	2
top	India	No
freq	4	5

```
df["Region"].unique() #Display unique values of a particular column
```

```
array(['India', 'Brazil', 'USA'], dtype=object)
```

```
# Data Pre-processing
# Drop the duplicates from the dataset
df.drop_duplicates(inplace=True)
# inplace=True : dataframe object is to be modified without creating a copy
# inplace=False : otherwise
df
```

	Region	Age	Income	Online Shopper
0	India	49.0	86400.0	No
1	Brazil	32.0	57600.0	Yes
2	USA	35.0	64800.0	No
3	Brazil	43.0	73200.0	No
4	USA	45.0	NaN	Yes
5	India	40.0	69600.0	Yes
6	Brazil	NaN	62400.0	No
7	India	53.0	94800.0	Yes
8	USA	55.0	99600.0	No
9	India	42.0	80400.0	Yes

```
missing = np.where(df["Age"].isnull() == True)
missing
```

```
(array([6]),)
```

```
# Fill missing values with the mean of the respective columns
numeric_cols = df.select_dtypes(include=np.number).columns
df[numeric_cols] = df[numeric_cols].fillna(df[numeric_cols].mean())
df
```

	Region	Age	Income	Online Shopper
0	India	49.000000	86400.000000	No
1	Brazil	32.000000	57600.000000	Yes
2	USA	35.000000	64800.000000	No
3	Brazil	43.000000	73200.000000	No
4	USA	45.000000	76533.333333	Yes
5	India	40.000000	69600.000000	Yes
6	Brazil	43.777778	62400.000000	No
7	India	53.000000	94800.000000	Yes
8	USA	55.000000	99600.000000	No
9	India	42.000000	80400.000000	Yes

```
# 1. Rescaling (Min-Max Normalization)
## -- simplest type of normalization where the features are rescaled to the range
# MinMax Normalization using MinMaxScaler in sklearn library
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
TotalPayReshaped = df_sal.TotalPay.values.reshape(-1,1)
df_sal.TotalPay = scaler.fit_transform(TotalPayReshaped)
# minimum is 0 and maximum is 1 for TotalPay column
```

```
df_sal.describe()
```

	Id	TotalPay	TotalPayBenefits	Year	Notes
count	148654.000000	148654.000000	148654.000000	148654.000000	0.0
mean	74327.500000	0.132673	93692.554811	2012.522643	NaN
std	42912.857795	0.088905	62793.533483	1.117538	NaN
min	1.000000	0.000000	-618.130000	2011.000000	NaN
25%	37164.250000	0.064742	44065.650000	2012.000000	NaN
50%	74327.500000	0.126792	92404.090000	2013.000000	NaN
75%	111490.750000	0.187354	132876.450000	2014.000000	NaN
max	148654.000000	1.000000	567595.430000	2014.000000	NaN

```
# 2.Standardization (Z-score normalization)
## -- Here the feature is scaled such that the resulting mean is zero and varianc
scaler = StandardScaler()
TotalPayReshaped = df_sal.TotalPay.values.reshape(-1,1)
df_sal.TotalPay = scaler.fit_transform(TotalPayReshaped)
# Here the resulting mean is close to 0 and standard deviation is 1
df_sal.describe()
```

	Id	TotalPay	TotalPayBenefits	Year	Notes
count	148654.000000	1.486540e+05	148654.000000	148654.000000	0.0
mean	74327.500000	-1.498959e-16	93692.554811	2012.522643	NaN
std	42912.857795	1.000003e+00	62793.533483	1.117538	NaN
min	1.000000	-1.492304e+00	-618.130000	2011.000000	NaN
25%	37164.250000	-7.640884e-01	44065.650000	2012.000000	NaN
50%	74327.500000	-6.615046e-02	92404.090000	2013.000000	NaN
75%	111490.750000	6.150586e-01	132876.450000	2014.000000	NaN
max	148654.000000	9.755700e+00	567595.430000	2014.000000	NaN

```
# convert text to lower-case
input_text = "The 5 Biggest countries population wise are China, India, Unites St
input_text = input_text.lower()
print(input_text)
```

```
# Remove Punctuation marks and special symbols
# [!”#$%&’()*+,-./:;<=>?@[\]^_`{|}~]:
punctuation = '''!()-[]{};:'"\,<>./?@#$%^&*_~'''
user_input = input("Enter a string: ")
no_punc_input = ""
for char in user_input:
if char not in punctuation:
```

```
no_punc_input = no_punc_input + char
print("Punctuation free user input string : ", no_punc_input)
```

```
Enter a string: Hello, this is Dattaram Santosh Kolte
Punctuation free user input string :  Hello this is Dattaram Santosh Kolte
```

```
# Remove whitespaces
# Particularly stripping a text of its leading and trailing white spaces
input_text = " \t a string example \t"
input_text = input_text.strip()
input_text
```

```
# Remove stopwords
!pip install -q wordcloud
import wordcloud
import nltk
nltk.download("stopwords")
nltk.download("wordnet")
nltk.download("punkt")
nltk.download("averaged_perceptron_tagger")
nltk.download('omw-1.4')
nltk.download('punkt_tab')

import matplotlib.pyplot as plt
import io
import unicodedata
import string
from nltk.corpus import stopwords
from nltk import word_tokenize

example_sentence = "This is a sample sentence, showing off the stopwords filtration,
You,re going to be amused. You all will be happy that i sent this"
stop_words = set(stopwords.words('english'))
word_tokens = word_tokenize(example_sentence)
filtered_sentence = []
for w in word_tokens:
  if w not in stop_words:
    filtered_sentence.append(w)

print(word_tokens)
print(filtered_sentence)
```

['This', 'is', 'a', 'sample', 'sentence', ',', 'showing', 'off', 'the', 'stopwords', 'filtration', ',', 'You', ',', 're', 'going', 'to', 'be', 'amused', '.', 'You', 'all', 'will', 'be', 'happy', 'that', 'i', 'sent', 'this']

['This', 'sample', 'sentence', ',', 'showing', 'stopwords', 'filtration', ',', 'You', ',', 'going', 'amused', '.', 'You', 'happy', 'sent']

```
# Stemming using NLTK
# Stemming: process of reducing words to their stem or base form
# Examples: books - book, looked - look, pens - pen
stemmer = nltk.stem.PorterStemmer()
input_text = 'There are several types of stemming algorithms'
input_text = word_tokenize(input_text)
for word in input_text:
  print(stemmer.stem(word))
```

```
there
are
sever
type
of
stem
algorithm
```

```
# Lemmatization
# to reduce inflectional forms to a common base form
# As opposed to stemming, lemmatization does not simply chop off the inflections.
# Instead it used lexical knowledge bases to get the correct base forms of words.

lemmatizer = nltk.stem.WordNetLemmatizer()
input_text = "been had languages cities mice"
input_text = word_tokenize(input_text)

for word in input_text:
  print(lemmatizer.lemmatize(word))
```

```
been
had
language
city
mouse
```

PRACTICAL NO-4

AIM: Implementation of Artificial Neural Networks – McCulloch-Pitts neuron with ANDNOT function.

```
import numpy as np
def mcculloch_pitts_andnot(x1, x2):
    """
    Implements a McCulloch-Pitts neuron for the AND-NOT function.
    Args:
        x1: Input 1 (0 or 1)
        x2: Input 2 (0 or 1)
    Returns:
        Output of the neuron (0 or 1)
    """
    w1 = 1
    w2 = -1
    theta = 0.5
    weighted_sum = x1 * w1 + x2 * w2
    return 1 if weighted_sum >= theta else 0
# Test and print results in specified format
print("ANDNOT(0,0) =", mcculloch_pitts_andnot(0, 0))
print("ANDNOT(0,1) =", mcculloch_pitts_andnot(0, 1))
print("ANDNOT(1,0) =", mcculloch_pitts_andnot(1, 0))
print("ANDNOT(1,1) =", mcculloch_pitts_andnot(1, 1))
```

```
ANDNOT(0,0) = 0
ANDNOT(0,1) = 0
ANDNOT(1,0) = 1
ANDNOT(1,1) = 0
```

PRACTICAL NO-5

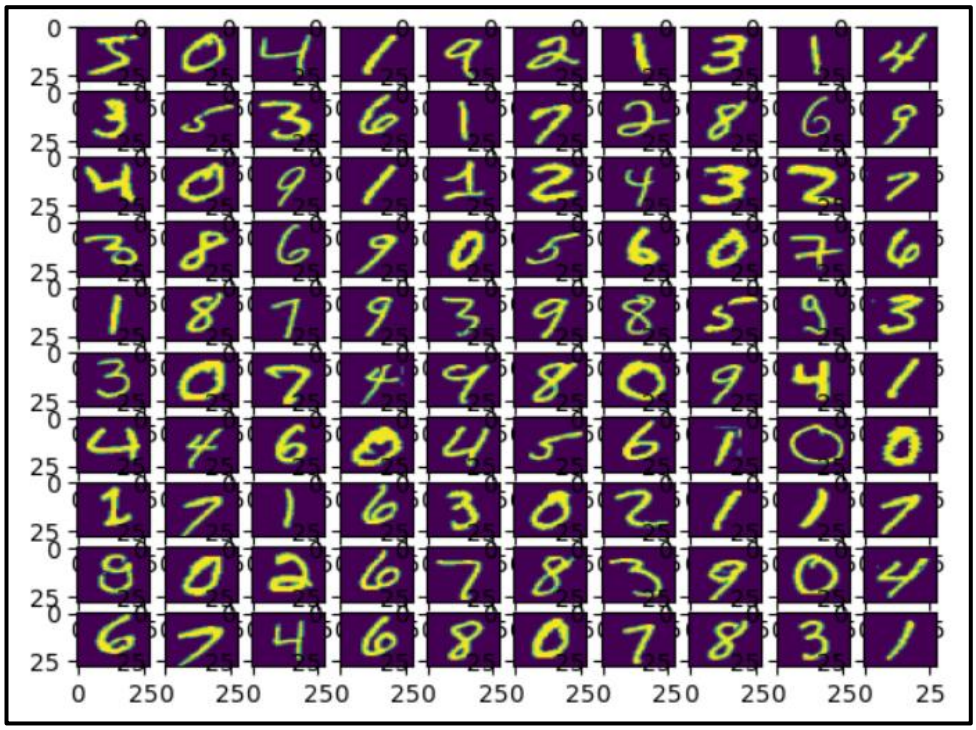
AIM: Multi-layer Perceptron Implementation in Tensorflow/Keras.

```
# importing modules
import tensorflow as tf
import numpy as np
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Activation
import matplotlib.pyplot as plt

# Step 2: Download the dataset.
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()

#Step 3: Now we will convert the pixels into floating-point values.
# Cast the records into float values
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')

# normalize image pixel values by dividing
# by 255
gray_scale = 255
x_train /= gray_scale
x_test /= gray_scale

# Step 4: Understand the structure of the dataset
print("Feature matrix:", x_train.shape)
print("Target matrix:", x_test.shape)
print("Feature matrix:", y_train.shape)
print("Target matrix:", y_test.shape)
```

```
Feature matrix: (60000, 28, 28)
Target matrix: (10000, 28, 28)
Feature matrix: (60000,)
Target matrix: (10000,)
```

```
# Step 5: Visualize the data.
fig, ax = plt.subplots(10, 10)
k = 0
for i in range(10):
   for j in range(10):
      ax[i][j].imshow(x_train[k].reshape(28, 28),
```

```
                aspect='auto')
    k += 1
plt.show()
```



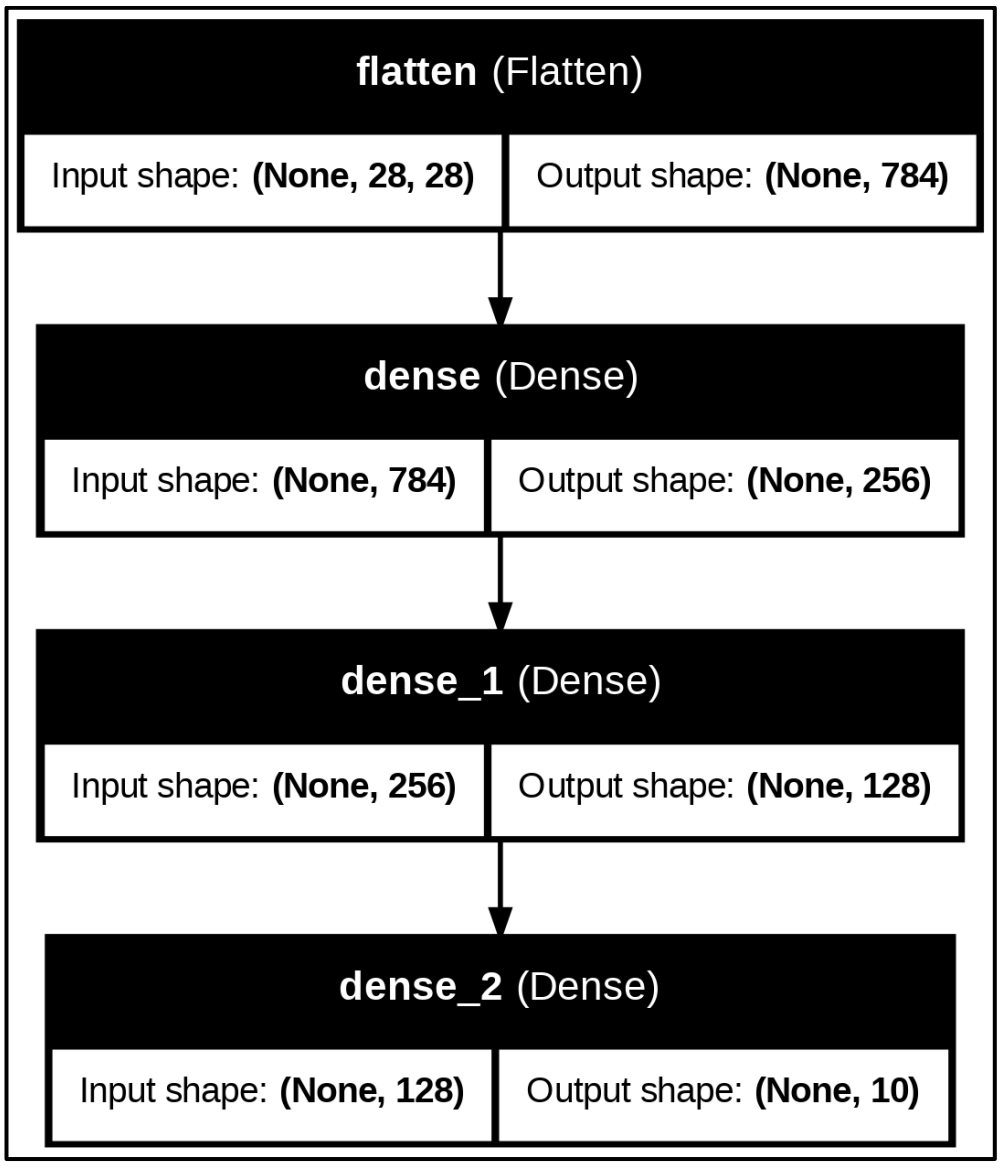
```
# Step 6: Form the Input, hidden, and output layers.
model = Sequential([
# reshape 28 row * 28 column data to 28*28 rows
Flatten(input_shape=(28, 28)),
# dense layer 1
Dense(256, activation='sigmoid'),
# dense layer 2
Dense(128, activation='sigmoid'),
# output layer
Dense(10, activation='sigmoid'),
])

# Step 7: Compile the model.
# Compile function is used here that involves the use of loss, optimizers, and metrics.
Here loss function
# used is sparse_categorical_crossentropy, optimizer used is adam.
model.compile(optimizer='adam',
loss='sparse_categorical_crossentropy',
metrics=['accuracy'])
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
flatten (Flatten)	(None, 784)	0
dense (Dense)	(None, 256)	200,960
dense_1 (Dense)	(None, 128)	32,896
dense_2 (Dense)	(None, 10)	1,290

Total params: 235,146 (918.54 KB)
Trainable params: 235,146 (918.54 KB)
Non-trainable params: 0 (0.00 B)

```
from tensorflow.keras.utils import plot_model
plot_model(model, to_file="model_plot.png", show_shapes=True,
show_layer_names=True)
```



```
# Step 8: Fit the model.
model.fit(x_train, y_train, epochs=10,
batch_size=2000,
validation_split=0.2)
# Step 9: Find Accuracy of the model.
results = model.evaluate(x_test, y_test, verbose = 0)
print('test loss, test acc:', results)
```

```
test loss, test acc: [0.2727438509464264, 0.9240999817848206]
```

Dattaram Santosh Kolte
SYMCA-B (97)

PRACTICAL NO-6

AIM: Back propagation Network for XOR function with Binary Input and Output.

```
import numpy as np
def sigmoid (x):
return 1/(1 + np.exp(-x))
def sigmoid_derivative(x):
return x * (1 - x)
#Input datasets
inputs = np.array([[0,0],[0,1],[1,0],[1,1]])
expected_output = np.array([[0],[1],[1],[0]])
epochs = 100
lr = 0.1
inputLayerNeurons, hiddenLayerNeurons, outputLayerNeurons = 2,2,1
#Random weights and bias initialization
hidden_weights =
np.random.uniform(size=(inputLayerNeurons,hiddenLayerNeurons))
hidden_bias =np.random.uniform(size=(1,hiddenLayerNeurons))
output_weights =
np.random.uniform(size=(hiddenLayerNeurons,outputLayerNeurons))
output_bias = np.random.uniform(size=(1,outputLayerNeurons))
print("Initial hidden weights: ",end='')
print(*hidden_weights)
print("Initial hidden biases: ",end='')
print(*hidden_bias)
print("Initial output weights: ",end='')
print(*output_weights)
print("Initial output biases: ",end='')
print(*output_bias)
```

```
Initial hidden weights: [0.25029523 0.96115627] [0.41591593 0.13533233]
Initial hidden biases: [0.59855215 0.83581803]
Initial output weights: [0.0754504] [0.64924904]
Initial output biases: [0.25890531]
```

```
#Training algorithm
for _ in range(epochs):
#Forward Propagation
hidden_layer_activation = np.dot(inputs,hidden_weights)
hidden_layer_activation += hidden_bias
hidden_layer_output = sigmoid(hidden_layer_activation)
output_layer_activation = np.dot(hidden_layer_output,output_weights)
output_layer_activation += output_bias
predicted_output = sigmoid(output_layer_activation)

#Backpropagation
error = expected_output - predicted_output
d_predicted_output = error * sigmoid_derivative(predicted_output)
error_hidden_layer = d_predicted_output.dot(output_weights.T)
d_hidden_layer = error_hidden_layer * sigmoid_derivative(hidden_layer_output)

#Updating Weights and Biases
output_weights += hidden_layer_output.T.dot(d_predicted_output) * lr
output_bias += np.sum(d_predicted_output,axis=0,keepdims=True) * lr
hidden_weights += inputs.T.dot(d_hidden_layer) * lr
hidden_bias += np.sum(d_hidden_layer,axis=0,keepdims=True) * lr
print("Final hidden weights: ",end='')
print(*hidden_weights)
print("Final hidden bias: ",end='')
print(*hidden_bias)
print("Final output weights: ",end='')
print(*output_weights)
print("Final output bias: ",end='')
print(*output_bias)
print("\nOutput from neural network after 10,000 epochs: ",end='')
print(*predicted_output)
#Hence, the neural network has converged to the expected output:
#[0] [1] [1] [0].
```

```
Final hidden weights: [0.25020304 0.96059346] [0.41582557 0.13515358]
Final hidden bias: [0.59830377 0.83410052]
Final output weights: [0.06374193] [0.63630773]
Final output bias: [0.24236418]

Output from neural network after 10,000 epochs: [0.68146247] [0.68680187] [0.7044664] [0.70782975]
```

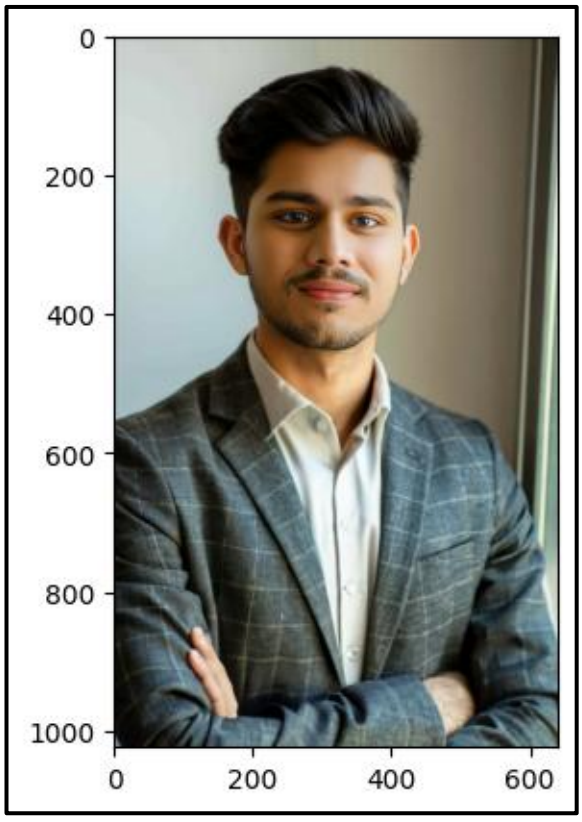
Dattaram Santosh Kolte
SYMCA-B (97)

PRACTICAL NO-7

AIM: Implementation of Regularization Techniques.

a) Dataset Augmentation

```
#The most commonly used operations are-
#1.Rotation#
#2.Shearing (transform the orientation of the image)
#3.Zooming
#4.Cropping
#5.Flipping
#6.Changing the brightness level###
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.utils import array_to_img, img_to_array, load_img
import matplotlib.pyplot as plt
import numpy as np
datagen = ImageDataGenerator(
rotation_range = 40,
shear_range = 0.2,
zoom_range = 0.2,
horizontal_flip = True,
brightness_range = (0.5, 1.5))
from google.colab import drive
drive.mount('/content/drive')
img_path='//content/drive/MyDrive/Brijesh.jpg' #path for image
bimg=plt.imread(img_path) # reading the image
plt.imshow(bimg)
```



```
# Converting the input sample image to an array
x = img_to_array(bimg)
# Reshaping the input image
# x = x.reshape((1, ) + x.shape) # Remove this lin
x = x.reshape((1,) + x.shape)
x
```

```
array([[[[198., 202., 201.],
         [198., 202., 201.],
         [198., 202., 201.],
         ...,
         [ 41.,  37.,  25.],
         [ 41.,  37.,  25.],
         [ 41.,  37.,  25.]],

        [[198., 202., 201.],
         [198., 202., 201.],
         [198., 202., 201.],
         ...,
         [ 41.,  37.,  25.],
         [ 41.,  37.,  25.],
         [ 41.,  37.,  25.]],

        [[198., 202., 201.],
         [198., 202., 201.],
         [198., 202., 201.],
         ...,
         [ 41.,  37.,  25.],
         [ 41.,  37.,  25.],
         [ 41.,  37.,  25.]],
```

```
     [[123., 157., 184.],
      [119., 153., 178.],
      [120., 152., 177.],
      ...,
      [126., 139., 129.],
      [106., 116., 105.],
      [ 88.,  96.,  83.]],

     [[124., 156., 181.],
      [120., 152., 177.],
      [123., 153., 179.],
      ...,
      [136., 149., 139.],
      [119., 129., 118.],
      [100., 111.,  97.]],

     [[122., 154., 179.],
      [119., 151., 176.],
      [122., 152., 176.],
      ...,
      [142., 155., 145.],
      [126., 138., 126.],
      [109., 120., 106.]]]], dtype=float32)
```

```
# Reshaping the input image
# x = x.reshape((1, ) + x.shape) # Remove or comment out this redundant reshape
# Generating and saving 5 augmented samplesusing the above defined parameters.
import os
if not os.path.exists('/content/drive/MyDrive/Results'):
   os.makedirs('/content/drive/MyDrive/Results')

i = 0
for batch in datagen.flow(x, batch_size = 1,
save_to_dir ='/content/drive/MyDrive/Results',
save_prefix ='imgage', save_format ='jpeg'):
   i += 1
   if i> 4:
      break
```

b) Early Stopping

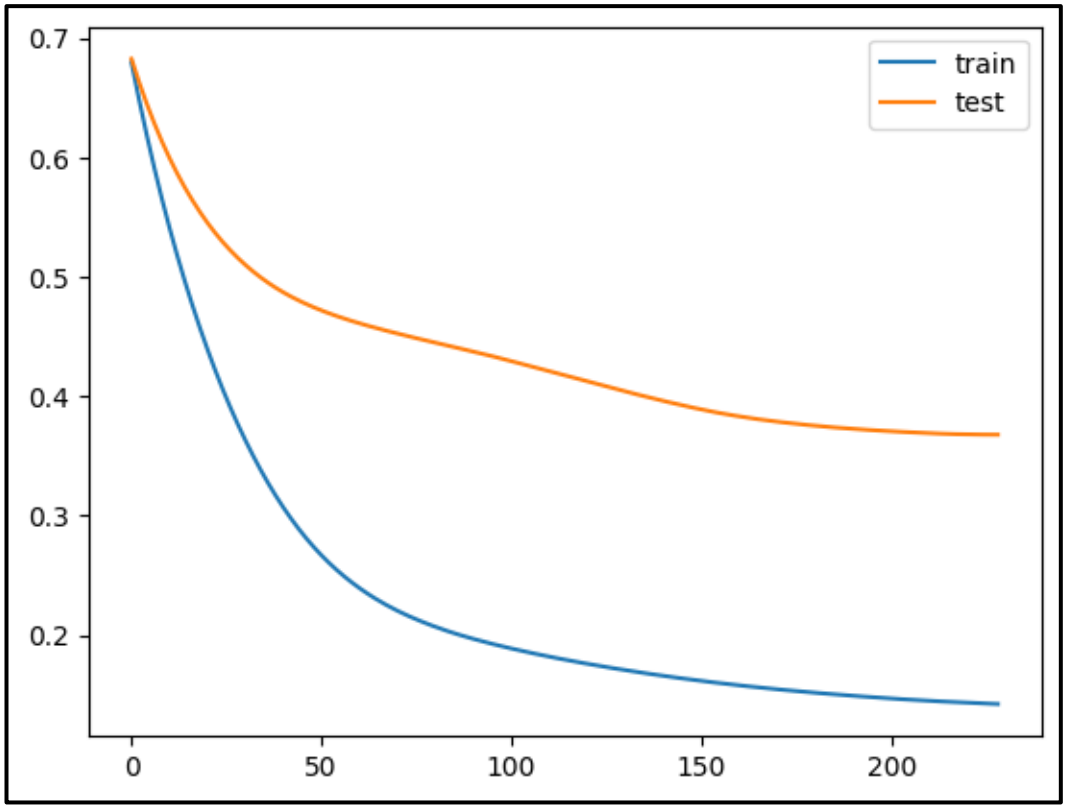
```
from sklearn.datasets import make_moons
from keras.models import Sequential
from keras.layers import Dense
from keras.callbacks import EarlyStopping
from matplotlib import pyplot
X, y = make_moons(n_samples=100, noise=0.2, random_state=1)
print(X)
```

```
[[ 1.36698238 -0.23541584]
 [ 1.76404402 -0.34563288]
 [-0.37868174  0.41004375]
 [ 1.15113747 -0.13597622]
 [ 2.31168314  0.32295125]
 [ 0.53866045  0.73704603]
 [-0.93583639  1.00686001]
 [ 1.32563024 -0.13540284]
 [ 0.75398022 -0.37261326]
 ...
 [ 0.65792377 -1.03466843]
 [ 1.12996342  0.70867167]
 [-1.04121655  0.88339745]]
```

```
# SPLIT INTO TRAIN AND TEST
n_train = 30
trainX, testX = X[:n_train, :], X[n_train:, :]
trainy, testy = y[:n_train], y[n_train:]
# DEFINE MODEL
model = Sequential()
model.add(Dense(500, input_dim=2, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
es = EarlyStopping(monitor='val_loss',verbose=1)
# FIT MODEL
history = model.fit(trainX, trainy, validation_data=(testX, testy),
epochs=300,verbose=1,callbacks=[es])
# EVALUATE THE MODEL
_, train_acc = model.evaluate(trainX, trainy, verbose=0)
_, test_acc = model.evaluate(testX, testy, verbose=0)
print('Train: %.3f, Test: %.3f' % (train_acc, test_acc))
```

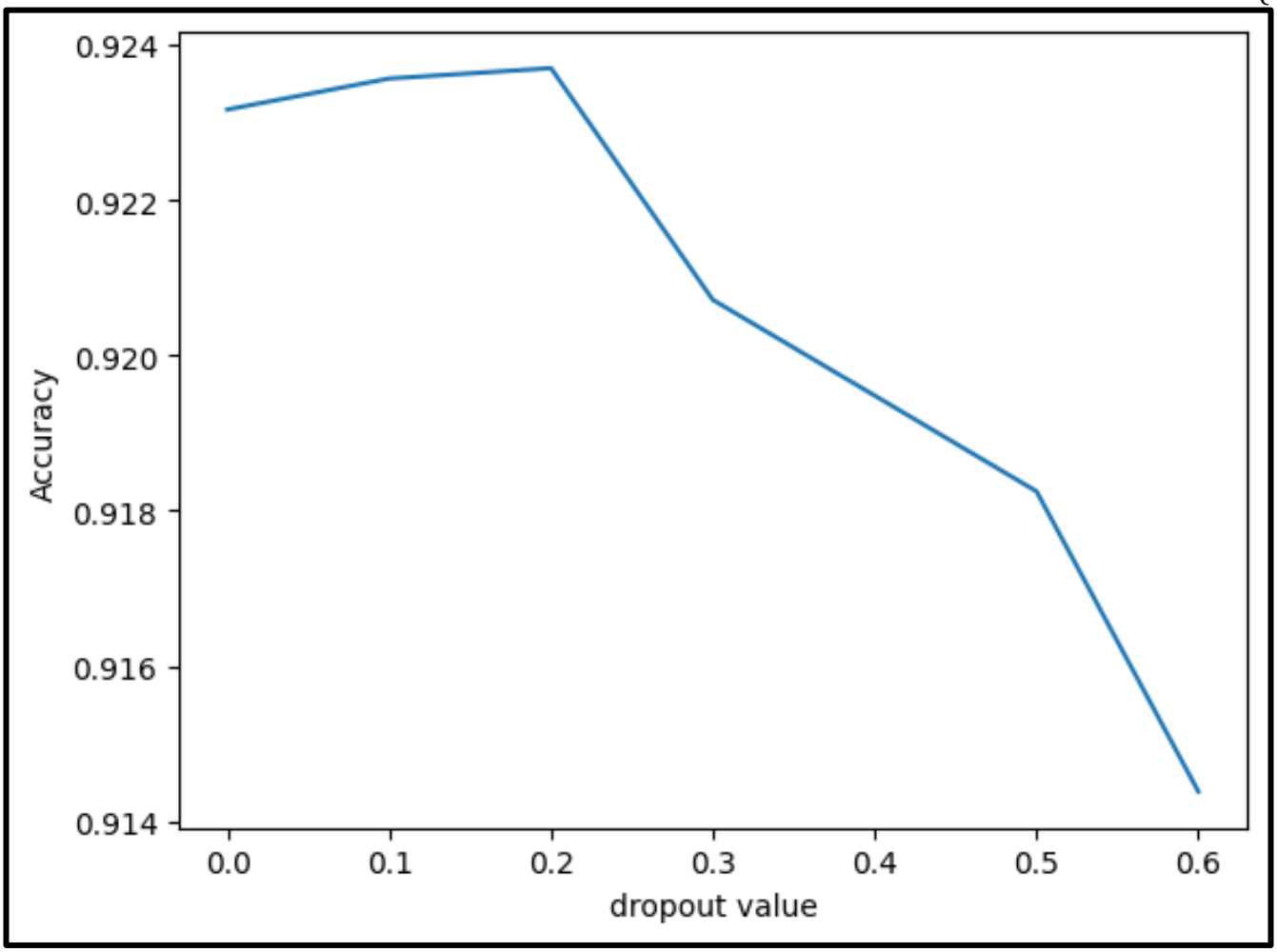
Epoch 230: early stopping
Train: 0.967, Test: 0.814

```
# PLOT TRAINING HISTORY
pyplot.plot(history.history[
'loss'
], label=
'train'
)
pyplot.plot(history.history[
'val_loss'
], label=
'test'
)
pyplot.legend()
pyplot.show()
```



c) Dropout

```
import tensorflow as tf
import numpy as np
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Activation
from tensorflow.keras.layers import Dropout
import matplotlib.pyplot as plt

(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()

x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
gray_scale = 255
x_train /= gray_scale
x_test /= gray_scale
dropouts = [0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6] #By increasing the dropout value, the MSE increases.
Thus, we can not use higher value
#The higher dropout value effects negatively to the accuracy level
accs=[]
for d in dropouts:
   model = Sequential()
   model.add(Flatten(input_shape=(28, 28)))
   model.add(Dense(256, activation='sigmoid'))
   model.add(Dense(128, activation='sigmoid'))
   model.add(Dropout(d))
   model.add(Dense(10, activation='sigmoid'))

model.compile(optimizer='adam',loss='sparse_categorical_crossentropy',metrics=['accuracy'])
   model.fit(x_train, y_train, epochs=10,batch_size=2000,validation_split=0.2,verbose=2)
   acc = model.evaluate(x_train, y_train)
   accs.append(acc[1])
# Append the accuracy value (second element of acc)
plt.plot(dropouts,accs)
plt.ylabel("Accuracy")
plt.xlabel("dropout value")
plt.show()
```


0.924
0.922
0.920
0.918
0.916
0.914
Accuracy
0.0
0.1
0.2
0.3
0.4
0.5
0.6
dropout value

PRACTICAL NO-8

AIM: Implementation and analysis of Deep Neural network algorithm: Convolutional neural network (CNN) – Object identification and classification.

```
import numpy as np
import matplotlib.pyplot as plt
from keras.datasets import fashion_mnist
from tensorflow.keras.utils import to_categorical
from sklearn.model_selection import train_test_split
import keras
from tensorflow import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
from tensorflow.keras.layers import BatchNormalization
from tensorflow.keras.layers import Dense

(train_X,train_Y), (test_X, test_Y) = fashion_mnist.load_data()
print('training data shape:', train_X.shape, train_Y.shape)
print('testing data shape:', test_X.shape, test_Y.shape)
```

```
training data shape: (60000, 28, 28) (60000,)
testing data shape: (10000, 28, 28) (10000,)
```

```
# find the unique labels (as numbers in this case)
labels_unique = np.unique(train_Y)
# find the total number of unique classes
num_classes = len(labels_unique)
print('Total number of classes : ', num_classes)
print('Unique labels : ', labels_unique)
```

```
Total number of classes :  10
Unique labels :  [0 1 2 3 4 5 6 7 8 9]
```

```
# reshape command parameters:
# reshape "all" of the rows by setting first parameter to -1
# the first column will have a value of 28
# the second column will have a value of 28 as well
# the third column will have a value of 1
train_X = train_X.reshape(-1, 28, 28, 1)
test_X = test_X.reshape(-1, 28, 28, 1)
train_X.shape, test_X.shape
```

```
((60000, 28, 28, 1), (10000, 28, 28, 1))
```

```
#Convert data type
#Currently this is an int8 data type, this won't work for our inputs because we will be
normalizing our pixel data which will requ
# in later steps.
train_X = train_X.astype('float32')
test_X = test_X.astype('float32')

train_X = train_X / 255
test_X = test_X / 255

# Change the labels from categorical to one-hot encoding
train_Y_one_hot = to_categorical(train_Y)
test_Y_one_hot = to_categorical(test_Y)
print('Shape:', train_X.shape)
print('Shape:', train_Y_one_hot.shape)
train_X,valid_X,train_label,valid_label = train_test_split(train_X, train_Y_one_hot,
test_size=0.2, random_state=13)
print('training set shape:', train_X.shape)
print('validation set shape:', valid_X.shape)
print('training label set shape:', train_label.shape)
print('validation label set shape:', valid_label.shape)
```

```
Shape: (60000, 28, 28, 1)
Shape: (60000, 10)
training set shape: (48000, 28, 28, 1)
validation set shape: (12000, 28, 28, 1)
training label set shape: (48000, 10)
validation label set shape: (12000, 10)
```

```
batch_size = 64
num_classes = 10
#Setting up sequential model layers
fashion_model = Sequential()
fashion_model.add(Conv2D(32, kernel_size=(3,
3),activation='relu',input_shape=(28,28,1),padding='same'))
fashion_model.add(MaxPooling2D((2, 2),padding='same'))
fashion_model.add(Conv2D(64, (3, 3), activation='linear',padding='same'))
fashion_model.add(Dropout(0.25))
fashion_model.add(MaxPooling2D(pool_size=(2, 2),padding='same'))
fashion_model.add(Dropout(0.25))
fashion_model.add(Conv2D(128, (3, 3), activation='linear',padding='same'))
fashion_model.add(MaxPooling2D(pool_size=(2, 2),padding='same'))
fashion_model.add(Dropout(0.4))
fashion_model.add(Flatten())
fashion_model.add(Dense(128, activation='linear'))
fashion_model.add(Dropout(0.3))
fashion_model.add(Dense(num_classes, activation='softmax'))
fashion_model.compile(loss=keras.losses.categorical_crossentropy,
optimizer=keras.optimizers.Adam(),metrics=['accuracy'])
fashion_model.summary()
```

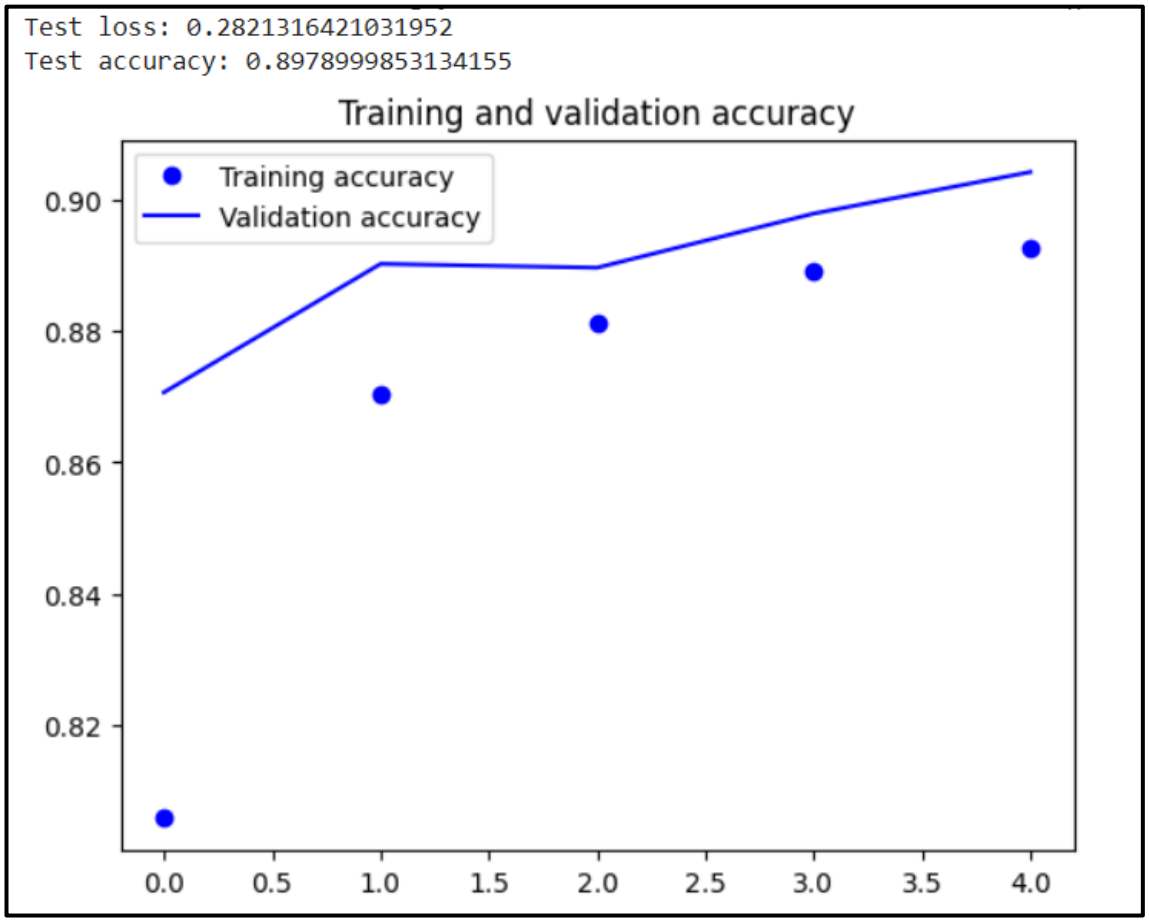
Model: "sequential"

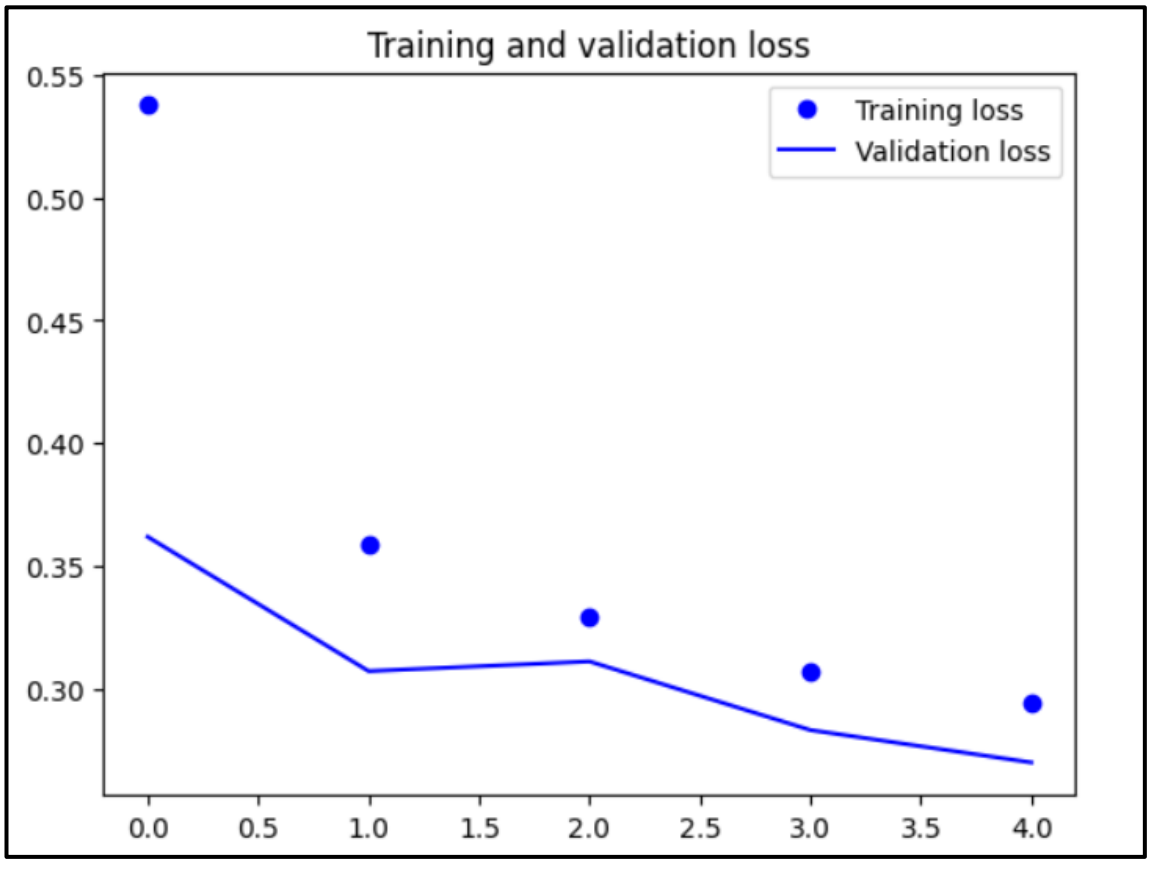
Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 28, 28, 32)	320
max_pooling2d (MaxPooling2D)	(None, 14, 14, 32)	0
conv2d_1 (Conv2D)	(None, 14, 14, 64)	18,496
dropout (Dropout)	(None, 14, 14, 64)	0
max_pooling2d_1 (MaxPooling2D)	(None, 7, 7, 64)	0
dropout_1 (Dropout)	(None, 7, 7, 64)	0
conv2d_2 (Conv2D)	(None, 7, 7, 128)	73,856
max_pooling2d_2 (MaxPooling2D)	(None, 4, 4, 128)	0
dropout_2 (Dropout)	(None, 4, 4, 128)	0
flatten (Flatten)	(None, 2048)	0
dense (Dense)	(None, 128)	262,272
dropout_3 (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 10)	1,290

Total params: 356,234 (1.36 MB)
Trainable params: 356,234 (1.36 MB)
Non-trainable params: 0 (0.00 B)

```
#Training the model
fashion_train = fashion_model.fit(train_X, train_label,
batch_size=batch_size,epochs=5,verbose=1,validation_data=(valid_X, valid_label))

#Evaluate the training data
test_eval = fashion_model.evaluate(test_X, test_Y_one_hot, verbose=0)
fashion_model.save("fashion_model.h5")
#from keras.models import load_model
#fashion_model = load_model('fashion_model.h5py')
print('Test loss:', test_eval[0])
print('Test accuracy:', test_eval[1])
accuracy = fashion_train.history['accuracy']
val_accuracy = fashion_train.history['val_accuracy']
loss = fashion_train.history['loss']
val_loss = fashion_train.history['val_loss']
epochs = range(len(accuracy))
plt.plot(epochs, accuracy, 'bo', label='Training accuracy')
plt.plot(epochs, val_accuracy, 'b', label='Validation accuracy')
plt.title('Training and validation accuracy')
plt.legend()
plt.figure()
plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()
plt.show()
```

```
Test loss: 0.2821316421031952
Test accuracy: 0.8978999853134155
```





PRACTICAL NO-9

AIM: Image recognition using CNN.

```
import numpy as np
import matplotlib.pyplot as plt
from keras.datasets import fashion_mnist
from tensorflow.keras.utils import to_categorical
from sklearn.model_selection import train_test_split
import keras
from tensorflow import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
from tensorflow.keras.layers import BatchNormalization
from tensorflow.keras.layers import Dense

#Load the fashion mnist dataset both training and testing data
(train_X,train_Y), (test_X, test_Y) = fashion_mnist.load_data()
print('training data shape:', train_X.shape, train_Y.shape)
print('testing data shape:', test_X.shape, test_Y.shape)
```

```
training data shape: (60000, 28, 28) (60000,)
testing data shape: (10000, 28, 28) (10000,)
```

```
# find the unique labels (as numbers in this case)
labels_unique = np.unique(train_Y)

# find the total number of unique classes
num_classes = len(labels_unique)

print('Total number of classes : ', num_classes)

print('Unique labels : ', labels_unique)
```

```
Total number of classes :  10
Unique labels :  [0 1 2 3 4 5 6 7 8 9]
```

```
# reshape command parameters:
# reshape "all" of the rows by setting first parameter to -1
# the first column will have a value of 28
# the second column will have a value of 28 as well
# the third column will have a value of 1
train_X = train_X.reshape(-1, 28, 28, 1)
test_X = test_X.reshape(-1, 28, 28, 1)
train_X.shape, test_X.shape
```

```
# Convert data type

train_X = train_X.astype('float32')
test_X = test_X.astype('float32')
```

```
((60000, 28, 28, 1), (10000, 28, 28, 1))
```

```
# Convert pixel values

train_X = train_X / 255
test_X = test_X / 255

# Change the labels from categorical to one-hot encoding
train_Y_one_hot = to_categorical(train_Y)
test_Y_one_hot = to_categorical(test_Y)

# Split training data

print('Shape:', train_X.shape)
print('Shape:', train_Y_one_hot.shape)

train_X,valid_X,train_label,valid_label = train_test_split(train_X, train_Y_one_hot, test_size=0.2,
random_state=13)

print('training set shape:', train_X.shape)
print('validation set shape:', valid_X.shape)
print('training label set shape:', train_label.shape)
print('validation label set shape:', valid_label.shape)
```

```
Shape: (60000, 28, 28, 1)
Shape: (60000, 10)
training set shape: (48000, 28, 28, 1)
validation set shape: (12000, 28, 28, 1)
training label set shape: (48000, 10)
validation label set shape: (12000, 10)
```

```
# Setting batch and epoch
batch_size = 64
num_classes = 10

#Setting up sequential model layers

fashion_model = Sequential()
fashion_model.add(Conv2D(32, kernel_size=(3,
3),activation='relu',input_shape=(28,28,1),padding='same'))

fashion_model.add(MaxPooling2D((2, 2),padding='same'))
```

```
fashion_model.add(Conv2D(64, (3, 3), activation='linear',padding='same'))
fashion_model.add(Dropout(0.25))
fashion_model.add(MaxPooling2D(pool_size=(2, 2),padding='same'))
fashion_model.add(Dropout(0.25))
fashion_model.add(Conv2D(128, (3, 3), activation='linear',padding='same'))
fashion_model.add(MaxPooling2D(pool_size=(2, 2),padding='same'))
fashion_model.add(Dropout(0.4))
fashion_model.add(Flatten())
fashion_model.add(Dense(128, activation='linear'))
fashion_model.add(Dropout(0.3))
fashion_model.add(Dense(num_classes, activation='softmax'))

fashion_model.compile(loss=keras.losses.categorical_crossentropy,
optimizer=keras.optimizers.Adam(),metrics=['accuracy'])

fashion_model.summary()
```

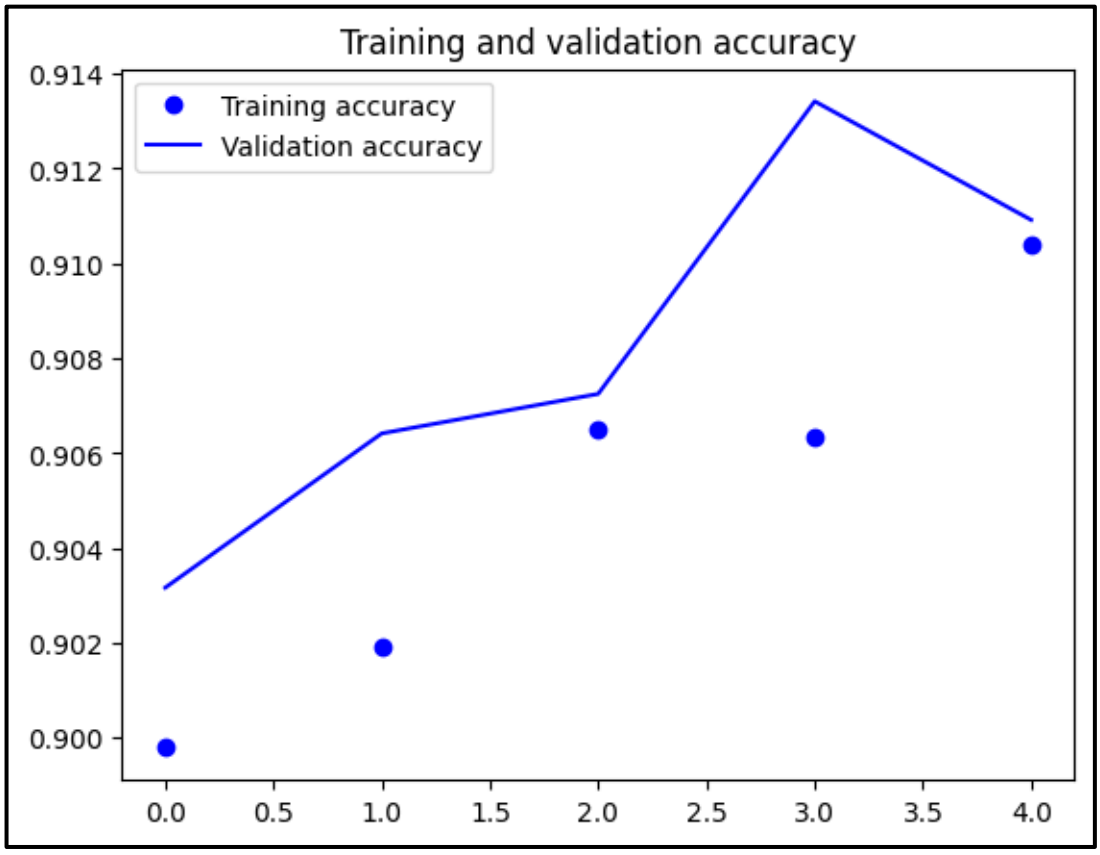
Model: "sequential"

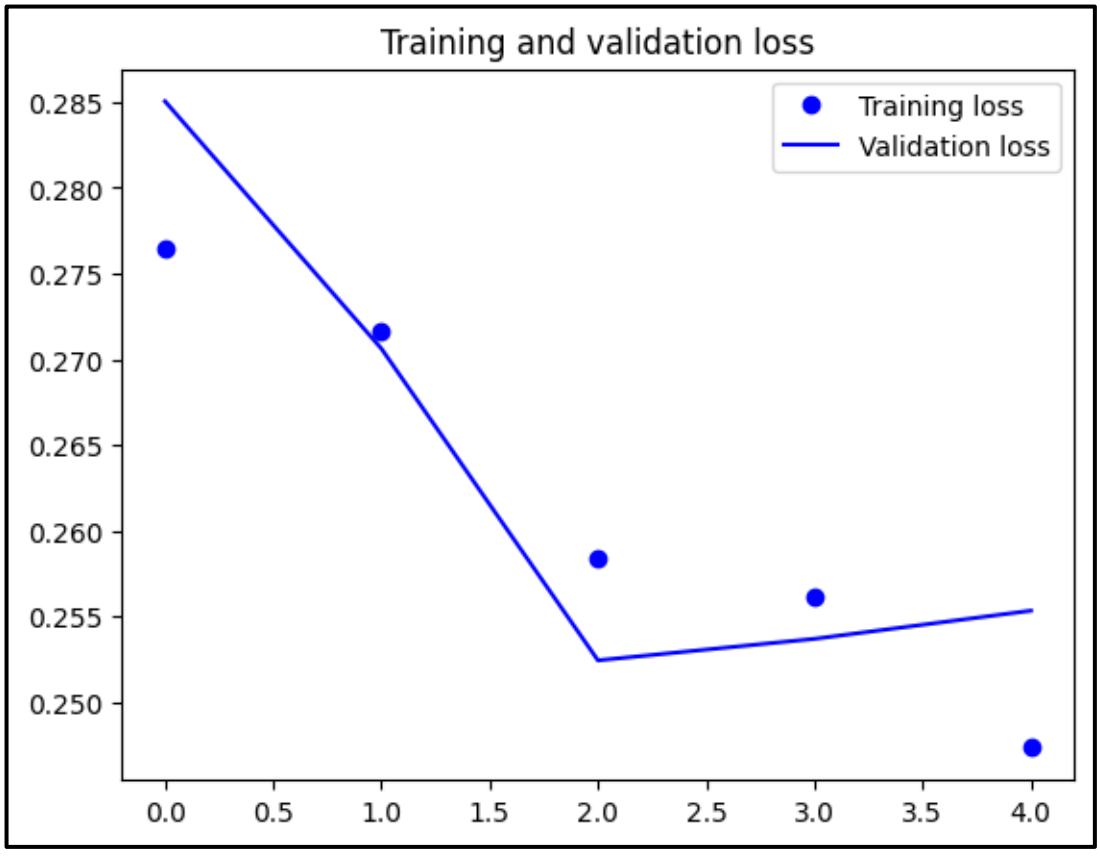
Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 28, 28, 32)	320
max_pooling2d (MaxPooling2D)	(None, 14, 14, 32)	0
conv2d_1 (Conv2D)	(None, 14, 14, 64)	18,496
dropout (Dropout)	(None, 14, 14, 64)	0
max_pooling2d_1 (MaxPooling2D)	(None, 7, 7, 64)	0
dropout_1 (Dropout)	(None, 7, 7, 64)	0
conv2d_2 (Conv2D)	(None, 7, 7, 128)	73,856
max_pooling2d_2 (MaxPooling2D)	(None, 4, 4, 128)	0
dropout_2 (Dropout)	(None, 4, 4, 128)	0
flatten (Flatten)	(None, 2048)	0
dense (Dense)	(None, 128)	262,272
dropout_3 (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 10)	1,290

Total params: 356,234 (1.36 MB)
Trainable params: 356,234 (1.36 MB)
Non-trainable params: 0 (0.00 B)

```
# Training the model

fashion_train = fashion_model.fit(train_X, train_label,
batch_size=batch_size,epochs=5,verbose=1,validation_data=(valid_X, valid_label))

# Evaluate the training data

test_eval = fashion_model.evaluate(test_X, test_Y_one_hot, verbose=0)

fashion_model.save("fashion_model.h5py")
#from keras.models import load_model
#fashion_model = load_model('fashion_model.h5py')

print('Test loss:', test_eval[0])
print('Test accuracy:', test_eval[1])

accuracy = fashion_train.history['accuracy']
val_accuracy = fashion_train.history['val_accuracy']
loss = fashion_train.history['loss']
val_loss = fashion_train.history['val_loss']
epochs = range(len(accuracy))

# Visulize
plt.plot(epochs, accuracy, 'bo', label='Training accuracy')
plt.plot(epochs, val_accuracy, 'b', label='Validation accuracy')
plt.title('Training and validation accuracy')
plt.legend()
plt.figure()
plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()
plt.show()
```

```
Test loss: 0.27140909433364887
Test accuracy: 0.9071000218391418
```


Training and validation accuracy
Training accuracy
Validation accuracy
0.914
0.912
0.910
0.908
0.906
0.904
0.902
0.900
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0


Training and validation loss
Training loss
Validation loss
0.285
0.280
0.275
0.270
0.265
0.260
0.255
0.250
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0

PRACTICAL NO-10

AIM: Implementation and analysis of Deep Neural network algorithm: Recurrent neural network (RNN) - Character recognition.

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import SimpleRNN, Dense
from tensorflow.keras.datasets import mnist
from tensorflow.keras.utils import plot_model
import matplotlib.pyplot as plt

# Load and preprocess the MNIST dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()
# Normalize the pixel values to be between 0 and 1
x_train, x_test = x_train / 255.0, x_test / 255.0
# Reshape the data for RNN input
x_train = x_train.reshape(x_train.shape[0], -1, 1)
x_test = x_test.reshape(x_test.shape[0], -1, 1)
x_train.shape
x_test.shape
# Reshape the data for RNN input
x_train = x_train.reshape(x_train.shape[0], -1, 1)
x_test = x_test.reshape(x_test.shape[0], -1, 1)
x_train.shape
x_test.shape
```

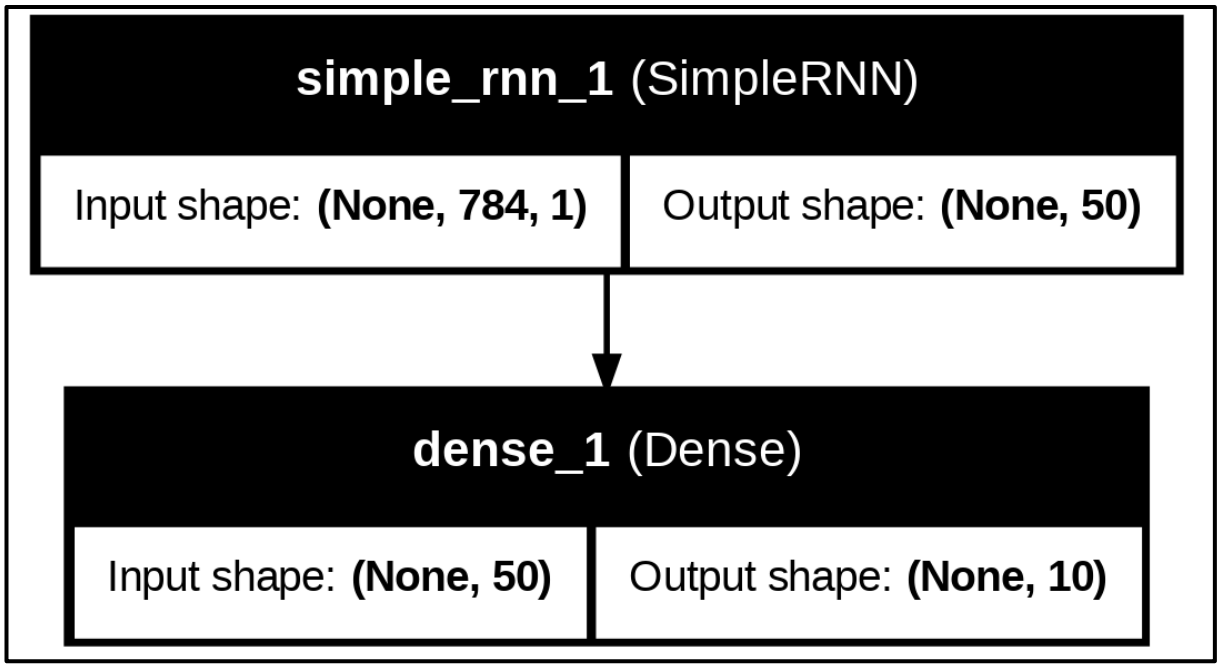
```
(10000, 784, 1)
```

```
# Build the RNN model
# SimpleRNN is the recurrent layer object in Keras
#simpleRNN() function is used to create a RNN layer consisting of a single SimpleRNNCell.
# Here 50 is the num_units.
# num_units is the dimensionality of the output space.
#The num_units in a RNN is the number of RNN memory units to each input of the sequence in
# vertical manner attached to each other, and each one is passing the filtered information to next memory units
model = Sequential([
SimpleRNN(50, input_shape=(x_train.shape[1], x_train.shape[2]), activation='relu'),
Dense(10, activation='softmax')
])
model.summary()
plot_model(model,show_shapes=True,show_layer_names=True,)
```

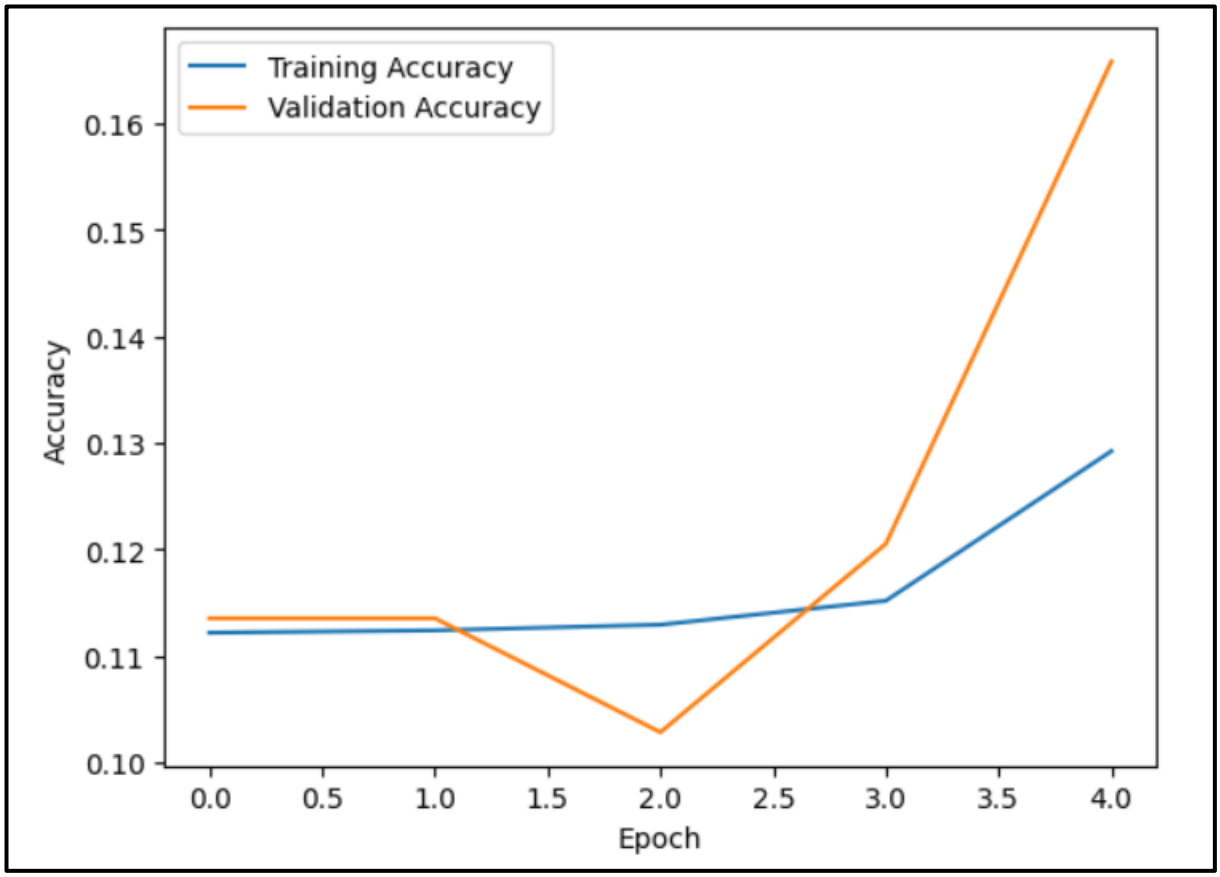
Model: "sequential_1"

Layer (type)	Output Shape	Param #
simple_rnn_1 (SimpleRNN)	(None, 50)	2,600
dense_1 (Dense)	(None, 10)	510

Total params: 3,110 (12.15 KB)
Trainable params: 3,110 (12.15 KB)
Non-trainable params: 0 (0.00 B)




```
# Compile the model
model.compile(optimizer='adam',loss='sparse_categorical_crossentropy',
metrics=['accuracy'])
# Train the model
history = model.fit(x_train, y_train, epochs=5, validation_data=(x_test, y_test))
# Evaluate the model
test_loss, test_acc = model.evaluate(x_test, y_test)
print(f'Test accuracy: {test_acc}')
predictions = model.predict(x_test)

# Plot training history
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```


Training Accuracy
Validation Accuracy
Accuracy
Epoch
0.16
0.15
0.14
0.13
0.12
0.11
0.10
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0

PRACTICAL NO-11

AIM: Character Recognition using LSTM.

```
# Import necessary libraries
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
from tensorflow.keras.datasets import mnist
from tensorflow.keras.utils import plot_model
import matplotlib.pyplot as plt
# 1. Load and preprocess the MNIST dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()
# Normalize the pixel values to be between 0 and 1
x_train = x_train / 255.0
x_test = x_test / 255.0
# Reshape the data for RNN input
# Treat each image as a sequence of 784 steps with 1 feature each
x_train = x_train.reshape(x_train.shape[0], -1, 1)
x_test = x_test.reshape(x_test.shape[0], -1, 1)
# 2. Model Creation
model = Sequential([
# Input LSTM layer
LSTM(50, input_shape=(x_train.shape[1], x_train.shape[2]), return_sequences=True),
# Hidden LSTM layer
LSTM(50),
# Output layer for 10 classes (digits 0-9)
Dense(10, activation='softmax')
])
model.summary()
plot_model(model, to_file='model_plot.png', show_shapes=True,
show_layer_names=True)
# 3. Compile the model
model.compile(optimizer='adam',
loss='sparse_categorical_crossentropy',
metrics=['accuracy'])
# 4. Train the model
history = model.fit(x_train, y_train, epochs=5, validation_data=(x_test, y_test))
# 5. Evaluate the model
test_loss, test_acc = model.evaluate(x_test, y_test)
print(f'Test accuracy: {test_acc}')
# Make predictions on the test set
predictions = model.predict(x_test)
```

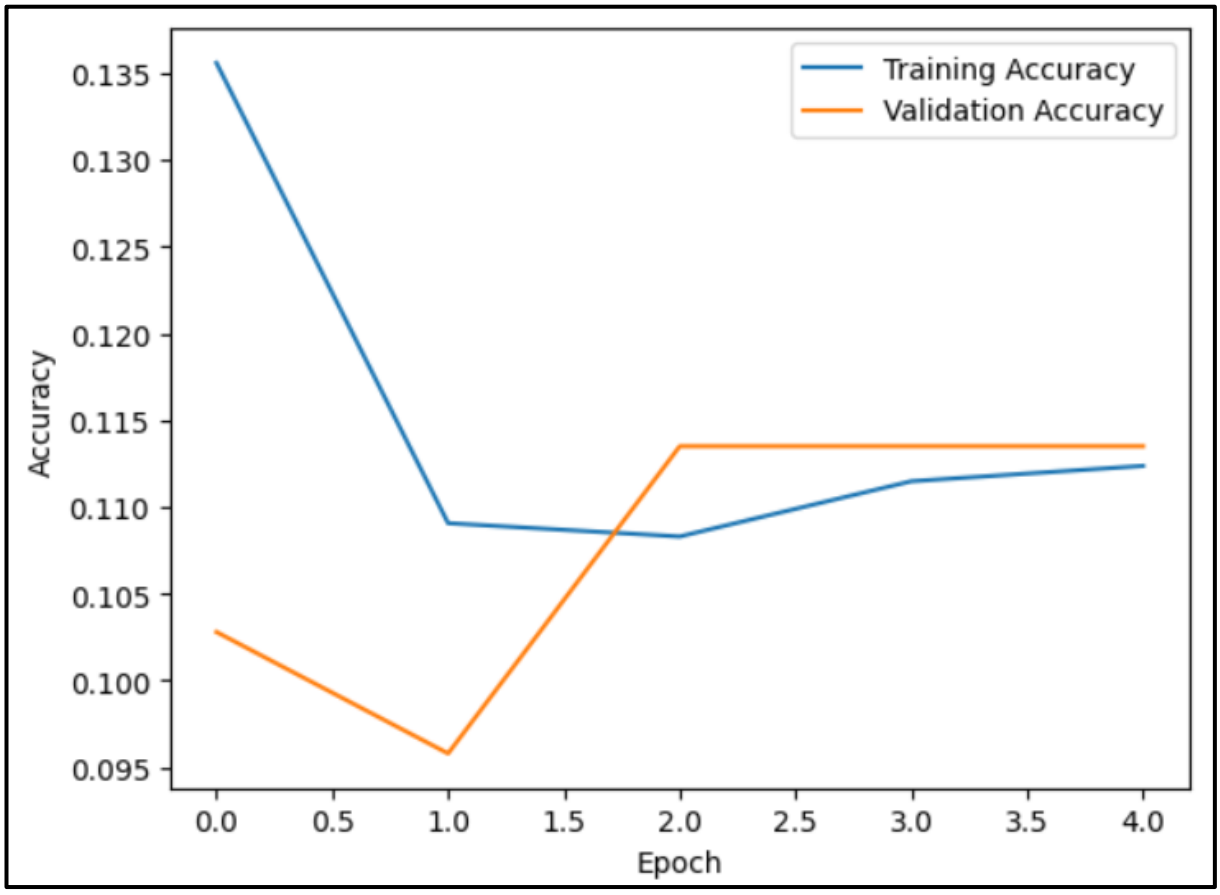
Model: "sequential_8"

Layer (type)	Output Shape	Param #
lstm (LSTM)	(None, 784, 50)	10,400
lstm_1 (LSTM)	(None, 50)	20,200
dense_23 (Dense)	(None, 10)	510

Total params: 31,110 (121.52 KB)
Trainable params: 31,110 (121.52 KB)
Non-trainable params: 0 (0.00 B)

```
Test accuracy: 0.11349999904632568
```

```
# 6. Plot training history
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```



PRACTICAL NO-12

AIM: LSTM Network: Sentiment analysis using LSTM.

Data Loading and Preprocessing

```
import pandas as pd
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense, SpatialDropout1D
from tensorflow.keras.callbacks import EarlyStopping
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
import re
import nltk
from nltk.tokenize import word_tokenize

# Download necessary NLTK data
try:
   nltk.data.find('tokenizers/punkt')
except LookupError:
   nltk.download('punkt')
try:
   nltk.data.find('tokenizers/punkt_tab')
except LookupError:
   nltk.download('punkt_tab')

# Load the dataframe with a specified encoding and handle bad lines
df = pd.read_csv('twitter_training.csv', encoding='ISO-8859-1', header=None,
on_bad_lines='skip')
# 2. Rename column 3 to 'text', and column 2 to 'sentiment'
df.rename(columns={3: 'text', 2: 'sentiment'}, inplace=True)
# 1. Handle missing values in the text column (column 3)
df.dropna(subset=['text'], inplace=True)
# 3. Remove rows where the sentiment is 'Irrelevant'
df = df[df['sentiment'] != 'Irrelevant'].copy()
# 4. Convert the 'sentiment' column to numerical labels
sentiment_mapping = {'Negative': 0, 'Neutral': 1, 'Positive': 2}
df['sentiment_numeric'] = df['sentiment'].map(sentiment_mapping)
# 5. Remove any rows where the sentiment is not one of 'Negative', 'Neutral', or
'Positive' after the previous step
df.dropna(subset=['sentiment_numeric'], inplace=True)

# 6. Perform basic text cleaning on the 'text' column
def clean_text(text):
  text = text.lower() # Convert to lowercase
  text = re.sub(r'http\S+|www\S+|https\S+', '', text, flags=re.MULTILINE) # Remove
URLs
  text = re.sub(r'@\w+', '', text) # Remove mentions
```

```
  text = re.sub(r'#\w+', '', text) # Remove hashtags
  text = re.sub(r'[^a-z0-9\s]', '', text) # Remove special characters
  return text
df['text_cleaned'] = df['text'].apply(clean_text)
```

Tokenization and Sequence Padding

```
# 7. Tokenize the text in the 'text' column
df['text_tokenized'] = df['text_cleaned'].apply(word_tokenize)
# 8. Convert the tokenized text data into numerical representations using TF-IDF
# Join tokens back into strings for TF-IDF
df['text_joined'] = df['text_tokenized'].apply(lambda tokens: ' '.join(tokens))
# 10. Prepare data for LSTM
# Tokenize and pad sequences
max_words = 5000 # Limit vocabulary size
max_len = 250 # Limit sequence length
tokenizer = Tokenizer(num_words=max_words, oov_token="<OOV>")
tokenizer.fit_on_texts(df['text_joined'])
sequences = tokenizer.texts_to_sequences(df['text_joined'])
padded_sequences = pad_sequences(sequences, maxlen=max_len, padding='post',
truncating='post')
# Prepare labels
labels = df['sentiment_numeric'].values
# Split data into training and testing sets
X_train_lstm, X_test_lstm, y_train_lstm, y_test_lstm = train_test_split(padded_sequences, labels,
test_size=0.2, random_state=42)
```

LSTM Model Building, Training, and Evaluation

```
# 11. Build the LSTM model
embedding_dim = 100
model_lstm = Sequential([
Embedding(max_words, embedding_dim, input_length=max_len),
SpatialDropout1D(0.2),
LSTM(100, dropout=0.2, recurrent_dropout=0.2),
Dense(3, activation='softmax') # 3 classes: Negative, Neutral, Positive
])
model_lstm.compile(loss='sparse_categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])

# 12. Train the LSTM model
epochs = 5
batch_size = 64
history = model_lstm.fit(X_train_lstm, y_train_lstm, epochs=epochs, batch_size=batch_size,
validation_split=0.1, callbacks=[EarlyStopping(monitor='val_loss', patience=3)])

# 13. Evaluate the LSTM model
loss, accuracy_lstm = model_lstm.evaluate(X_test_lstm, y_test_lstm, verbose=0)
print(f"LSTM Test Accuracy: {accuracy_lstm:.4f}")
# Make predictions for classification report
y_pred_lstm = np.argmax(model_lstm.predict(X_test_lstm), axis=-1)
```

```
# Print classification report (requires sklearn)
print("\nLSTM Classification Report:")
print(classification_report(y_test_lstm, y_pred_lstm, target_names=['Negative', 'Neutral',
'Positive']))
```

```
LSTM Classification Report:
              precision    recall  f1-score   support

    Negative       0.36      1.00      0.53      4427
     Neutral       0.00      0.00      0.00      3678
    Positive       0.00      0.00      0.00      4120

    accuracy                           0.36     12225
   macro avg       0.12      0.33      0.18     12225
weighted avg       0.13      0.36      0.19     12225
```