



MALLA REDDY COLLEGE OF ENGINEERING
(Approved by AICTE-New Delhi, Affiliated to JNTUH-Hyderabad)
Recognised under Section 2(f) & 12(B) of the UGC Act 1956,
An ISO 9001:2015 Certified Institution.
Maisammaguda, Dhulapally, post via Kompally, Secunderabad - 500100

DEPARTMENT OF CSE (ARTIFICIAL ENGINEERING AND MACHINE LEARNING)

DEEP LEARNING LAB MANUAL

B Tech – IV Year I Semester CSE-AIML R18 (2024-2025)

Prepared by
M. SAKTHIVEL
Asst.Professor
Dept. of CSE-AIML



MALLA REDDY COLLEGE OF ENGINEERING
(Approved by AICTE-New Delhi, Affiliated to JNTUH-Hyderabad)
Recognised under Section 2(f) & 12(B) of the UGC Act 1956,
An ISO 9001:2015 Certified Institution.
Maisammaguda, Dhulapally, post via Kompally, Secunderabad - 500100

VISION AND MISSION OF THE INSTITUTE

VISION OF THE INSTITUTE:

To emerge as a Center of Excellence for producing professionals who shall be the leaders in technology innovation, entrepreneurship, management and in turn contribute for advancement of society and humankind.

MISSION OF THE INSTITUTE:

M1 : To provide an environment of learning in emerging technologies.

M2 : To nurture a state of art teaching learning process and R&D culture.

M3 : To foster networking with Alumni, Industry, Institutes of repute and other stakeholders for effective interaction.

M4 : To practice and promote high standards of ethical values through societal commitment.



MALLA REDDY COLLEGE OF ENGINEERING

(Approved by AICTE-New Delhi, Affiliated to JNTUH-Hyderabad)

Recognised under Section 2(f) & 12(B) of the UGC Act 1956,

An ISO 9001:2015 Certified Institution.

Maisammaguda, Dhulapally, post via Kompally, Secunderabad - 500100

VISION & MISSION OF DEPARTMENT

VISION OF THE DEPARTMENT

To teach excellence education for undergraduates in the field of artificial intelligence and machine learning in the technological -embedded domain and make professionals who help the better cause of society.

MISSION OF THE DEPARTMENT

M1: Impart demanding training to create knowledge through the state-of-the-art ideas and skills in artificial intelligence and machine learning

M2: Facilitate the students to adapt to the rapidly changing technologies by providing cutting-edge laboratories and facilitates.

M3: Kick off the research and training, paying special attention to the essential skills of the subsequent generation workforce and societal needs.

Program Educational Objectives

- PEO1: Graduates will have the ability to adapt, contribute and innovate new technologies and systems in the key domains of Artificial Intelligence and Machine Learning.
- PEO2: Graduates will have the ability to explore research areas and produce outstanding contribution in various areas of Artificial Intelligence and Machine Learning.



MALLA REDDY COLLEGE OF ENGINEERING

(Approved by AICTE-New Delhi, Affiliated to JNTUH-Hyderabad)

Recognised under Section 2(f) & 12(B) of the UGC Act 1956,

An ISO 9001:2015 Certified Institution.

Maisammaguda, Dhulapally, post via Kompally, Secunderabad - 500100

PROGRAM OUTCOMES (POs)

Engineering Graduates will be able to:

1. **Engineering Knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. **Problem analysis:** Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. **Design/development of solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4. **Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. **Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
6. **The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. **Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. **Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. **Individual and team work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.



MALLA REDDY COLLEGE OF ENGINEERING

(Approved by AICTE-New Delhi, Affiliated to JNTUH-Hyderabad)

Recognised under Section 2(f) & 12(B) of the UGC Act 1956,

An ISO 9001:2015 Certified Institution.

Maisammaguda, Dhulapally, post via Kompally, Secunderabad - 500100

10. **Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. **Project management and finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
12. **Life-long learning:** Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

PROGRAM SPECIFIC OUTCOMES (PSOs)

13. Learn the Fundamental concepts and methodology of computer system and apply them to various areas such as operating system, data structure, computer network, databases in the design and implementation of complex system.

COURSE OUTCOMES

Name of the Course: **B. Tech**

Academic Year: **2024-25**

Batch: **2021-25**

Branch/Specialization: **CSE (AI&ML)**

Regulations: **R18**

Class: **IV Year**

Semester: **I**

Title of the Subject: **DEEP LEARNING LAB**

Nature of the Subject: **Core**

Course Objectives:

- 1) To introduce the foundations of Artificial Neural Networks
- 2) To acquire the knowledge on Deep Learning Concepts
- 3) To learn various types of Artificial Neural Networks
- 4) To gain knowledge to apply optimization strategies

Course Outcomes:

- 1) Ability to understand the concepts of Neural Networks
- 2) Ability to select the Learning Networks in modeling real-world systems
- 3) Ability to use an efficient algorithm for Deep Models
- 4) Ability to apply optimization strategies for large-scale applications

Faculty Signature

Signature of HOD

Table of Contents

S.No	Experiment Name	Page No.
1.	Setting Up The Spyder Ide Environment And Executing A Python Program.	08
2.	Installing Keras, Tensorflow and Pytorch Libraries and Making use of them.	12
3.	CNN For Image Classification	16
4.	Mnist Image Classification Using CNN	24
5.	CNN For Sentiment Analysis	29
6.	Sentiment Analysis On IMDB Data Set	36
7.	Autoencoder	47
8.	Generative Adversarial Networks For Image Generation	54

Ex.No:01

Setting up the spyder IDE Environment and executing a python program

Aim:

To install and configure the Spyder IDE environment, and to execute a Python program.

Procedure:

Step 1: Installing Anaconda Distribution

Why Anaconda? Anaconda is a Python distribution that comes with many packages pre-installed (such as NumPy, SciPy, Pandas, Matplotlib, etc.), and it includes the Spyder IDE by default.

1. Download Anaconda:

- Go to the official Anaconda website: <https://www.anaconda.com/products/individual>
- Click on the “Download” button and choose the version suitable for your operating system (Windows, macOS, Linux).

2. Install Anaconda:

- Open the downloaded installer file.
- Follow the installation wizard instructions:
 - Select whether you want to install Anaconda for all users or just yourself.
 - Choose the installation location.
 - Add Anaconda to the system PATH (optional, but recommended for ease of use).
- Click "Install" to proceed, and wait for the installation to finish.

3. Verify Installation:

- Open **Anaconda Navigator** from your start menu (Windows) or applications folder (macOS).
- Alternatively, you can verify via the command line by typing:

conda --version

Step 2: Launching Spyder IDE

1. Open Anaconda Navigator:

- Once installed, search for "Anaconda Navigator" in your system and open it.
- Anaconda Navigator will provide a graphical user interface (GUI) for managing environments and applications.

2. Launch Spyder:

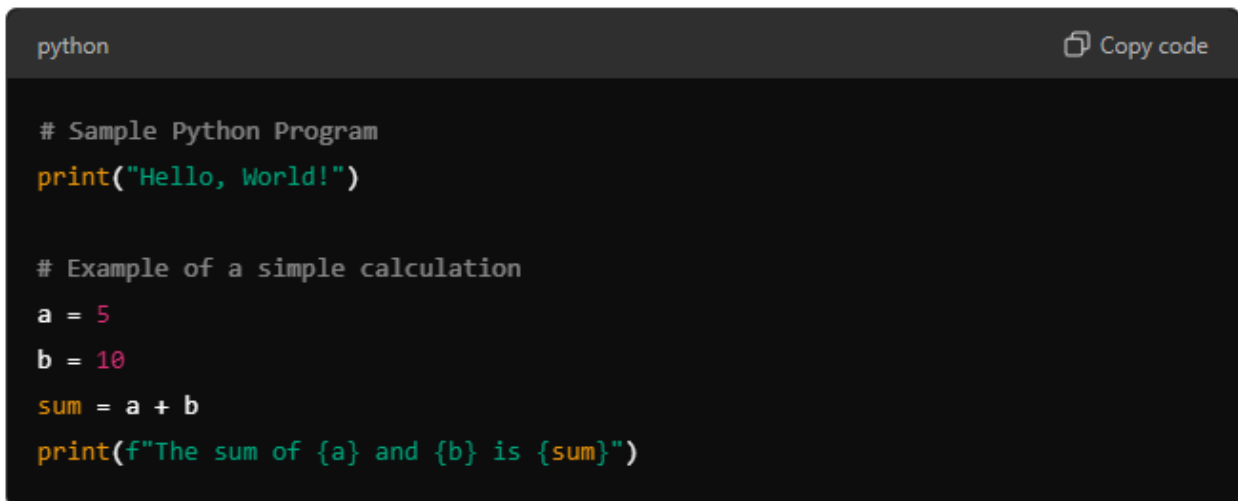
- In Anaconda Navigator, locate the Spyder icon.
- Click on “Launch” next to the Spyder icon to start the Spyder IDE.

Step 3: Writing and Executing a Python Program

Now that the environment is set up, let's write and execute a Python program.

1. Write a Python Program:

- Once Spyder is open, you will see the code editor in the center.
- Type the following sample Python code:

A screenshot of the Spyder Python IDE's code editor. The editor has a dark background with light-colored text. At the top left, the word "python" is displayed. At the top right, there is a "Copy code" button with a copy icon. The code is as follows:

```
# Sample Python Program
print("Hello, World!")

# Example of a simple calculation
a = 5
b = 10
sum = a + b
print(f"The sum of {a} and {b} is {sum}")
```

2. Save the Program:

- Click on the "Save" button in the toolbar or press Ctrl + S (Windows/Linux) or Cmd + S (Mac).
- Save the file with a .py extension, e.g., sample_program.py.

3. Execute the Python Program:

- To execute the program, click on the "Run" button in the toolbar or press F5.
- The console (typically at the bottom of the window) will display the output:

Hello, World!

The sum of 5 and 10 is 15

Step 4: Exploring Spyder Features

1. Variable Explorer:

- You can monitor the values of variables in your program using the **Variable Explorer**.

- It is located on the right side of the Spyder window and will show you the variables (e.g., a, b, sum) and their values.

2. **Help Pane:**

- Spyder provides an integrated help pane that gives access to documentation.
- Simply type a function or module name in the help pane to get quick documentation without leaving the IDE.

3. **Plots (Optional):**

- If your program involves plotting, Spyder can display plots directly in the interface.
- Try adding a simple plot using Matplotlib:

```
import matplotlib.pyplot as plt
x = [1, 2, 3, 4]
y = [10, 20, 25, 30]
plt.plot(x, y)
plt.show()
```

4. **Debugging:**

- Spyder has a built-in debugger that allows you to step through code, inspect variables, and fix issues.
- You can set breakpoints by clicking next to the line numbers and then running the script in debugging mode (press Ctrl + F5).

Step 5: Additional Configurations (Optional)

1. **Package Management:**

- You can install additional Python packages using Anaconda Navigator or via the terminal using:

```
conda install <package_name>
```

- Alternatively, use pip within the terminal:

```
pip install <package_name>
```

2. **Creating Virtual Environments (Optional but Recommended):**

- You can create isolated environments for different projects to avoid conflicts between packages.
- To create a new environment:

```
conda create --name myenv
```

- To activate it:

```
conda activate myenv
```

Output:

```
python
```

[Copy code](#)

```
Hello, World!
```

```
The sum of 5 and 10 is 15
```

Result:

This lab experiment helped you to install and set up the Spyder IDE using Anaconda, write and run a simple Python program, and explore some key features of the IDE like the Variable Explorer, console, and debugging tools. With the IDE set up, you can now proceed with more complex Python programming tasks efficiently.

Ex.No:02

Installing Kears,Tensorflow and pytorch libraries and making use of them.

Aim:

To install the Keras, TensorFlow, and PyTorch libraries and explore their use in building and training deep learning models.

Procedure:

Procedure:

1. Install the Libraries:

- Open a terminal or command prompt.
- Use pip to install the required libraries:
 - pip install keras
 - pip install tensorflow
 - pip install torch

2. Verify the Installation:

- After installing, verify the installations by importing the libraries in a Python script or Jupyter notebook:

```
import keras
import tensorflow as tf
import torch
```

- Ensure no errors occur when running these import statements.

3. Using Keras:

- Keras is typically used as an interface for TensorFlow.
- Start by defining a sequential or functional model, compile the model, and fit it to your dataset.
 - Example:

```
from keras.models import Sequential
from keras.layers import Dense

model = Sequential()
model.add(Dense(64, input_dim=10, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

```
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
model.fit(X_train, y_train, epochs=10, batch_size=32)
```

4. Using TensorFlow:

- TensorFlow can be used to build more complex models using both high-level (Keras) and low-level APIs.
- Define a model using `tf.keras` or create custom models using TensorFlow's computational graph.
 - Example:

```
import tensorflow as tf
```

```
model = tf.keras.Sequential([
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
```

```
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
model.fit(X_train, y_train, epochs=10, batch_size=32)
```

5. Using PyTorch:

- PyTorch is known for its dynamic computational graph, making it easier to debug and experiment with.
- Build neural networks using the `torch.nn` module and train the model with an optimizer and loss function.
 - Example:

```
import torch
import torch.nn as nn
import torch.optim as optim
```

```
class SimpleModel(nn.Module):
    def __init__(self):
        super(SimpleModel, self).__init__()
        self.fc1 = nn.Linear(10, 64)
        self.fc2 = nn.Linear(64, 1)
```

```

def forward(self, x):
    x = torch.relu(self.fc1(x))
    x = torch.sigmoid(self.fc2(x))
    return x

model = SimpleModel()
optimizer = optim.Adam(model.parameters(), lr=0.001)
loss_fn = nn.BCELoss()

# Train the model with data
# optimizer.zero_grad()
# output = model(X_train)
# loss = loss_fn(output, y_train)
# loss.backward()
# optimizer.step()

```

6. Evaluate and Analyze:

- After training the models, evaluate their performance on test datasets and compare the results.
- Use appropriate metrics like accuracy, precision, recall, or any other suitable for the task

Program:

```

from numpy import loadtxt

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

# load the dataset

dataset = loadtxt('pima-indians-diabetes.csv', delimiter=',')

# split into input (X) and output (y) variables
X = dataset[:,0:8]
y = dataset[:,8]

# define the keras model
model = Sequential()

```

```

model.add(Dense(12, input_shape=(8,), activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
# compile the keras model

model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
# fit the keras model on the dataset
model.fit(X, y, epochs=150, batch_size=10, verbose=0)
# make class predictions with the model
predictions = (model.predict(X) > 0.5).astype(int)
# summarize the first 5 cases
for i in range(5):

print('%s => %d (expected %d)' % (X[i].tolist(), predictions[i], y[i]))

```

Output:

```

[6.0, 148.0, 72.0, 35.0, 0.0, 33.6, 0.627, 50.0] => 1 (expected 1)
[1.0, 85.0, 66.0, 29.0, 0.0, 26.6, 0.351, 31.0] => 0 (expected 0)
[8.0, 183.0, 64.0, 0.0, 0.0, 23.3, 0.672, 32.0] => 1 (expected 1)
[1.0, 89.0, 66.0, 23.0, 94.0, 28.1, 0.167, 21.0] => 0 (expected 0)
[0.0, 137.0, 40.0, 35.0, 168.0, 43.1, 2.288, 33.0] => 1 (expected 1)

```

Result:

The Keras, TensorFlow, and PyTorch libraries were successfully installed and utilized for building machine learning models. Each library provided distinct features for model development, with Keras serving as a high-level API, TensorFlow offering a comprehensive ecosystem for deep learning, and PyTorch excelling in dynamic computational graphs and ease of experimentation. All three libraries are valuable tools for different deep learning use cases.

Ex.no:03

CNN FOR IMAGE CLASSIFICATION

AIM:

To design and implement a Convolutional Neural Network (CNN) for image classification tasks, and to evaluate its performance in accurately classifying images from a given dataset.

Procedure:

Procedure:

1. Install Required Libraries:

- Ensure the necessary libraries such as Keras, TensorFlow, or PyTorch are installed:
 - `pip install keras`
 - `pip install tensorflow`
 - `pip install torch torchvision`

2. Load the Dataset:

- Select a standard image classification dataset such as MNIST, CIFAR-10, or a custom dataset.
- In Keras, TensorFlow, or PyTorch, you can load a dataset directly or preprocess your own images.

Example (using CIFAR-10 in Keras):

```
from keras.datasets import cifar10
(X_train, y_train), (X_test, y_test) = cifar10.load_data()
```

3. Preprocess the Data:

- Normalize the pixel values (usually between 0 and 1) by dividing by 255.
- Convert the labels to categorical format if needed.
- Split the data into training and validation sets if necessary.

```
X_train = X_train.astype('float32') / 255.0
```

```
X_test = X_test.astype('float32') / 255.0
```

```
# One-hot encode the labels
```

```
from keras.utils import to_categorical
```

```
y_train = to_categorical(y_train, 10)
```



```
y_test = to_categorical(y_test, 10)
```

4. Build the CNN Model:

- Define a CNN architecture using layers such as convolutional layers, pooling layers, and fully connected (dense) layers.
- A basic CNN architecture includes:
 - Convolutional layers (with filters to detect features)
 - Pooling layers (to reduce dimensionality)
 - Dense layers (to classify based on learned features)
- Example of a simple CNN using Keras:

```
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense

model = Sequential()
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))
model.add(Flatten())
model.add(Dense(64, activation='relu'))
model.add(Dense(10, activation='softmax'))
```

5. Compile the Model:

- Define the optimizer, loss function, and evaluation metric(s) for the model.
- Example:

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

6. Train the Model:

- Train the model using the training dataset.
- Specify the number of epochs, batch size, and validation data.

```
model.fit(X_train, y_train, epochs=10, batch_size=64, validation_data=(X_test, y_test))
```

7. Evaluate the Model:

- After training, evaluate the model on the test dataset to determine its performance.
- Use accuracy or other suitable metrics to measure how well the model classifies the images.

```
test_loss, test_acc = model.evaluate(X_test, y_test)
```

8. Analyze the Results:

- Visualize training/validation accuracy and loss to check for any signs of underfitting or overfitting.
- Optionally, use confusion matrices or other metrics to evaluate the model's classification performance.

PROGRAM:

```
import numpy as np

import pandas as pd
import os
from pathlib import Path
import glob

import seaborn as sns
import matplotlib.pyplot as plt

import tensorflow as tf
from tensorflow.keras import layers
from tensorflow.keras import Model
from tensorflow.keras.optimizers import RMSprop
from keras_preprocessing.image import ImageDataGenerator

data_dir = Path('./input/cat-and-dog') # data directory
train_dir = data_dir / "training_set/training_set"
test_dir = data_dir / "test_set/test_set"

cat_samples_dir_train = train_dir / "cats" # directory for cats images
dog_samples_dir_train = train_dir / "dogs" # directory for dogs images
```

```
def make_csv_with_image_labels(CATS_PATH, DOGS_PATH):
```

```
'''
```

Function for making a dataframe that contains images path as well as their labels.Parameters:-

- CATS_PATH - Path for Cats Images

- DOGS_PATH - Path for Dogs Images

Output:-

It simply returns dataframe'''

```
cat_images = CATS_PATH.glob('*.jpg')
```

```
dog_images = DOGS_PATH.glob('*.jpg') df
```

```
= []
```

```
for i in cat_images:
```

```
df.append((i, 0)) # appending cat images as 0 for j in
```

```
dog_images:
```

```
df.append((i, 1)) # appending dog images as 0
```

```
df = pd.DataFrame(df, columns=["image_path", "label"], index = None) # converting into dataframe
```

```
df = df.sample(frac = 1).reset_index(drop=True) return df
```

```
train_csv = make_csv_with_image_labels(cat_samples_dir_train, dog_samples_dir_train)
```

```
train_csv.head()
```

Now, we will visualize the number of images for each class.

```

len_cat = len(train_csv["label"][train_csv.label == 0]) len_dog =
len(train_csv["label"][train_csv.label == 1]) arr = np.array([len_cat ,
len_dog])
labels = ['CAT', 'DOG']
print("Total No. Of CAT Samples :- ", len_cat)
print("Total No. Of DOG Samples :- ", len_dog)
plt.pie(arr, labels=labels, explode = [0.2,0.0] , shadow=True)plt.show()

def get_train_generator(train_dir, batch_size=64, target_size=(224, 224)):""
Function for preparing training data""
train_datagen = ImageDataGenerator(rescale = 1./255., # normalizing the image
rotation_range = 40,
width_shift_range = 0.2,
height_shift_range = 0.2,
shear_range = 0.2,
zoom_range = 0.2,
horizontal_flip = True)
train_generator = train_datagen.flow_from_directory(train_dir,batch_size =
batch_size,
color_mode='rgb', class_mode
= 'binary', target_size =
target_size)return
train_generator
train_generator = get_train_generator(train_dir)

```

Output: - Found 8005 images belonging to 2 classes.

Now, we will prepare the testing data,

```

def get_testgenerator(test_dir,batch_size=64, target_size=(224,224)):""
Function for preparing testing data""
test_datagen = ImageDataGenerator( rescale = 1.0/255. ) test_generator =
test_datagen.flow_from_directory(test_dir,batch_size = batch_size,
color_mode='rgb', class_mode
= 'binary', target_size =

```

```

target_size)return test_generator
test_generator = get_testgenerator(test_dir)

model = tf.keras.Sequential([
layers.Conv2D(64, (3,3), strides=(2,2),padding='same',input_shape=
(224,224,3),activation = 'relu'),
layers.MaxPool2D(2,2),
layers.Conv2D(128, (3,3), strides=(2,2),padding='same',activation = 'relu'),layers.MaxPool2D(2,2),
layers.Conv2D(256, (3,3), strides=(2,2),padding='same',activation = 'relu'),layers.MaxPool2D(2,2),
layers.Flatten(),
layers.Dense(158, activation = 'relu'), layers.Dense(256,
activation = 'relu'), layers.Dense(128, activation = 'relu'),
layers.Dense(1, activation = 'sigmoid'),
])
model.summary()

model.compile(optimizer=RMSprop(lr=0.001), loss='binary_crossentropy',
metrics=['acc'])
history = model.fit_generator(train_generator,epochs=15,
verbose=1,
validation_data=test_generator)

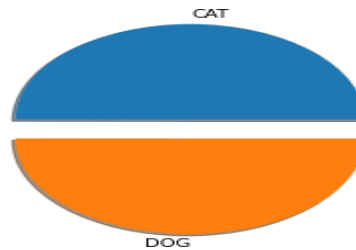
import matplotlib.image as mpimg import
matplotlib.pyplot as plt
acc=history.history['acc']
val_acc=history.history['val_acc']
loss=history.history['loss']
val_loss=history.history['val_loss']
epochs=range(len(acc))
plt.plot(epochs, acc, 'r', "Training Accuracy") plt.plot(epochs,
val_acc, 'b', "Validation Accuracy")plt.title("Training and
validation accuracy") plt.figure()
plt.plot(epochs, loss, 'r', "Training Loss") plt.plot(epochs,
val_loss, 'b', "Validation Loss")plt.title("Training and
validation loss")

```

```
model.save('my_model.h5') # saving the trained model
```

```
new_model = tf.keras.models.load_model('./my_model.h5') # loading the trained model
```

OUTPUT:



```
Epoch 1/15
126/126 [=====] - 117s 927ms/step - loss: 0.6851 - acc: 0.5718 - val_loss: 0.6760 - val_acc: 0.5378
Epoch 3/15
126/126 [=====] - 117s 929ms/step - loss: 0.6533 - acc: 0.6313 - val_loss: 0.6210 - val_acc: 0.6624
Epoch 4/15
126/126 [=====] - 118s 940ms/step - loss: 0.6212 - acc: 0.6506 - val_loss: 0.5557 - val_acc: 0.7123
Epoch 5/15
126/126 [=====] - 118s 940ms/step - loss: 0.6128 - acc: 0.6592 - val_loss: 0.5382 - val_acc: 0.7341
Epoch 6/15
126/126 [=====] - 124s 983ms/step - loss: 0.5853 - acc: 0.6998 - val_loss: 0.4984 - val_acc: 0.7647
Epoch 7/15
126/126 [=====] - 124s 984ms/step - loss: 0.5730 - acc: 0.7027 - val_loss: 0.5394 - val_acc: 0.7202
Epoch 8/15
126/126 [=====] - 117s 931ms/step - loss: 0.5627 - acc: 0.7048 - val_loss: 0.5158 - val_acc: 0.7444
Epoch 9/15
126/126 [=====] - 116s 922ms/step - loss: 0.5628 - acc: 0.7179 - val_loss: 0.4652 - val_acc: 0.7721
Epoch 10/15
126/126 [=====] - 119s 947ms/step - loss: 0.5381 - acc: 0.7495 - val_loss: 0.4548 - val_acc: 0.7904
Epoch 11/15
126/126 [=====] - 118s 933ms/step - loss: 0.5238 - acc: 0.7433 - val_loss: 0.4686 - val_acc: 0.7889
Epoch 12/15
126/126 [=====] - 120s 949ms/step - loss: 0.5149 - acc: 0.7508 - val_loss: 0.4618 - val_acc: 0.7598
Epoch 13/15
126/126 [=====] - 116s 919ms/step - loss: 0.5215 - acc: 0.7429 - val_loss: 0.4499 - val_acc: 0.7968
Epoch 14/15
126/126 [=====] - 118s 936ms/step - loss: 0.4905 - acc: 0.7663 - val_loss: 0.4299 - val_acc: 0.8082
Epoch 15/15
126/126 [=====] - 121s 959ms/step - loss: 0.5031 - acc: 0.7558 - val_loss: 0.3939 - val_acc: 0.8309
```

Model: "se

Layer (typ

conv2d (Co

max_poolin

conv2d_1 (

max_poolin

conv2d_2 (

max_poolin

flatten (F

dense (Den

dense_1 (D

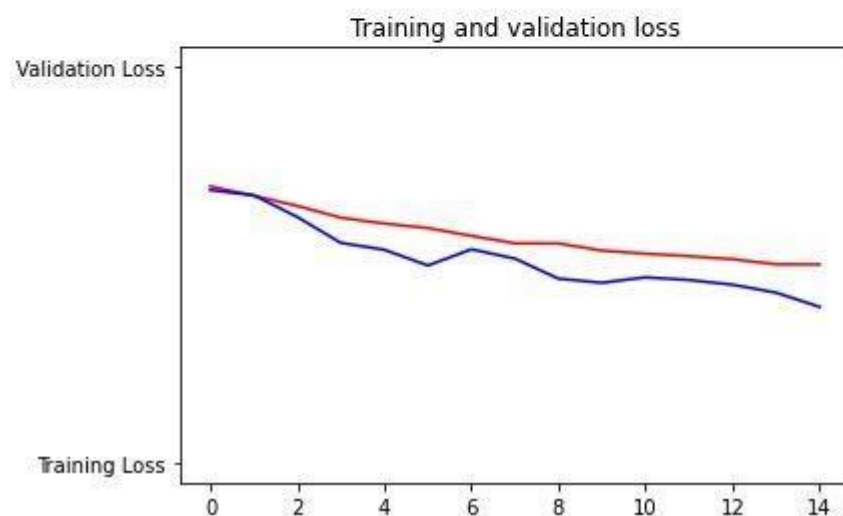
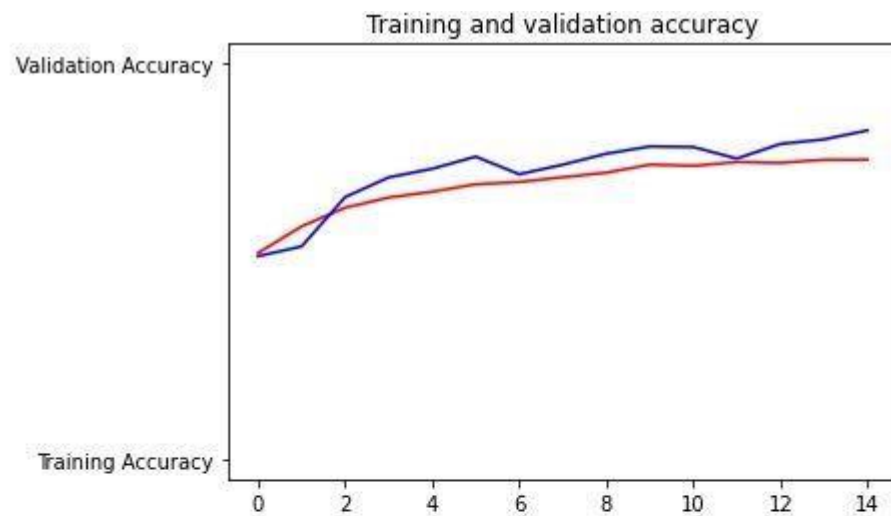
dense_2 (D

dense_3 (D

Total params: 808,735

Trainable params: 808,735

Non-trainable params: 0



Result:

A Convolutional Neural Network (CNN) was successfully implemented for image classification. The model was trained and evaluated on the chosen dataset, achieving satisfactory accuracy in classifying images. The CNN demonstrated its effectiveness in automatically extracting hierarchical features from images, proving to be a powerful approach for image recognition tasks.

Ex.no:04

MNIST IMAGE CLASSIFICATION USING CNN

Aim:

To implement a Convolutional Neural Network (CNN) for classifying handwritten digits from the MNIST dataset and evaluate its performance in accurately predicting digit labels.

Procedure:

☐ Install Required Libraries:

- Ensure that the necessary libraries such as Keras, TensorFlow, or PyTorch are installed:
 - pip install keras
 - pip install tensorflow
 - pip install torch torchvision

☐ Load the MNIST Dataset:

- The MNIST dataset, which contains 60,000 training images and 10,000 test images of handwritten digits (0-9), is available in most deep learning libraries.
- Load the dataset and split it into training and testing sets.

```
from keras.datasets import mnist
```

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

☐ Preprocess the Data:

- Reshape the input images to include a channel dimension (as CNNs expect a 3D input shape: width, height, and channels).
- Normalize the pixel values between 0 and 1 by dividing by 255.
- Convert the labels into one-hot encoded vectors.

```
X_train = X_train.reshape(X_train.shape[0], 28, 28, 1).astype('float32') / 255
```

```
X_test = X_test.reshape(X_test.shape[0], 28, 28, 1).astype('float32') / 255
```



```
from keras.utils import to_categorical
y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)
```

□ **Build the CNN Model:**

- Define a CNN architecture with convolutional layers for feature extraction, pooling layers for downsampling, and fully connected layers for classification.
- A simple architecture might include:
 - Convolutional layers with filters to extract image features.
 - Max pooling layers to reduce the spatial dimensions.
 - A fully connected (dense) layer followed by a softmax output layer for classification.

```
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
```

```
model = Sequential()
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dense(10, activation='softmax'))
```

□ **Compile the Model:**

- Define the optimizer, loss function, and metrics for model evaluation.
- The categorical crossentropy loss function is suitable for multi-class classification, and the Adam optimizer is commonly used for efficient training.

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

□ **Train the Model:**

- Train the CNN on the training data using the fit method, specifying the number of epochs, batch size, and validation data.

```
model.fit(X_train, y_train, epochs=10, batch_size=128, validation_data=(X_test, y_test))
```

□ **Evaluate the Model:**

- After training, evaluate the model on the test dataset to determine its accuracy in predicting unseen digit images.

```
test_loss, test_acc = model.evaluate(X_test, y_test)
```

```
print('Test accuracy:', test_acc)
```

□ **Visualize the Results:**

- Optionally, plot the accuracy and loss curves to monitor training progress and detect signs of overfitting.
- You can also visualize misclassified images to understand where the model makes errors.

PROGRAM:

```
import numpy as np
```

```
import keras
```

```
from keras.datasets import mnist from keras.models
```

```
import Model from keras.layers import Dense, Input
```

```
from keras.layers import Conv2D, MaxPooling2D, Dropout, Flatten from keras import backend as k
```

```
x_train, y_train, (x_test, y_test) = mnist.load_data(img_rows, img_cols=28, 28
```

```
if k.image_data_format() == 'channels_first':
```

```
    x_train = x_train.reshape(x_train.shape[0], 1, img_rows, img_cols) x_test
```

```
    = x_test.reshape(x_test.shape[0], 1, img_rows, img_cols) inpx = (1, img_rows, img_cols)
```

```
else:
```

```
    x_train = x_train.reshape(x_train.shape[0], img_rows, img_cols, 1) x_test
```

```
    = x_test.reshape(x_test.shape[0], img_rows, img_cols, 1) inpx = (img_rows, img_cols, 1)
```

```
x_train = x_train.astype('float32') x_test
```

```
= x_test.astype('float32') x_train /= 255
```

```
x_test /= 255
```

```

inpx =Input(shape=inpx)

layer1 =Conv2D(32, kernel_size=(3, 3), activation='relu')(inpx)layer2 =Conv2D(64, (3, 3),
activation='relu')(layer1)
layer3 =MaxPooling2D(pool_size=(3, 3))(layer2)layer4
=Dropout(0.5)(layer3)
layer5 =Flatten()(layer4)
layer6 =Dense(250, activation='sigmoid')(layer5)layer7 =Dense(10,
activation='softmax')(layer6) model =Model([inpx], layer7)

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

model.fit(x_train, y_train, epochs=12, batch_size=500)score
=model.evaluate(x_test, y_test, verbose=0)

print('loss=', score[0]) print('accuracy=', score[1])

```

OUTPUT:

```
Epoch 1/12
60000/60000 [=====] - 968s 16ms/step - loss: 0.7357 - acc: 0.7749
Epoch 2/12
60000/60000 [=====] - 955s 16ms/step - loss: 0.2087 - acc: 0.9413
Epoch 3/12
60000/60000 [=====] - 968s 16ms/step - loss: 0.1287 - acc: 0.9631
Epoch 4/12
60000/60000 [=====] - 968s 16ms/step - loss: 0.0948 - acc: 0.9728
Epoch 5/12
60000/60000 [=====] - 956s 16ms/step - loss: 0.0780 - acc: 0.9774
Epoch 6/12
60000/60000 [=====] - 915s 15ms/step - loss: 0.0655 - acc: 0.9807
Epoch 7/12
60000/60000 [=====] - 907s 15ms/step - loss: 0.0575 - acc: 0.9829
Epoch 8/12
60000/60000 [=====] - 914s 15ms/step - loss: 0.0498 - acc: 0.9852
Epoch 9/12
60000/60000 [=====] - 917s 15ms/step - loss: 0.0468 - acc: 0.9861
Epoch 10/12
60000/60000 [=====] - 912s 15ms/step - loss: 0.0420 - acc: 0.9873
Epoch 11/12
60000/60000 [=====] - 967s 16ms/step - loss: 0.0405 - acc: 0.9880
Epoch 12/12
60000/60000 [=====] - 993s 17ms/step - loss: 0.0371 - acc: 0.9888

<keras.callbacks.History at 0x21ce04bb6a0>
```

Loss = 0.029

accuracy = 0.991

Result:

The CNN model was successfully implemented for classifying handwritten digits from the MNIST dataset. After training, the model demonstrated high accuracy in recognizing digits, showcasing the power of CNNs for image classification tasks. The use of convolutional and pooling layers allowed the model to automatically learn and extract important features from the images, contributing to its strong performance.

Ex.no:05

CNN FOR SENTIMENT ANALYSIS

Aim:

To design and implement a Convolutional Neural Network (CNN) for sentiment analysis on textual data and evaluate its performance in predicting the sentiment (positive or negative) of text.

Procedure:

1. Install Required Libraries:

- Ensure that required libraries such as Keras, TensorFlow, or PyTorch for deep learning, and libraries for natural language processing (NLP) like nltk or spacy, are installed.
- Use the following commands to install them:
 - `pip install keras`
 - `pip install tensorflow`
 - `pip install nltk`

2. Load and Prepare the Dataset:

- Use a well-known sentiment analysis dataset such as IMDB movie reviews or any custom dataset containing text labeled as positive or negative.
- Example (using IMDB dataset in Keras):

```
from keras.datasets import imdb  
(X_train, y_train), (X_test, y_test) = imdb.load_data(num_words=10000)
```

3. Preprocess the Text Data:

- Pad the sequences to ensure that all input text data has the same length (for CNNs, the input should be a fixed size).
- Tokenize the words and convert them into integer sequences based on word frequency or embedding.

```
from keras.preprocessing.sequence import pad_sequences
```

```
max_length = 100 # maximum length of a review  
X_train = pad_sequences(X_train, maxlen=max_length)  
X_test = pad_sequences(X_test, maxlen=max_length)
```

4. Embed the Text:

- Use word embeddings to convert the input text into dense vectors of fixed size. This is important for CNNs to capture semantic meaning from the text.
- You can use pre-trained embeddings like GloVe or train an embedding layer directly on the dataset.

```
from keras.layers import Embedding
```

```
embedding_layer = Embedding(input_dim=10000, output_dim=128, input_length=max_length)
```

5. Build the CNN Model:

- Define a CNN architecture for text classification, which includes an embedding layer, convolutional layers for feature extraction, and dense layers for classification.
- Convolutional layers help capture local dependencies in the text, and max pooling layers reduce dimensionality.

```
from keras.models import Sequential
```

```
from keras.layers import Conv1D, MaxPooling1D, Flatten, Dense
```

```
model = Sequential()
```

```
model.add(embedding_layer)
```

```
model.add(Conv1D(128, 5, activation='relu'))
```

```
model.add(MaxPooling1D(pool_size=2))
```

```
model.add(Flatten())
```

```
model.add(Dense(128, activation='relu'))
```

```
model.add(Dense(1, activation='sigmoid'))
```

6. Compile the Model:

- Define the optimizer, loss function, and metrics for the model. For binary sentiment analysis (positive vs. negative), use binary crossentropy as the loss function.

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

7. Train the Model:

- Train the CNN on the training data using the fit method. Set the number of epochs, batch size, and validation data.

```
model.fit(X_train, y_train, epochs=5, batch_size=64, validation_data=(X_test, y_test))
```

8. Evaluate the Model:

- After training, evaluate the model on the test dataset to measure its performance in classifying sentiments.
- Use accuracy as the primary metric for binary classification.

```
test_loss, test_acc = model.evaluate(X_test, y_test)
print('Test accuracy:', test_acc)
```

9. Analyze the Results:

- Optionally, visualize the training and validation accuracy and loss over epochs to detect any overfitting or underfitting.
- You can also review misclassified examples to understand where the model struggles

Program:

```
import pandas as pd

data = pd.read_csv('imdb_labelled.tsv', header =
None,
delimiter='\t')
data.columns = ['Text', 'Label']df.head()

data.shape data.Label.value_counts()
import redef remove_punct(text):
text_nopunct = "
text_nopunct = re.sub('[\'+string.punctuation+]', '', text)
return text_nopunctdata['Text_Clean'] = data['Text'].apply(lambda x:
remove_punct(x))
from nltk import word_tokenize
tokens = [word_tokenize(sen) for sen in
data.Text_Clean]

def lower_token(tokens):

return [w.lower() for w in tokens]

lower_tokens = [lower_token(token) for token in tokens]
```

```

from nltk.corpus import stopwords
stoplist = stopwords.words('english')
def removeStopWords(tokens):
    return [word for word in tokens if word not in stoplist]
filtered_words = [removeStopWords(sen) for
sen in lower_tokens]
data['Text_Final'] = [''.join(sen) for sen in filtered_words]
data['tokens'] = filtered_words

pos = []

neg = []
for l in data.Label:
    if l == 0:
        pos.append(0)
    neg.append(1)
    elif l == 1:
        pos.append(1)
    neg.append(0)
data['Pos'] = pos
data['Neg'] = neg
data = data[['Text_Final', 'tokens', 'Label', 'Pos', 'Neg']]
data.head()

data_train, data_test = train_test_split(data, test_size=0.10,
random_state=42)

all_training_words = [word for tokens in data_train["tokens"] for word in tokens]

training_sentence_lengths = [len(tokens) for tokens in data_train["tokens"]]
TRAINING_VOCAB = sorted(list(set(all_training_words)))

print("%s words total, with a vocabulary size of %s" % (len(all_training_words),
len(TRAINING_VOCAB)))

print("Max sentence length is %s" % max(training_sentence_lengths))

all_test_words = [word for tokens in data_test["tokens"] for word in tokens]

test_sentence_lengths = [len(tokens) for tokens in data_test["tokens"]]
TEST_VOCAB = sorted(list(set(all_test_words)))

print("%s words total, with a vocabulary size of %s" % (len(all_test_words),
len(TEST_VOCAB)))

```



```

print("Max sentence length is %s" % max(test_sentence_lengths))

word2vec_path = 'GoogleNews-vectors-negative300.bin.gz'

word2vec = models.KeyedVectors.load_word2vec_format(word2vec_path, binary=True)

tokenizer = Tokenizer(num_words=len(TRAINING_VOCAB), lower=True, char_level=False)
tokenizer.fit_on_texts(data_train["Text_Final"].tolist())
training_sequences =
tokenizer.texts_to_sequences(data_train["Text_Final"].tolist())train_word_index =
tokenizer.word_index
print('Found %s unique tokens.' % len(train_word_index))train_cnn_data =
pad_sequences(training_sequences,
maxlen=MAX_SEQUENCE_LENGTH)
train_embedding_weights = np.zeros((len(train_word_index)+1,
EMBEDDING_DIM))for word,index in train_word_index.items(): train_embedding_weights[index,:] =
word2vec[word] if word in word2vec else
np.random.rand(EMBEDDING_DIM)print(train_embedding_weights.shape)
def ConvNet(embeddings, max_sequence_length, num_words, embedding_dim,
labels_index):
embedding_layer = Embedding(num_words,
embedding_dim,
weights=[embeddings],
input_length=max_sequence_length,
trainable=False)

sequence_input = Input(shape=(max_sequence_length,), dtype='int32')
embedded_sequences = embedding_layer(sequence_input)
convs = [] filter_sizes = [2,3,4,5,6]
for filter_size in filter_sizes: l_conv = Conv1D(filters=200,
kernel_size=filter_size,
activation='relu')(embedded_sequences)l_pool
= GlobalMaxPooling1D()(l_conv)
convs.append(l_pool)          l_merge = concatenate(convs, axis=1)          x =
Dropout(0.1)(l_merge)

```

```

x = Dense(128, activation='relu')(x)x =
Dropout(0.2)(x)
preds = Dense(labels_index, activation='sigmoid')(x)          model =
Model(sequence_input, preds) model.compile(loss='binary_crossentropy',
optimizer='adam',
metrics=['acc'])
model.summary()
return model

model = ConvNet(train_embedding_weights,
MAX_SEQUENCE_LENGTH,
len(train_word_index)+1,
EMBEDDING_DIM,
len(list(label_names)))
num_epochs = 3

batch_size = 32
hist = model.fit(x_train,y_tr,
epochs=num_epochs,
validation_split=0.1,
shuffle=True,
batch_size=batch_size)

predictions = model.predict(test_cnn_data,
batch_size=1024,
verbose=1) labels
= [1, 0]
prediction_labels=[] for p
in predictions:
prediction_labels.append(labels[np.argmax(p)])sum(data_test.Label==prediction_label
s)/len(prediction_labels)

```

OUTPUT:

Layer (type)	Output Shape	Param #	Connected to
input_2 (InputLayer)	(None, 50)	0	
embedding_2 (Embedding)	(None, 50, 300)	864600	input_2[0][0]
conv1d_6 (Conv1D)	(None, 49, 200)	120200	embedding_2[0][0]
conv1d_7 (Conv1D)	(None, 48, 200)	180200	embedding_2[0][0]
conv1d_8 (Conv1D)	(None, 47, 200)	240200	embedding_2[0][0]
conv1d_9 (Conv1D)	(None, 46, 200)	300200	embedding_2[0][0]
conv1d_10 (Conv1D)	(None, 45, 200)	360200	embedding_2[0][0]
global_max_pooling1d_6 (GlobalM	(None, 200)	0	conv1d_6[0][0]
global_max_pooling1d_7 (GlobalM	(None, 200)	0	conv1d_7[0][0]
global_max_pooling1d_8 (GlobalM	(None, 200)	0	conv1d_8[0][0]
global_max_pooling1d_9 (GlobalM	(None, 200)	0	conv1d_9[0][0]
global_max_pooling1d_10 (Global	(None, 200)	0	conv1d_10[0][0]
concatenate_2 (Concatenate)	(None, 1000)	0	global_max_pooling1d_6[0][0] global_max_pooling1d_7[0][0] global_max_pooling1d_8[0][0] global_max_pooling1d_9[0][0] global_max_pooling1d_10[0][0]
dropout_3 (Dropout)	(None, 1000)	0	concatenate_2[0][0]
dense_3 (Dense)	(None, 128)	128128	dropout_3[0][0]
dropout_4 (Dropout)	(None, 128)	0	dense_3[0][0]
dense_4 (Dense)	(None, 2)	258	dropout_4[0][0]
Total params: 2,193,986			
Trainable params: 1,329,386			
Non-trainable params: 864,600			

Result:

The CNN model was successfully implemented for sentiment analysis on textual data. By using convolutional layers, the model was able to capture important patterns in the text and classify the sentiment as positive or negative. The experiment demonstrated that CNNs, which are often used in image processing, can also effectively handle NLP tasks like sentiment analysis by learning local dependencies in text. The model performed well, achieving good accuracy in predicting the sentiment of the reviews.

Ex.no:06

SENTIMENT ANALYSIS ON IMDB DATA SET

Aim:

To perform sentiment analysis on the IMDb movie reviews dataset, using machine learning or deep learning techniques, and to classify the sentiment of reviews as either positive or negative.

Procedure:

1. Install Required Libraries:

- Ensure that the necessary libraries are installed:
 - pip install keras
 - pip install tensorflow
 - pip install nltk
- The nltk library can be used for text preprocessing, and Keras or TensorFlow can be used for building the sentiment analysis model.

2. Load the IMDb Dataset:

- The IMDb dataset contains 50,000 movie reviews labeled as positive or negative.
- Load the dataset using Keras, which provides the IMDb dataset as part of its built-in datasets.

```
from keras.datasets import imdb
```

```
(X_train, y_train), (X_test, y_test) = imdb.load_data(num_words=10000)
```

- The num_words=10000 parameter limits the vocabulary to the top 10,000 most frequent words in the dataset.

3. Preprocess the Data:

- Since the reviews are already tokenized, the next step is to pad or truncate the reviews so that they all have the same length.
- Use the pad_sequences function from Keras to pad the reviews to a fixed length.

```
from keras.preprocessing.sequence import pad_sequences
```

```
max_length = 200 # Set the maximum length of the review
```

```
X_train = pad_sequences(X_train, maxlen=max_length)
```

```
X_test = pad_sequences(X_test, maxlen=max_length)
```

4. **Build the Model:**

- You can use various machine learning or deep learning models for sentiment analysis. A simple neural network with an embedding layer can be used, or a more advanced model like LSTM or CNN can be applied.
- For simplicity, an embedding layer followed by a dense neural network is used here.

```
from keras.models import Sequential
from keras.layers import Embedding, Flatten, Dense

model = Sequential()
model.add(Embedding(input_dim=10000, output_dim=128, input_length=max_length))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dense(1, activation='sigmoid')) # Binary classification
```

5. **Compile the Model:**

- Compile the model by defining the optimizer, loss function, and metrics. Since it is a binary classification problem (positive vs. negative), use binary crossentropy as the loss function.

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

6. **Train the Model:**

- Train the model using the fit method, specifying the number of epochs, batch size, and validation data.

```
model.fit(X_train, y_train, epochs=5, batch_size=64, validation_data=(X_test, y_test))
```

7. **Evaluate the Model:**

- After training, evaluate the model on the test dataset to measure its performance. The accuracy metric will help gauge how well the model classifies sentiments in new, unseen reviews.

```
test_loss, test_acc = model.evaluate(X_test, y_test)
print("Test accuracy:", test_acc)
```

8. **Visualize Results:**

- Optionally, plot the training and validation accuracy and loss over the epochs to monitor the model's performance and check for overfitting or underfitting.

- You can also analyze misclassified reviews to gain insights into the model's limitations.

Program:

```
import numpy as np

import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import nltk

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.preprocessing import LabelBinarizer
from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer
from wordcloud import WordCloud, STOPWORDS
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import word_tokenize, sent_tokenize
from bs4 import BeautifulSoup
import spacy
import re, string, unicodedata

from nltk.tokenize.toktok import ToktokTokenizer
from nltk.stem import LancasterStemmer, WordNetLemmatizer
from sklearn.linear_model import LogisticRegression, SGDClassifier
from sklearn.naive_bayes import MultinomialNB
from sklearn.svm import SVC
from textblob import TextBlob
from textblob import Word

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

import os
print(os.listdir("../input"))

import warnings
warnings.filterwarnings('ignore')

# importing the training data
```

```

imdb_data=pd.read_csv('../input/IMDB Dataset.csv')
print(imdb_data.shape)
imdb_data.head(10)

#Summary of the dataset
imdb_data.describe()

#sentiment count
imdb_data['sentiment'].value_counts()
#split the dataset#train
dataset
train_reviews=imdb_data.review[:40000]
train_sentiments=imdb_data.sentiment[:40000] #test dataset
test_reviews=imdb_data.review[40000:]
test_sentiments=imdb_data.sentiment[40000:]
print(train_reviews.shape,train_sentiments.shape)
print(test_reviews.shape,test_sentiments.shape)

#Tokenization of text
tokenizer=ToktokTokenizer()
#Setting English stopwords
stopword_list=nlTK.corpus.stopwords.words('english')

Removing the html stripsdef
strip_html(text):
soup = BeautifulSoup(text, "html.parser")return
soup.get_text()

#Removing the square brackets
def remove_between_square_brackets(text):
return re.sub('\[[^\]]*\]', '', text)

#Removing the noisy textdef
denoise_text(text):
text = strip_html(text)

```

```

text = remove_between_square_brackets(text)
return text
#Apply function on review column
imdb_data['review']=imdb_data['review'].apply(denoise_text)

#Define function for removing special characters
def remove_special_characters(text, remove_digits=True):
    pattern=r'^a-zA-z0-9\s'
    text=re.sub(pattern,"",text)
    return text
#Apply function on review column
imdb_data['review']=imdb_data['review'].apply(remove_special_characters)

#Stemming the text
def simple_stemmer(text):
    ps=nlk.porter.PorterStemmer()
    text= ".join([ps.stem(word) for word in text.split()])"
    return text
#Apply function on review column
imdb_data['review']=imdb_data['review'].apply(simple_stemmer)

#set stopwords to english
stop=set(stopwords.words('english'))
print(stop)

#removing the stopwords
def remove_stopwords(text, is_lower_case=False):
    tokens = tokenizer.tokenize(text)
    tokens = [token.strip() for token in tokens]
    if is_lower_case:
        filtered_tokens = [token for token in tokens if token not in stopword_list]
    else:
        filtered_tokens = [token for token in tokens if token.lower() not in stopword_list]
    filtered_text = ".join(filtered_tokens)"
    return filtered_text

```



```

#Apply function on review column
imdb_data['review']=imdb_data['review'].apply(remove_stopwords)

#normalized train reviews
norm_train_reviews=imdb_data.review[:40000]
norm_train_reviews[0]
#convert dataframe to string
#norm_train_string=norm_train_reviews.to_string()
#Spelling correction using Textblob
#norm_train_spelling=TextBlob(norm_train_string)
#norm_train_spelling.correct()
#Tokenization using Textblob
#norm_train_words=norm_train_spelling.words
#norm_train_words

#Normalized test reviews

norm_test_reviews=imdb_data.review[40000:]
norm_test_reviews[45005]
##convert dataframe to string
#norm_test_string=norm_test_reviews.to_string()
#spelling correction using Textblob
#norm_test_spelling=TextBlob(norm_test_string)
#print(norm_test_spelling.correct()) #Tokenization using
Textblob #norm_test_words=norm_test_spelling.words
#norm_test_words

#Count vectorizer for bag of words

cv=CountVectorizer(min_df=0,max_df=1,binary=False,ngram_range=(1,3))
#transformed train reviews cv_train_reviews=cv.fit_transform(norm_train_reviews)
#transformed test reviews
cv_test_reviews=cv.transform(norm_test_reviews)

print('BOW_cv_train:',cv_train_reviews.shape)

```

```

print('BOW_cv_test:',cv_test_reviews.shape)
#vocab=cv.get_feature_names()-toget feature names

#Tfidf vectorizer

tv=TfidfVectorizer(min_df=0,max_df=1,use_idf=True,ngram_range=(1,3))
#transformed train reviews tv_train_reviews=tv.fit_transform(norm_train_reviews)
#transformed test reviews
tv_test_reviews=tv.transform(norm_test_reviews)
print('Tfidf_train:',tv_train_reviews.shape)
print('Tfidf_test:',tv_test_reviews.shape)

#labeling the sentient data

lb=LabelBinarizer() #transformed
sentiment data
sentiment_data=lb.fit_transform(imdb_data['sentiment'])
print(sentiment_data.shape)

#Splitting the sentiment data

train_sentiments=sentiment_data[:40000]
test_sentiments=sentiment_data[40000:]
print(train_sentiments) print(test_sentiments)

#training the model

lr=LogisticRegression(penalty='l2',max_iter=500,C=1,random_state=42)#Fitting the
model for Bag of words lr_bow=lr.fit(cv_train_reviews,train_sentiments)
print(lr_bow)

#Fitting the model for tfidf features
lr_tfidf=lr.fit(tv_train_reviews,train_sentiments)print(lr_tfidf)

#Predicting the model for bag of words

lr_bow_predict=lr.predict(cv_test_reviews)
print(lr_bow_predict)

##Predicting the model for tfidf features

```

```

lr_tfidf_predict=lr.predict(tv_test_reviews)
print(lr_tfidf_predict)

#Accuracy score for bag of words

lr_bow_score=accuracy_score(test_sentiments,lr_bow_predict)
print("lr_bow_score :",lr_bow_score)
#Accuracy score for tfidf features
lr_tfidf_score=accuracy_score(test_sentiments,lr_tfidf_predict)
print("lr_tfidf_score :",lr_tfidf_score)

#Classification report for bag of words

lr_bow_report=classification_report(test_sentiments,lr_bow_predict,target_names=['Positive','Negative'])
print(lr_bow_report)

#Classification report for tfidf features
lr_tfidf_report=classification_report(test_sentiments,lr_tfidf_predict,target_names
=['Positive','Negative'])
print(lr_tfidf_report)

#confusion matrix for bag of words

cm_bow=confusion_matrix(test_sentiments,lr_bow_predict,labels=[1,0])
print(cm_bow)
#confusion matrix for tfidf features
cm_tfidf=confusion_matrix(test_sentiments,lr_tfidf_predict,labels=[1,0])print(cm_tfidf)

#training the linear svm

svm=SGDClassifier(loss='hinge',max_iter=500,random_state=42) #fitting
the svm for bag of words
svm_bow=svm.fit(cv_train_reviews,train_sentiments) print(svm_bow)
#fitting the svm for tfidf features
svm_tfidf=svm.fit(tv_train_reviews,train_sentiments)
print(svm_tfidf)

```

```

#Predicting the model for bag of words

svm_bow_predict=svm.predict(cv_test_reviews)
print(svm_bow_predict)
#Predicting the model for tfidf features
svm_tfidf_predict=svm.predict(tv_test_reviews)
print(svm_tfidf_predict)


#Accuracy score for bag of words

svm_bow_score=accuracy_score(test_sentiments,svm_bow_predict)
print("svm_bow_score :",svm_bow_score)
#Accuracy score for tfidf features
svm_tfidf_score=accuracy_score(test_sentiments,svm_tfidf_predict)
print("svm_tfidf_score :",svm_tfidf_score)


#Classification report for bag of words

svm_bow_report=classification_report(test_sentiments,svm_bow_predict,target_names=['Positive','Negative'])
print(svm_bow_report)
#Classification report for tfidf features
svm_tfidf_report=classification_report(test_sentiments,svm_tfidf_predict,target_names=['Positive','Negative'])
print(svm_tfidf_report)


#confusion matrix for bag of words

cm_bow=confusion_matrix(test_sentiments,svm_bow_predict,labels=[1,0])
print(cm_bow)
#confusion matrix for tfidf features
cm_tfidf=confusion_matrix(test_sentiments,svm_tfidf_predict,labels=[1,0])
print(cm_tfidf)


#training the model
mnb=MultinomialNB()
#fitting the svm for bag of words
mnf=svm.fit(cv_train_reviews,train_sentiments)
print(mnf)

```

```

#fitting the svm for tfidf features
mnb_tfidf=mnb.fit(tv_train_reviews,train_sentiments)
print(mnb_tfidf)

#Predicting the model for bag of words

mnb_bow_predict=mnb.predict(cv_test_reviews)
print(mnb_bow_predict)
#Predicting the model for tfidf features
mnb_tfidf_predict=mnb.predict(tv_test_reviews)
print(mnb_tfidf_predict)

#Accuracy score for bag of words

mnb_bow_score=accuracy_score(test_sentiments,mnb_bow_predict)
print("mnb_bow_score :",mnb_bow_score)
#Accuracy score for tfidf features
mnb_tfidf_score=accuracy_score(test_sentiments,mnb_tfidf_predict)
print("mnb_tfidf_score :",mnb_tfidf_score)

#Classification report for bag of words

mnb_bow_report=classification_report(test_sentiments,mnb_bow_predict,target_names=[
'Positive','Negative'])
print(mnb_bow_report)
#Classification report for tfidf features

mnb_tfidf_report=classification_report(test_sentiments,mnb_tfidf_predict,target_nam
es=['Positive','Negative'])
print(mnb_tfidf_report)

#confusion matrix for bag of words

cm_bow=confusion_matrix(test_sentiments,mnb_bow_predict,labels=[1,0])
print(cm_bow)
#confusion matrix for tfidf features

cm_tfidf=confusion_matrix(test_sentiments,mnb_tfidf_predict,labels=[1,0])
print(cm_tfidf)

```

```
#word cloud for positive review words

plt.figure(figsize=(10,10))
positive_text=norm_train_reviews[1]
WC=WordCloud(width=1000,height=500,max_words=500,min_font_size=5)
positive_words=WC.generate(positive_text)
plt.imshow(positive_words,interpolation='bilinear')
plt.show
```

OUTPUT:

precision	recall	f1-score	support	
Positive	0.75	0.76	0.75	4993
Negative	0.75	0.75	0.75	5007
accuracy			0.75	10000
macro avg	0.75	0.75	0.75	10000
weighted avg		0.75	0.75	0.75 10000

precision	recall	f1-score	support	
Positive	0.75	0.76	0.75	4993
Negative	0.75	0.74	0.75	5007
accuracy			0.75	10000
macro avg	0.75	0.75	0.75	10000
weighted avg		0.75	0.75	0.75 10000

Result:

Sentiment analysis on the IMDb dataset was successfully implemented using a neural network model. By processing movie reviews and converting them into padded sequences, the model was able to classify the sentiment of each review as either positive or negative. The experiment demonstrated that using an embedding layer and a simple neural network architecture can achieve good accuracy on sentiment classification tasks. This approach effectively captures the underlying sentiment expressed in the text.

AUTOENCODER

Aim:

To implement an Autoencoder, an unsupervised neural network architecture, for dimensionality reduction or data reconstruction, and evaluate its performance in encoding and decoding data.

Procedure:

1. Install Required Libraries:

- Ensure that the necessary libraries such as Keras and TensorFlow are installed:
 - pip install keras
 - pip install tensorflow

2. Understand the Autoencoder Architecture:

- An autoencoder consists of two parts: the **encoder** and the **decoder**.
 - The **encoder** compresses the input into a lower-dimensional representation (latent space).
 - The **decoder** reconstructs the original input from the compressed data.

3. Load and Preprocess the Dataset:

- Choose a dataset for the autoencoder. Commonly used datasets for autoencoders are MNIST (for image reconstruction) or any dataset where dimensionality reduction is desired.
- For image datasets like MNIST, load and preprocess the data by normalizing pixel values between 0 and 1.

```
from keras.datasets import mnist
(X_train, _), (X_test, _) = mnist.load_data()
X_train = X_train.astype('float32') / 255.0
X_test = X_test.astype('float32') / 255.0
X_train = X_train.reshape((len(X_train), 28, 28, 1))
X_test = X_test.reshape((len(X_test), 28, 28, 1))
```

4. Build the Encoder:

- Define the encoder part of the autoencoder, which reduces the input dimensions to a smaller latent space.
- Use convolutional or dense layers based on the type of input (image or tabular data).

```

from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D

encoder = Sequential()
encoder.add(Conv2D(32, (3, 3), activation='relu', padding='same', input_shape=(28, 28, 1)))
encoder.add(MaxPooling2D((2, 2), padding='same'))
encoder.add(Conv2D(16, (3, 3), activation='relu', padding='same'))
encoder.add(MaxPooling2D((2, 2), padding='same'))

```

5. Build the Decoder:

- The decoder part reconstructs the original data from the compressed latent space representation.
- Use upsampling or dense layers, depending on the input type.

```

from keras.layers import UpSampling2D, Conv2DTranspose

decoder = Sequential()
decoder.add(Conv2D(16, (3, 3), activation='relu', padding='same',
input_shape=encoder.output_shape[1:]))
decoder.add(UpSampling2D((2, 2)))
decoder.add(Conv2D(32, (3, 3), activation='relu', padding='same'))
decoder.add(UpSampling2D((2, 2)))
decoder.add(Conv2D(1, (3, 3), activation='sigmoid', padding='same'))

```

6. Combine Encoder and Decoder:

- Create the full autoencoder model by combining the encoder and decoder.

```

from keras.models import Model
from keras.layers import Input

input_img = Input(shape=(28, 28, 1))
encoded = encoder(input_img)
decoded = decoder(encoded)
autoencoder = Model(input_img, decoded)

```

7. Compile the Model:

- Compile the autoencoder using an optimizer and loss function. The mean squared error (MSE) or binary crossentropy is typically used as the loss function, depending on the type of data.

```
autoencoder.compile(optimizer='adam', loss='binary_crossentropy')
```

8. Train the Autoencoder:

- Train the model using the training data. Since this is an unsupervised learning task, the model tries to reconstruct the input data.

```
autoencoder.fit(X_train, X_train, epochs=50, batch_size=256, shuffle=True, validation_data=(X_test, X_test))
```

9. Evaluate the Model:

- After training, evaluate the model on the test dataset to see how well it can reconstruct or compress the data.
- Visualize the original and reconstructed data to measure the effectiveness of the autoencoder.

```
decoded_imgs = autoencoder.predict(X_test)
```

Analyze the Latent Space:

- Optionally, analyze the compressed latent space representation (output of the encoder) for dimensionality reduction or clustering purposes.

PROGRAM:

```
import keras
```

```
from keras import layers
```

```
# This is the size of our encoded representations
```

```
encoding_dim = 32 # 32 floats -> compression of factor 24.5, assuming the input is 784 floats
```

```
# This is our input image
```

```
input_img = keras.Input(shape=(784,))
```

```
# "encoded" is the encoded representation of the input
```

```
encoded = layers.Dense(encoding_dim, activation='relu')(input_img) # "decoded"
```

is the lossy reconstruction of the input

```
decoded = layers.Dense(784, activation='sigmoid')(encoded)
```

```
# This model maps an input to its reconstruction
```

```
autoencoder = keras.Model(input_img, decoded)
```

```
# This model maps an input to its encoded representationencoder =
```

```
keras.Model(input_img, encoded)
```

```
# This is our encoded (32-dimensional) input
```

```
encoded_input = keras.Input(shape=(encoding_dim,)) #
```

```
Retrieve the last layer of the autoencoder modeldecoder_layer
```

```
= autoencoder.layers[-1]
```

```
# Create the decoder model
```

```
decoder = keras.Model(encoded_input, decoder_layer(encoded_input))
```

```
autoencoder.compile(optimizer='adam', loss='binary_crossentropy')from
```

```
keras.datasets import mnist
```

```
import numpy as np
```

```
(x_train, _), (x_test, _) = mnist.load_data()
```

```
x_train = x_train.astype('float32') / 255.x_test =
```

```
x_test.astype('float32') / 255.
```

```
x_train = x_train.reshape((len(x_train), np.prod(x_train.shape[1:])))x_test =
```

```
x_test.reshape((len(x_test), np.prod(x_test.shape[1:]))) print(x_train.shape)
```

```
print(x_test.shape)
```

```
autoencoder.fit(x_train, x_train,
```

```
epochs=50,
```

```
batch_size=256,
```

```
shuffle=True,
```

```
validation_data=(x_test, x_test))
```

```
# Encode and decode some digits
```

```
# Note that we take them from the *test* set
```

```

encoded_imgs = encoder.predict(x_test)
decoded_imgs = decoder.predict(encoded_imgs)

# Use Matplotlib (don't ask) import
matplotlib.pyplot as plt

n = 10 # How many digits we will display
plt.figure(figsize=(20, 4))
for i in range(n):
    # Display original
    ax = plt.subplot(2, n, i + 1)
    plt.imshow(x_test[i].reshape(28, 28))
    plt.gray() ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)

    # Display reconstruction
    ax = plt.subplot(2, n, i + 1 + n)
    plt.imshow(decoded_imgs[i].reshape(28, 28))
    plt.gray() ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False) plt.show()

from keras import regularizers

encoding_dim = 32

input_img = keras.Input(shape=(784,))
# Add a Dense layer with a L1 activity regularizer encoded =
layers.Dense(encoding_dim, activation='relu',
activity_regularizer=regularizers.l1(10e-5))(input_img)
decoded = layers.Dense(784, activation='sigmoid')(encoded)
autoencoder = keras.Model(input_img, decoded)

input_img = keras.Input(shape=(784,))

encoded = layers.Dense(128, activation='relu')(input_img) encoded =
layers.Dense(64, activation='relu')(encoded) encoded =

```

```
layers.Dense(32, activation='relu')(encoded)
```

```
decoded = layers.Dense(64, activation='relu')(encoded) decoded =
```

```
layers.Dense(128, activation='relu')(decoded) decoded =
```

```
layers.Dense(784, activation='sigmoid')(decoded)
```

```
autoencoder = keras.Model(input_img, decoded)
```

```
autoencoder.compile(optimizer='adam', loss='binary_crossentropy')
```

```
autoencoder.fit(x_train, x_train,
```

```
epochs=100,
```

```
batch_size=256,
```

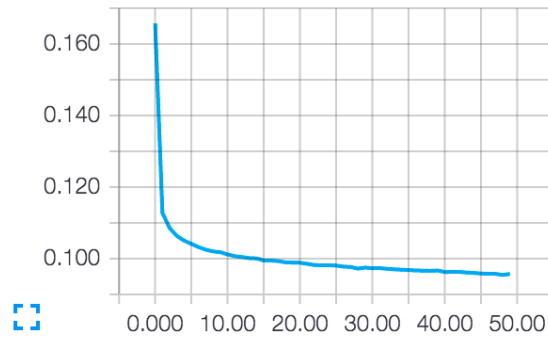
```
shuffle=True,
```

```
validation_data=(x_test, x_test))
```

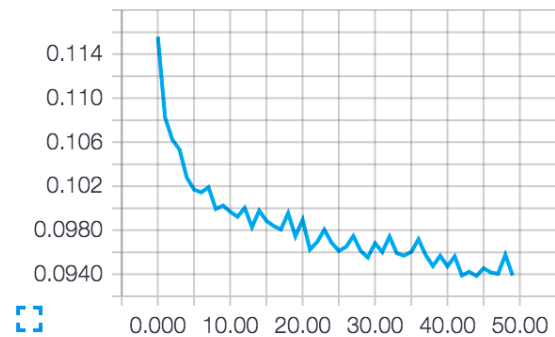
OUTPUT:



loss



val_loss



Result:

The autoencoder was successfully implemented for dimensionality reduction and data reconstruction. By learning a compressed representation of the input data, the autoencoder could effectively reconstruct the original input with minimal loss. This demonstrates the power of autoencoders in unsupervised learning, where they can capture underlying patterns in the data. The model can be further applied to tasks like noise reduction, anomaly detection, or feature extraction

Ex.no:8

Generative Adversarial Networks for image generation

Aim:

To implement a Generative Adversarial Network (GAN) for image generation, where the model learns to generate realistic images by training two neural networks: a generator and a discriminator, in an adversarial setting.

Procedure:

1. Install Required Libraries:

- Ensure the necessary libraries such as Keras, TensorFlow, or PyTorch are installed:
 - `pip install keras`
 - `pip install tensorflow`
- These will be used to create and train the GAN model.

2. GAN Architecture Overview:

- The GAN model consists of two neural networks:
 - **Generator:** Takes random noise as input and generates images.
 - **Discriminator:** Takes an image as input and classifies it as real or fake.
- Both networks are trained in an adversarial setup: the generator improves in creating realistic images, while the discriminator improves in detecting generated images.

3. Load and Preprocess Dataset:

- Use an image dataset like MNIST (for generating handwritten digits) or CIFAR-10 (for generating more complex images).
- Preprocess the dataset by normalizing the pixel values between -1 and 1 to match the generator's output.

```
from keras.datasets import mnist
(X_train, _), (_, _) = mnist.load_data()
X_train = X_train.astype('float32') / 127.5 - 1
X_train = np.expand_dims(X_train, axis=-1)
```

4. Build the Generator Network:

- The generator starts with a random noise vector and produces an image. Use dense and transposed convolution layers to upsample the noise into a meaningful image.

```
from keras.models import Sequential
from keras.layers import Dense, Reshape, Conv2DTranspose, BatchNormalization, ReLU
```

```
generator = Sequential()
generator.add(Dense(256, input_dim=100))
generator.add(Reshape((7, 7, 256)))
generator.add(Conv2DTranspose(128, (3, 3), strides=(2, 2), padding='same'))
generator.add(BatchNormalization())
generator.add(ReLU())
generator.add(Conv2DTranspose(1, (3, 3), strides=(2, 2), padding='same', activation='tanh'))
```

5. Build the Discriminator Network:

- The discriminator takes an image as input and outputs a probability (real or fake). Use convolutional layers to downsample the image and extract features.

```
from keras.layers import Conv2D, LeakyReLU, Flatten
```

```
discriminator = Sequential()
discriminator.add(Conv2D(64, (3, 3), padding='same', input_shape=(28, 28, 1)))
discriminator.add(LeakyReLU(alpha=0.2))
discriminator.add(Conv2D(128, (3, 3), strides=(2, 2)))
discriminator.add(LeakyReLU(alpha=0.2))
discriminator.add(Flatten())
discriminator.add(Dense(1, activation='sigmoid'))
```

6. Compile the Discriminator:

- Compile the discriminator using binary crossentropy loss, as it performs binary classification (real vs. fake).

```
discriminator.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

7. Build and Compile the GAN:

- The GAN combines the generator and discriminator. When training the generator, the discriminator's weights are frozen. The goal is for the generator to fool the discriminator.

```
from keras.models import Model
from keras.layers import Input

discriminator.trainable = False
gan_input = Input(shape=(100,))
generated_image = generator(gan_input)
gan_output = discriminator(generated_image)
gan = Model(gan_input, gan_output)
gan.compile(optimizer='adam', loss='binary_crossentropy')
```

8. Train the GAN:

- Train the GAN by alternating between training the discriminator and the generator:
 - First, train the discriminator on real and fake images.
 - Then, train the generator to create images that can fool the discriminator.

```
import numpy as np

for epoch in range(epochs):
    # Train discriminator on real and fake images
    real_images = X_train[np.random.randint(0, X_train.shape[0], batch_size)]
    noise = np.random.normal(0, 1, (batch_size, 100))
    fake_images = generator.predict(noise)

    real_labels = np.ones((batch_size, 1))
    fake_labels = np.zeros((batch_size, 1))

    discriminator.train_on_batch(real_images, real_labels)
    discriminator.train_on_batch(fake_images, fake_labels)

    # Train generator
    noise = np.random.normal(0, 1, (batch_size, 100))
    gan.train_on_batch(noise, real_labels)
```


9. Generate and Visualize Images:

- After training, generate new images from random noise and visualize them to evaluate how well the generator has learned to create realistic images.

program:

```
import torch

from torch import nn

import math
import matplotlib.pyplot as plt
import torchvision
import torchvision.transforms as transforms

torch.manual_seed(111)
device = ""
if torch.cuda.is_available():
    device = torch.device("cuda")
else:
    device = torch.device("cpu")

transform = transforms.Compose(
    [transforms.ToTensor(), transforms.Normalize((0.5,), (0.5,))]
)

train_set = torchvision.datasets.MNIST(
    root=".", train=True, download=True, transform=transform
)

batch_size = 32

train_loader = torch.utils.data.DataLoader(
    train_set, batch_size=batch_size, shuffle=True
)
```

```

real_samples, mnist_labels = next(iter(train_loader))for i in
range(16):
ax = plt.subplot(4, 4, i + 1) plt.imshow(real_samples[i].reshape(28, 28),
cmap="gray_r")plt.xticks([])
plt.yticks([])

```

```

class Discriminator(nn.Module):def__
init__(self):

```

```

super().__init__() self.model =
nn.Sequential(nn.Linear(784,
1024), nn.ReLU(),
nn.Dropout(0.3),
nn.Linear(1024, 512),
nn.ReLU(),
nn.Dropout(0.3),
nn.Linear(512, 256),
nn.ReLU(),
nn.Dropout(0.3),
nn.Linear(256, 1),
nn.Sigmoid(),
)

```

```

def forward(self, x):

```

```

x = x.view(x.size(0), 784)output
= self.model(x) return output
discriminator = Discriminator().to(device=device)class

```

```

Generator(nn.Module):

```

```

def __init__(self): super().__init__()
self.model = nn.Sequential(
nn.Linear(100, 256),

nn.ReLU(),

```

```

nn.Linear(256, 512),
nn.ReLU(),
nn.Linear(512, 1024),
nn.ReLU(),

nn.Linear(1024, 784),
nn.Tanh(),
)

def forward(self, x):

output = self.model(x)

output = output.view(x.size(0), 1, 28, 28)
return output

generator = Generator().to(device=device)
lr = 0.0001
num_epochs = 50
loss_function = nn.BCELoss()

optimizer_discriminator = torch.optim.Adam(discriminator.parameters(), lr=lr)
optimizer_generator = torch.optim.Adam(generator.parameters(), lr=lr)

for epoch in range(num_epochs):

for n, (real_samples, mnist_labels) in enumerate(train_loader):
    # Data for training the discriminator
    real_samples = real_samples.to(device=device)
    real_samples_labels = torch.ones((batch_size, 1)).to(device=device)
    )

latent_space_samples = torch.randn((batch_size, 100)).to(device=device)

```

```

)

generated_samples = generator(latent_space_samples)
generated_samples_labels = torch.zeros((batch_size, 1)).to(
device=device
)

all_samples = torch.cat((real_samples, generated_samples))
all_samples_labels = torch.cat(
(real_samples_labels, generated_samples_labels)

)

# Training the discriminator
discriminator.zero_grad()
output_discriminator = discriminator(all_samples)
loss_discriminator = loss_function( output_discriminator,
all_samples_labels
)

loss_discriminator.backward()
optimizer_discriminator.step()

# Data for training the generator

latent_space_samples = torch.randn((batch_size, 100)).to(
device=device
)

# Training the generator
generator.zero_grad()
generated_samples = generator(latent_space_samples)
output_discriminator_generated = discriminator(generated_samples)
loss_generator = loss_function(

```

```
output_discriminator_generated, real_samples_labels
```

```
)
```

```
loss_generator.backward()
```

```
optimizer_generator.step()#
```

```
Show loss
```

```
if n == batch_size - 1:
```

```
print(f'Epoch: {epoch} Loss D.: {loss_discriminator}')
```

```
print(f'Epoch: {epoch} Loss G.: {loss_generator}')
```

```
latent_space_samples = torch.randn(batch_size, 100).to(device=device)
```

```
generated_samples = generator(latent_space_samples)
```

```
generated_samples = generated_samples.cpu().detach()for i in
```

```
range(16):
```

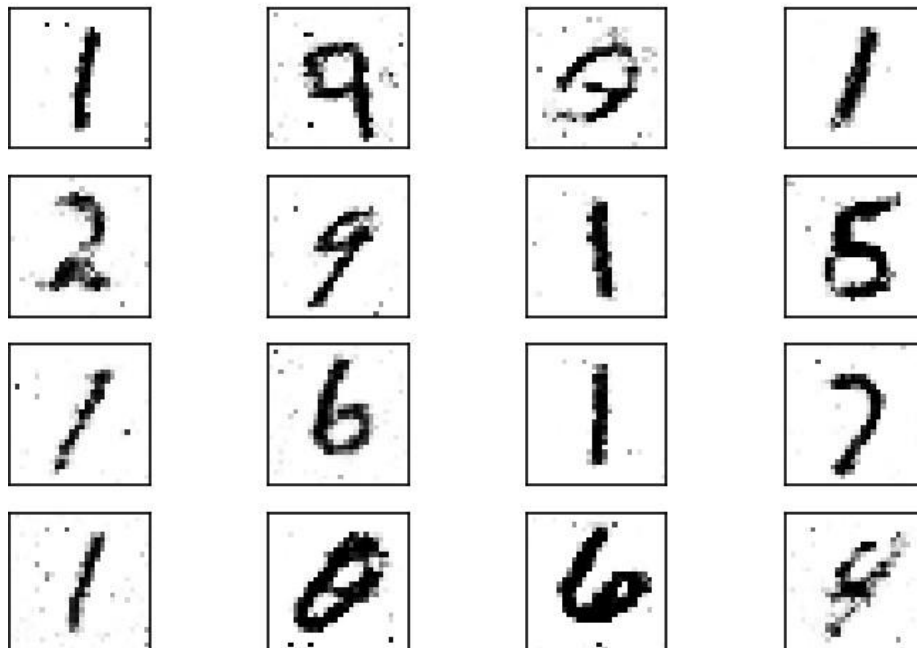
```
ax = plt.subplot(4, 4, i + 1) plt.imshow(generated_samples[i].reshape(28, 28),
```

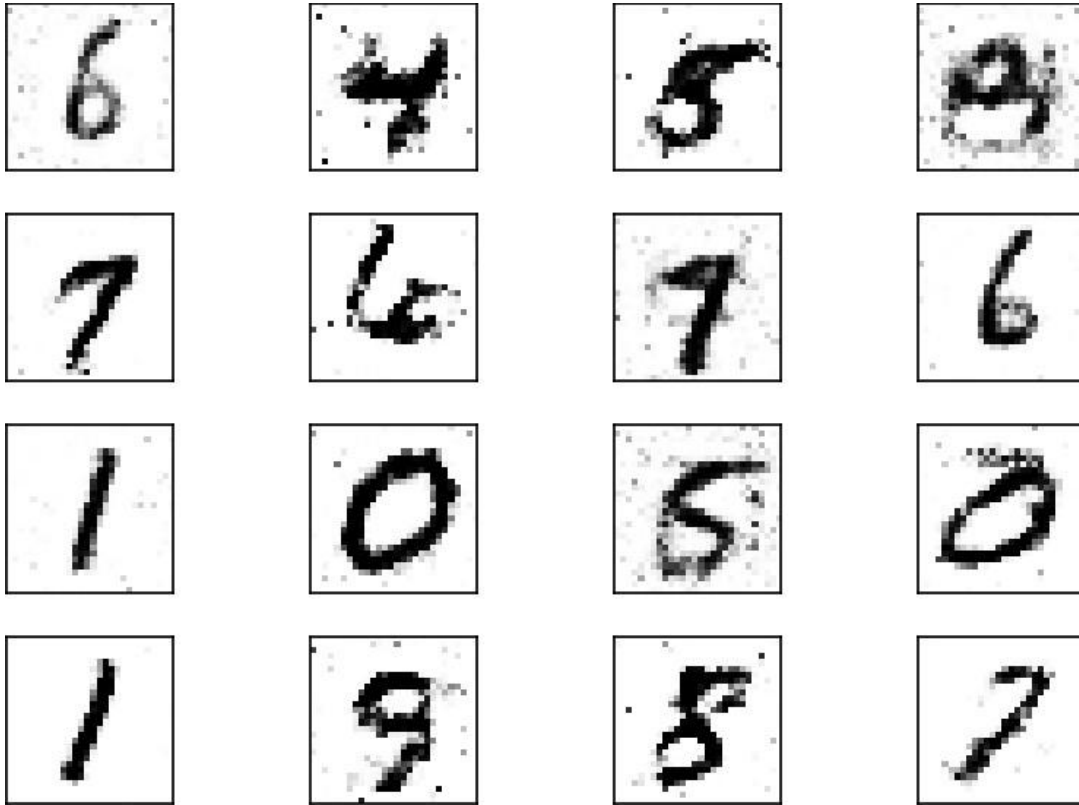
```
cmap="gray_r")plt.xticks([])
```

```
plt.yticks([])
```

OUTPUT:

After 46 epoch(s)



**Result:**

The Generative Adversarial Network (GAN) was successfully implemented for image generation. The generator learned to create realistic images through adversarial training, where it competes with the discriminator. This experiment demonstrates the effectiveness of GANs in generating synthetic images that resemble real data, showcasing the potential of GANs in applications such as image synthesis, style transfer, and data augmentation.