6.Write a program for Time-Series Forecasting with the LSTM Model.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
from sklearn.preprocessing import MinMaxScaler
# Load the dataset (assuming NFLX.csv is uploaded)
df = pd.read_csv("/content/NFLX.csv")
# Ensure Date is in datetime format
df['Date'] = pd.to_datetime(df['Date'])
df = df.sort_values('Date')  # Sort by date
# Extract closing prices
data = df['Close'].values.reshape(-1, 1)
# Scale data
scaler = MinMaxScaler()
data_scaled = scaler.fit_transform(data)
# Prepare dataset (10 days input -> next day output)
X, Y = [], []
sequence_length = 10
for i in range(len(data_scaled) - sequence_length):
    X.append(data_scaled[i:i + sequence_length])
    Y.append(data_scaled[i + sequence_length])
X, Y = np.array(X), np.array(Y)
# Split data (80% train, 20% test)
split = int(len(X) * 0.8)
X_train, X_test = X[:split], X[split:]
Y_train, Y_test = Y[:split], Y[split:]
# Reshape for LSTM (samples, time steps, features)
X_train = X_train.reshape((X_train.shape[0], X_train.shape[1], 1))
X_test = X_test.reshape((X_test.shape[0], X_test.shape[1], 1))
# Build LSTM model
model = Sequential([
    LSTM(50, activation='relu', input_shape=(sequence_length, 1)),
    Dense(1)
])
model.compile(optimizer='adam', loss='mse')
# Train the model
model.fit(X_train, Y_train, epochs=20, batch_size=16, verbose=1)
# Make predictions
Y_pred = model.predict(X_test)
# Inverse transform predictions and actual values
Y_pred_actual = scaler.inverse_transform(Y_pred)
Y_test_actual = scaler.inverse_transform(Y_test.reshape(-1, 1))
# Plot actual vs. predicted closing prices
plt.figure(figsize=(10, 5))
plt.plot(Y_test_actual, label="Actual Closing Price", color='blue')
plt.plot(Y_pred_actual, label="Predicted Closing Price", color='red')
plt.legend()
plt.xlabel("Days")
plt.ylabel("Price")
plt.title("Netflix Stock Price Prediction (LSTM)")
plt.show()
```

7.Build a feed forward neural network for prediction of logic gates.

```
import pandas as pd
import numpy as np
import tensorflow as tf
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
# Load dataset
file_path = "/content/logical_gate.csv"  # Ensure correct file path
df = pd.read_csv(file_path)
# Convert text values to numbers
text_to_num = {"zero": 0, "one": 1}
for col in df.columns[2:]:
   df[col] = df[col].map(text_to_num)
# Drop rows with missing values
df = df.dropna().astype(int)
# Prepare input and output data
X = df[['Value_A', 'Value_B']].values  # Input features
Y = df.iloc[:, 2:].values  # Target (logic gate outputs)
# Split data into train and test sets
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=42)
# Define the neural network model
model = tf.keras.Sequential([
   tf.keras.layers.Dense(8, activation='relu', input_shape=(2,)),
   tf.keras.layers.Dense(8, activation='relu'),
   tf.keras.layers.Dense(Y.shape[1], activation='sigmoid')
])
# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
# Train the model
model.fit(X_train, Y_train, epochs=100, batch_size=4, verbose=1)
# Evaluate the model
loss, accuracy = model.evaluate(X_test, Y_test)
print(f"\nModel Accuracy: {accuracy:.4f}")
scaler=StandardScaler()
X = scaler.fit_transform(X)
# Function to predict logic gate outputs
def predict_logic_gate():
   while True:
      try:
         a = int(input("\nEnter Value_A (0 or 1): "))
         b = int(input("Enter Value_B (0 or 1): "))
         if a not in [0, 1] or b not in [0, 1]:
            print("Invalid input! Please enter 0 or 1 only.")
```

```
                continue
        except ValueError:
            print("Invalid input! Enter numbers 0 or 1 only.")
            continue
        # Predict output
        user_input = scaler.transform(np.array([[a, b]], dtype=np.float32))
        prediction = model.predict(user_input)
        output = (prediction > 0.5).astype(int)[0]
        # Print results
        gate_names = df.columns[2:]
        print("\nPredicted Logic Gate Outputs:")
        for gate, result in zip(gate_names, output):
            print(f"{gate}: {result}")
        # Ask if the user wants to continue
        cont = input("\nDo you want to test another input? (yes/no): ").strip().lower()
        if cont != "yes":
            print("Exiting...")
            break
# Run the prediction function
predict_logic_gate()
```

8. Write a program to implement deep learning Techniques for image segmentation.

```
import tensorflow as tf
from tensorflow.keras import layers, Model
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.preprocessing.image import load_img, img_to_array
# Define U-Net model
def unet_model(input_size=(128, 128, 3)):
    inputs = layers.Input(input_size)
    # Encoding path
    c1 = layers.Conv2D(64, (3, 3), activation='relu', padding='same')(inputs)
    c1 = layers.Conv2D(64, (3, 3), activation='relu', padding='same')(c1)
    p1 = layers.MaxPooling2D((2, 2))(c1)
    c2 = layers.Conv2D(128, (3, 3), activation='relu', padding='same')(p1)
    c2 = layers.Conv2D(128, (3, 3), activation='relu', padding='same')(c2)
    p2 = layers.MaxPooling2D((2, 2))(c2)
    # Bottleneck
    c3 = layers.Conv2D(256, (3, 3), activation='relu', padding='same')(p2)
    c3 = layers.Conv2D(256, (3, 3), activation='relu', padding='same')(c3)
    # Decoding path
    u1 = layers.UpSampling2D((2, 2))(c3)
    u1 = layers.Conv2D(128, (3, 3), activation='relu', padding='same')(u1)
    m1 = layers.concatenate([u1, c2])
    u2 = layers.UpSampling2D((2, 2))(m1)
    u2 = layers.Conv2D(64, (3, 3), activation='relu', padding='same')(u2)
    m2 = layers.concatenate([u2, c1])
    outputs = layers.Conv2D(1, (1, 1), activation='sigmoid')(m2)
    model = Model(inputs, outputs)
    return model
# Compile model
model = unet_model()
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
model.summary()
# Load and preprocess example image
def preprocess_image(image_path, target_size=(128, 128)):
    image = load_img(image_path, target_size=target_size)
    image = img_to_array(image) / 255.0  # Normalize
    return np.expand_dims(image, axis=0)  # Add batch dimension
# Example usage (Replace 'image.jpg' with actual image path)
image = preprocess_image('/content/cat.jpg')
predicted_mask = model.predict(image)
plt.imshow(predicted_mask[0, :, :, 0], cmap='gray')
plt.show()
```

9. Write a program for object detection using image labeling tools.

```
import torch
import cv2
import numpy as np
from transformers import DetrImageProcessor, DetrForObjectDetection
from PIL import Image
import matplotlib.pyplot as plt
import matplotlib.patches as patches
# Load pre-trained DETR model from Hugging Face
processor = DetrImageProcessor.from_pretrained("facebook/detr-resnet-50")
model = DetrForObjectDetection.from_pretrained("facebook/detr-resnet-50")
# Load an image from Google Drive
image_path = "/content/cat.jpg"
image = Image.open(image_path)
# Convert image to model input format
inputs = processor(images=image, return_tensors="pt")
# Perform object detection
with torch.no_grad():
    outputs = model(**inputs)
# Get bounding boxes, labels, and scores
target_sizes = torch.tensor([image.size[::-1]])
results = processor.post_process_object_detection(outputs, target_sizes=target_sizes,
threshold=0.5)[0]

# Visualize results using Matplotlib
fig, ax = plt.subplots(1, figsize=(10, 6))
ax.imshow(image)
for score, label, box in zip(results["scores"], results["labels"], results["boxes"]):
    box = [int(i) for i in box]  # Convert to integers
    rect = patches.Rectangle((box[0], box[1]), box[2] - box[0], box[3] - box[1],
                             linewidth=2, edgecolor='r', facecolor='none')
    ax.add_patch(rect)
    ax.text(box[0], box[1] - 10, f"{model.config.id2label[label.item()]}: {score:.2f}",
            color='red', fontsize=10, bbox=dict(facecolor='white', alpha=0.5))
plt.show()
```

10. Write a program to predict a caption for a sample image using LSTM.

```
from transformers import BlipProcessor, BlipForConditionalGeneration
from PIL import Image
# Load the pre-trained BLIP model
processor = BlipProcessor.from_pretrained("Salesforce/blip-image-captioning-base")
model = BlipForConditionalGeneration.from_pretrained("Salesforce/blip-image-captioning-base")
# Load an image (Change path to your image)
image_path = "/content/img five.png"
image = Image.open(image_path).convert("RGB")
# Generate caption
inputs = processor(images=image, return_tensors="pt")
caption_ids = model.generate(**inputs)
caption = processor.decode(caption_ids[0], skip_special_tokens=True)
print("Predicted Caption:", caption)
```

11. Write a program for character recognition using CNN.

```
import tensorflow as tf
from tensorflow.keras.models import Sequential, load_model
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
from tensorflow.keras.datasets import mnist
from tensorflow.keras.utils import to_categorical
import matplotlib.pyplot as plt
import numpy as np
# Load and preprocess data
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0
x_train = x_train.reshape(-1, 28, 28, 1)
x_test = x_test.reshape(-1, 28, 28, 1)
y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)
# Build CNN model
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(128, activation='relu'),
    Dense(10, activation='softmax')
])
# Compile and train model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
model.fit(x_train, y_train, epochs=3, batch_size=32, validation_data=(x_test, y_test))
# Evaluate model and predict
test_acc = model.evaluate(x_test, y_test)[1]
print(f"Test Accuracy: {test_acc:.4f}")
# Function to predict a given character
def predict_character(image):
    image = image / 255.0  # Normalize
    image = image.reshape(1, 28, 28, 1)  # Reshape for model input
    prediction = model.predict(image)
    return np.argmax(prediction)
# Test with user-provided image
import cv2
image_path = input("enter path of image")
image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
image = cv2.resize(image, (28, 28))
predicted_label = predict_character(image)
print(f"Predicted Character: {predicted_label}")
```

12. Write a program to predict a caption for a sample image using CNN.

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.applications.inception_v3 import InceptionV3, decode_predictions,
preprocess_input
from tensorflow.keras.preprocessing.image import load_img, img_to_array
# Load CNN model
model = InceptionV3(weights="imagenet")
# Function to generate a single-word caption
def generate_single_caption(image_path):
    # Load and preprocess image
    image = load_img(image_path, target_size=(299, 299))
    image = img_to_array(image)
    image = np.expand_dims(image, axis=0)
    image = preprocess_input(image)

    # Predict objects in the image
    predictions = model.predict(image)
    label = decode_predictions(predictions, top=1)[0][0][1]  # Get the top predicted label
    return label.replace("_", " ")  # Replace underscores if any
# Example usage
print("Generated Caption:", generate_single_caption("/content/tiger.jpg"))
```