**(1all 1-9)**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*HOUSE PRICE PREDICTION(LINEAR REGRESSION HYPERTUNUNG)\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**import pandas as pd**

**import matplotlib.pyplot as plt**

**import numpy as np**

**from sklearn.model\_selection import train\_test\_split**

**from sklearn.preprocessing import StandardScaler, OneHotEncoder**

**from sklearn.compose import ColumnTransformer**

**from tensorflow.keras.models import Sequential**

**from tensorflow.keras.layers import Dense**

**from tensorflow.keras.optimizers import Adam**

**# Load your dataset (replace 'your\_data.csv' with your actual dataset)**

**# Make sure your dataset includes features and target variable (house prices)**

**data = pd.read\_csv('/content/Housing.csv')**

**# Assuming 'date' is the name of the column with date values**

**data['date'] = pd.to\_datetime(data['date'])**

**# Extract features from the date column**

**data['year'] = data['date'].dt.year**

**data['month'] = data['date'].dt.month**

**data['day'] = data['date'].dt.day**

**# Drop the original date column**

**data = data.drop(['date'], axis=1)**

**# Separate features and target variable**

**X = data.drop('price', axis=1)**

**y = data['price']**

**# Identify categorical columns**

**categorical\_features = X.select\_dtypes(include=['object']).columns**

**# Create a preprocessor using ColumnTransformer**

**preprocessor = ColumnTransformer(**

**transformers=[**

**('num', StandardScaler(), X.select\_dtypes(include=['number']).columns),**

**('cat', OneHotEncoder(), categorical\_features)**

**],**

**remainder='passthrough'**

**)**

**# Transform the data**

**X\_scaled = preprocessor.fit\_transform(X)**

**# Split the data into training and testing sets**

**X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_scaled, y, test\_size=0.2, random\_state=42)**

**# Build the MLP model**

**model = Sequential()**

**# Add input layer and first hidden layer**

**model.add(Dense(units=64, activation='relu', input\_dim=X\_scaled.shape[1]))**

**# Add additional hidden layers**

**model.add(Dense(units=32, activation='relu'))**

**model.add(Dense(units=16, activation='relu'))**

**# Add output layer (1 unit for regression, no activation function)**

**model.add(Dense(units=1))**

**# Compile the model**

**model.compile(optimizer=Adam(learning\_rate=0.001), loss='mean\_squared\_error')**

**# Train the model**

**history = model.fit(X\_train, y\_train, epochs=50, batch\_size=32, validation\_split=0.2, verbose=0)**

**plt.figure(figsize=(5,3))**

**plt.plot(history.history['loss'], label='Training Loss')**

**plt.plot(history.history['val\_loss'], label='Validation Loss')**

**plt.title('Training and Validation Loss')**

**plt.xlabel('Epochs')**

**plt.ylabel('Mean Squared Error Loss')**

**plt.legend()**

**plt.show()**

**# Evaluate the model on the test set**

**loss = model.evaluate(X\_test, y\_test)**

**print(f'Mean Squared Error on Test Set: {loss}')**

**# Make predictions**

**predictions = model.predict(X\_test)**

**# Plot actual vs predicted values with the best-fit regression line**

**plt.figure(figsize=(5,3))**

**plt.scatter(y\_test, predictions, label='Actual vs Predicted')**

**plt.xlabel('Actual Prices')**

**plt.ylabel('Predicted Prices')**

**# Fit a linear regression line**

**regression\_line = np.polyfit(y\_test, predictions.flatten(), 1)**

**plt.plot(y\_test, np.polyval(regression\_line, y\_test), color='red', label='Regression Line')**

**plt.title('Actual Prices vs Predicted Prices with Regression Line')**

**plt.legend()**

**plt.show()**

**#OUTPUT:**

**136/136 [==============================] - 0s 2ms/step - loss: 33518297088.0000**

**Mean Squared Error on Test Set: 33518297088.0**

**136/136 [==============================] - 0s 1ms/step**

**#TRAINING AND VALIDATION LOSS GRAPH**

**#ACTUAL PRICES AND PREDICTED PRICES WITH REGRESSION**

**\*\*\*\*\*\*\*IMPLEMENT KERAS WITH TENSOR FLOW WITH CLASSIFICATION PROBLEM(HEART DISEASE)\*\*\***

**import numpy as np**

**import pandas as pd**

**import matplotlib.pyplot as plt**

**from sklearn.model\_selection import train\_test\_split**

**from sklearn.preprocessing import StandardScaler**

**import tensorflow as tf**

**from tensorflow.keras.models import Sequential**

**from tensorflow.keras.layers import Dense**

**from sklearn.metrics import confusion\_matrix, accuracy\_score, precision\_score, recall\_score, f1\_score**

**import seaborn as sns**

**# Load the heart disease dataset (replace with the path to your downloaded dataset)**

**# For example, if you upload the dataset to Colab, you can use the following:**

**# from google.colab import files**

**# uploaded = files.upload()**

**# df = pd.read\_csv(io.BytesIO(uploaded['heart.csv']))**

**# Replace 'heart.csv' with the actual name of your dataset file**

**df = pd.read\_csv('heart.csv')**

**# Assuming the dataset has columns 'target' as labels and other columns as features**

**X = df.drop('target', axis=1)**

**y = df['target']**

**# Split the data into training and testing sets**

**X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.1, random\_state=42)**

**# Standardize the numerical features**

**scaler = StandardScaler()**

**X\_train\_scaled = scaler.fit\_transform(X\_train)**

**X\_test\_scaled = scaler.transform(X\_test)**

**# Convert labels to categorical format**

**y\_train\_categorical = tf.keras.utils.to\_categorical(y\_train)**

**y\_test\_categorical = tf.keras.utils.to\_categorical(y\_test)**

**# Build the MLP model**

**model = Sequential()**

**model.add(Dense(64, activation='relu', input\_shape=(X\_train\_scaled.shape[1],)))**

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

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

**model.add(Dense(64, activation='relu'))**

**model.add(Dense(2, activation='softmax')) # Assuming binary classification**

**# Compile the model**

**model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])**

**# Train the model**

**history = model.fit(X\_train\_scaled, y\_train\_categorical, epochs=50, batch\_size=32, validation\_split=0.1)**

**# Evaluate the model on the test set**

**model.evaluate(X\_test\_scaled, y\_test\_categorical)**

**# Assuming 'model' is your trained MLP model**

**# Make predictions on the test set**

**y\_pred\_probs = model.predict(X\_test\_scaled)**

**y\_pred = np.argmax(y\_pred\_probs, axis=1)**

**# Convert one-hot encoded labels back to integers (if needed)**

**y\_test\_int = np.argmax(y\_test\_categorical, axis=1)**

**# Calculate confusion matrix**

**cm = confusion\_matrix(y\_test\_int, y\_pred)**

**# Calculate accuracy**

**accuracy = accuracy\_score(y\_test\_int, y\_pred)**

**# Calculate precision**

**precision = precision\_score(y\_test\_int, y\_pred)**

**# Calculate recall**

**recall = recall\_score(y\_test\_int, y\_pred)**

**# Calculate F1 score**

**f1 = f1\_score(y\_test\_int, y\_pred)**

**# Display the results**

**print("Confusion Matrix:")**

**print(cm)**

**print("\nAccuracy:", accuracy)**

**print("Precision:", precision)**

**print("Recall:", recall)**

**print("F1 Score:", f1)**

**# Plot the confusion matrix**

**plt.figure(figsize=(4, 4))**

**sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False,**

**annot\_kws={'size': 5}, linewidths=0.5, linecolor='black')**

**plt.title('Confusion Matrix')**

**plt.xlabel('Predicted')**

**plt.ylabel('Actual')**

**plt.show()**

**# Plot training history**

**plt.plot(history.history['accuracy'])**

**plt.plot(history.history['val\_accuracy'])**

**plt.title('Model Accuracy')**

**plt.xlabel('Epoch')**

**plt.ylabel('Accuracy')**

**plt.legend(['Training', 'Validation'], loc='upper left')**

**plt.show()**

**plt.plot(history.history['loss'])**

**plt.plot(history.history['val\_loss'])**

**plt.title('Model Loss')**

**plt.xlabel('Epoch')**

**plt.ylabel('Loss')**

**plt.legend(['Training', 'Validation'], loc='upper right')**

**plt.show()**

**\*\*\*\*\*\*\*\*\*\*\*\*\* TO IMPLEMENT A CNN FOR DOG/CAT CLASSIFICATION PROBLEM \*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**import pandas as pd**

**import numpy as np**

**import matplotlib.pyplot as plt**

**import warnings**

**import os**

**import tqdm**

**import random**

**from keras.preprocessing.image import load\_img**

**warnings.filterwarnings('ignore')**

**input\_path = []**

**label = []**

**for class\_name in os.listdir("PetImages"):**

**for path in os.listdir("PetImages/"+class\_name):**

**if class\_name == 'Cat':**

**label.append(0)**

**else:**

**label.append(1)**

**input\_path.append(os.path.join("PetImages", class\_name, path))**

**print(input\_path[0], label[0])**

**df = pd.DataFrame()**

**df['images'] = input\_path**

**df['label'] = label**

**df = df.sample(frac=1).reset\_index(drop=True)**

**df.head()**

**for i in df['images']:**

**if '.jpg' not in i:**

**print(i)**

**import PIL**

**l = []**

**for image in df['images']:**

**try:**

**img = PIL.Image.open(image)**

**except:**

**l.append(image)**

**l**

**# delete db files**

**df = df[df['images']!='PetImages/Dog/Thumbs.db']**

**df = df[df['images']!='PetImages/Cat/Thumbs.db']**

**df = df[df['images']!='PetImages/Cat/666.jpg']**

**df = df[df['images']!='PetImages/Dog/11702.jpg']**

**len(df)**

**# to display grid of images**

**plt.figure(figsize=(25,25))**

**temp = df[df['label']==1]['images']**

**start = random.randint(0, len(temp))**

**files = temp[start:start+25]**

**for index, file in enumerate(files):**

**plt.subplot(5,5, index+1)**

**img = load\_img(file)**

**img = np.array(img)**

**plt.imshow(img)**

**plt.title('Dogs')**

**plt.axis('off')**

**####DOG IMAGES OF 25**

**# to display grid of images**

**plt.figure(figsize=(25,25))**

**temp = df[df['label']==0]['images']**

**start = random.randint(0, len(temp))**

**files = temp[start:start+25]**

**for index, file in enumerate(files):**

**plt.subplot(5,5, index+1)**

**img = load\_img(file)**

**img = np.array(img)**

**plt.imshow(img)**

**plt.title('Cats')**

**plt.axis('off')**

**#### CAT IMAGES OF 25**

**import seaborn as sns**

**sns.countplot(df['label'])**

**### GRAPH BETWEEN LABEL AND COUNT**

**df['label'] = df['label'].astype('str')**

**df.head()**

**from sklearn.model\_selection import train\_test\_split**

**train, test = train\_test\_split(df, test\_size=0.2, random\_state=42)**

**from keras.preprocessing.image import ImageDataGenerator**

**train\_generator = ImageDataGenerator(**

**rescale = 1./255, # normalization of images**

**rotation\_range = 40, # augmention of images to avoid overfitting**

**shear\_range = 0.2,**

**zoom\_range = 0.2,**

**horizontal\_flip = True,**

**fill\_mode = 'nearest'**

**)**

**val\_generator = ImageDataGenerator(rescale = 1./255)**

**train\_iterator = train\_generator.flow\_from\_dataframe(**

**train,**

**x\_col='images',**

**y\_col='label',**

**target\_size=(128,128),**

**batch\_size=512,**

**class\_mode='binary'**

**)**

**val\_iterator = val\_generator.flow\_from\_dataframe(**

**test,**

**x\_col='images',**

**y\_col='label',**

**target\_size=(128,128),**

**batch\_size=512,**

**class\_mode='binary'**

**)**

**from keras import Sequential**

**from keras.layers import Conv2D, MaxPool2D, Flatten, Dense**

**model = Sequential([**

**Conv2D(16, (3,3), activation='relu', input\_shape=(128,128,3)),**

**MaxPool2D((2,2)),**

**Conv2D(32, (3,3), activation='relu'),**

**MaxPool2D((2,2)),**

**Conv2D(64, (3,3), activation='relu'),**

**MaxPool2D((2,2)),**

**Flatten(),**

**Dense(512, activation='relu'),**

**Dense(1, activation='sigmoid')**

**model.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])**

**model.summary()**

**history = model.fit(train\_iterator, epochs=10, validation\_data=val\_iterator)**

**acc = history.history['accuracy']**

**val\_acc = history.history['val\_accuracy']**

**epochs = range(len(acc))**

**plt.plot(epochs, acc, 'b', label='Training Accuracy')**

**plt.plot(epochs, val\_acc, 'r', label='Validation Accuracy')**

**plt.title('Accuracy Graph')**

**plt.legend()**

**plt.figure()**

**loss = history.history['loss']**

**val\_loss = history.history['val\_loss']**

**plt.plot(epochs, loss, 'b', label='Training Loss')**

**plt.plot(epochs, val\_loss, 'r', label='Validation Loss')**

**plt.title('Loss Graph')**

**plt.legend()**

**plt.show()**

**image\_path = "/content/kitten-2354016\_1280.jpg" # path of the image**

**img = load\_img(image\_path, target\_size=(128, 128))**

**img = np.array(img)**

**img = img / 255.0 # normalize the image**

**img = img.reshape(1, 128, 128, 3) # reshape for prediction**

**pred = model.predict(img)**

**if pred[0] > 0.5:**

**label = 'Dog'**

**else:**

**label = 'Cat'**

**print(label)**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*5.CNN FOR OBJECT DETECTION IN GIVEN IMAGE\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**!pip install ultralytics -q**

**!pip install pyyaml -q**

**# ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 259.3/259.3 KB 6.7 MB/s eta 0:00:00 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 178.9/178.9 KB 18.0 MB/s eta 0:00:00 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.6/1.6 MB 44.8 MB/s eta 0:00:00**

**━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 140.6/140.6 KB 14.1 MB/s eta 0:00:00**

**from ultralytics import YOLO**

**import yaml**

**import cv2**

**from google.colab.patches import cv2\_imshow**

**model = YOLO("yolov8n.pt")**

**#Downloading https://github.com/ultralytics/assets/releases/download/v0.0.0/yolov8n.pt to yolov8n.pt...**

**0%| | 0.00/6.23M [00:00<?, ?B/s]**

**model.predict("/content/car\_and\_dog.jpg" , save = True , save\_txt = True)**

**#**

**Ultralytics YOLOv8.0.18 🚀 Python-3.8.10 torch-1.13.1+cu116 CPU**

**YOLOv8n summary (fused): 168 layers, 3151904 parameters, 0 gradients, 8.7 GFLOPs**

**Results saved to runs/detect/predict**

**1 label saved to runs/detect/predict/labels**

**[Ultralytics YOLO <class 'ultralytics.yolo.engine.results.Boxes'> masks**

**type: <class 'torch.Tensor'>**

**shape: torch.Size([2, 6])**

**dtype: torch.float32**

**+ tensor([[ 0.00000, 161.00000, 491.00000, 432.00000, 0.75548, 2.00000],**

**[134.00000, 66.00000, 220.00000, 178.00000, 0.73854, 16.00000]])]**

**file\_name = "../usr/local/lib/python3.8/dist-packages/ultralytics/yolo/data/datasets/coco8.yaml"**

**with open(file\_name , "r") as stream:**

**names = yaml.safe\_load(stream)["names"]**

**names**

**lis = open("/content/runs/detect/predict/labels/car\_and\_dog.txt" , "r").readlines()**

**lis**

**#'16 0.345703 0.238281 0.167969 0.21875\n',**

**'2 0.479492 0.579102 0.958984 0.529297\n']**

**for l in lis:**

**ind = int(l.split()[0])**

**print(ind , names[ind])**

**#16 dog**

**2 car**

**float("0.21875\n")**

**#0.21875**

**li = lis[0].split()**

**xc , yc , nw , nh = float(li[1]) , float(li[2]) , float(li[3]) , float(li[4])**

**img = cv2.imread("/content/car\_and\_dog.jpg")**

**h , w = img.shape[0] , img.shape[1]**

**xc \*= w**

**yc \*= h**

**nw \*= w**

**nh \*= h**

**top\_left = (int(xc - nw/2) , int(yc - nh/2))**

**bottom\_right = (int(xc + nw/2) , int(yc + nh/2))**

**top\_left , bottom\_right**

**#((133, 65), (220, 177))**

**img = cv2.rectangle(img , top\_left , bottom\_right , (0 , 255 , 0) , 3)**

**cv2\_imshow(img)**

**#DOG IMAGE ON CAR**

**model.predict("/content/doggo.jpg" , save = True , save\_txt = True)**

**#Results saved to runs/detect/predict**

**2 labels saved to runs/detect/predict/labels**

**[Ultralytics YOLO <class 'ultralytics.yolo.engine.results.Boxes'> masks**

**type: <class 'torch.Tensor'>**

**shape: torch.Size([4, 6])**

**dtype: torch.float32**

**+ tensor([[1.31000e+02, 2.20000e+02, 3.09000e+02, 5.42000e+02, 9.08002e-01, 1.60000e+01],**

**[1.31000e+02, 1.40000e+02, 5.68000e+02, 4.21000e+02, 8.88764e-01, 1.00000e+00],**

**[4.67000e+02, 7.50000e+01, 6.92000e+02, 1.72000e+02, 5.30585e-01, 2.00000e+00],**

**[4.67000e+02, 7.50000e+01, 6.93000e+02, 1.72000e+02, 5.08616e-01, 7.00000e+00]])]**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* TO IMPLEMENT A RNN FOR PREDICTING THE SERIES DATA\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**import numpy as np**

**from keras.models import Sequential**

**from keras.layers import Dense**

**from keras.layers import Dense,LSTM,SimpleRNN**

**from sklearn.model\_selection import train\_test\_split**

**import matplotlib.pyplot as plt**

**data = [[[(i+j)/100] for i in range(5)] for j in range(100)]**

**data = np.array(data, dtype=np.float32)**

**target = [[(i+5)/100] for i in range(100)]**

**target = np.array(target, dtype=np.float32)**

**data[2]**

**#array([[0.02],**

**[0.03],**

**[0.04],**

**[0.05],**

**[0.06]], dtype=float32)**

**target[2]**

**#array([0.07], dtype=float32)**

**data.shape**

**#(100, 5, 1)**

**target.shape**

**#(100, 1)**

**x\_train,x\_test,y\_train,y\_test=train\_test\_split(data,target,test\_size=0.2,random\_state=4)**

**model = Sequential()**

**model.add(LSTM((20), input\_shape=(5,1),return\_sequences=True,activation="sigmoid"))**

**model.add(LSTM((1),return\_sequences=False,activation="sigmoid"))**

**#model.add(Dense(1))**

**model.compile(loss='mean\_absolute\_error', optimizer='adam',metrics=['accuracy'])**

**model.summary()**

**#Model: "sequential"**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Layer (type) Output Shape Param #**

**=================================================================**

**lstm (LSTM) (None, 5, 20) 1760**

**lstm\_1 (LSTM) (None, 1) 88**

**=================================================================**

**Total params: 1848 (7.22 KB)**

**Trainable params: 1848 (7.22 KB)**

**Non-trainable params: 0 (0.00 Byte)**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**history=model.fit(data, target, epochs=100, batch\_size=1, verbose=2,validation\_data=(x\_test, y\_test))**

**#2427 - accuracy: 0.0100 - val\_loss: 0.2247 - val\_accuracy: 0.0500 - 438ms/epoch - 4ms/step**

**Epoch 4/100**

**100/100 - 0s - loss: 0.2386 - accuracy: 0.0100 - val\_loss: 0.2179 - val\_accuracy: 0.0500 - 425ms/epoch - 4ms/step**

**Epoch 5/100**

**100/100 - 0s - loss: 0.2322 - accuracy: 0.0100 - val\_loss: 0.2215 - val\_accuracy: 0.0500 - 430ms/epoch - 4ms/step**

**Epoch 6/100**

**100/100 - 0s - loss: 0.2262 - accuracy: 0.0100 - val\_loss: 0.2175 - val\_accuracy: 0.0500 - 425ms/epoch - 4ms/step**

**Epoch 7/100**

**100/100 - 0s - loss: 0.2207 - accuracy: 0.0100 - val\_loss: 0.2115 - val\_accuracy: 0.0500 - 437ms/epoch - 4ms/step**

**Epoch 8/100**

**100/100 - 0s - loss: 0.2088 - accuracy: 0.0100 - val\_loss: 0.2079 - val\_accuracy: 0.0500 - 433ms/epoch - 4ms/step**

**Epoch 9/100**

**100/100 - 0s - loss: 0.2004 - accuracy: 0.0100 - val\_loss: 0.1839 - val\_accuracy: 0.0500 - 402ms/epoch - 4ms/step**

**Epoch 10/100**

**100/100 - 0s - loss: 0.1839 - accuracy: 0.0100 - val\_loss: 0.1946 - val\_accuracy: 0.0500 - 405ms/epoch - 4ms/step**

**Epoch 11/100**

**100/100 - 0s - loss: 0.1634 - accuracy: 0.0100 - val\_loss: 0.1507 - val\_accuracy: 0.0500 - 433ms/epoch - 4ms/step**

**Epoch 12/100**

**100/100 - 0s - loss: 0.1358 - accuracy: 0.0100 - val\_loss: 0.1305 - val\_accuracy: 0.0500 - 423ms/epoch - 4ms/step**

**Epoch 13/100**

**...**

**Epoch 99/100**

**100/100 - 0s - loss: 0.0303 - accuracy: 0.0100 - val\_loss: 0.0318 - val\_accuracy: 0.0500 - 359ms/epoch - 4ms/step**

**Epoch 100/100**

**100/100 - 0s - loss: 0.0307 - accuracy: 0.0100 - val\_loss: 0.0336 - val\_accuracy: 0.0500 - 408ms/epoch - 4ms/step**

**results=model.predict(x\_test)**

**#1/1 [==============================] - 0s 479ms/step**

**results**

**array([[0.25683218],**

**[0.18767577],**

**[0.91957057],**

**[0.2264341 ],**

**[0.7294549 ],**

**[0.29109192],**

**[0.62157506],**

**[0.92222303],**

**[0.47553238],**

**[0.54921806],**

**[0.4999913 ],**

**[0.14717652],**

**[0.9167971 ],**

**[0.3096802 ],**

**[0.20607205],**

**[0.4276456 ],**

**[0.21263492],**

**[0.33933866],**

**[0.40447798],**

**[0.63323027]], dtype=float32)**

**plt.scatter(range(20),results,c="r")**

**plt.scatter(range(20),y\_test,c="g")**

**plt.show()**

**plt.plot(history.history['loss'])**

**plt.show()**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* LSTM PREDICTING TIME SERIES DATA \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**from keras.datasets import imdb**

**from keras.preprocessing.text import Tokenizer**

**from keras.utils import pad\_sequences**

**from keras import Sequential**

**from keras.layers import Dense,SimpleRNN,Embedding,Flatten,LSTM**

**import matplotlib.pyplot as plt**

**(X\_train,y\_train),(X\_test,y\_test) = imdb.load\_data()**

**#Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/imdb.npz**

**17464789/17464789 [==============================] - 0s 0us/step**

**X\_train.shape**

**#(25000,)**

**X\_test.shape**

**#(25000,)**

**y\_test[100]**

**#1**

**X\_train[100]**

**# 337,**

**7,**

**628,**

**2219,**

**5,**

**28,**

**285,**

**15,**

**240,**

**93,**

**23,**

**288,**

**549,**

**18,**

**1455,**

**673,**

**4,**

**241,**

**534,**

**3635,**

**...**

**14,**

**241,**

**46,**

**7,**

**158]**

**X\_train = pad\_sequences(X\_train,padding='post',maxlen=50)**

**X\_test = pad\_sequences(X\_test,padding='post',maxlen=50)**

**X\_train.shape**

**X\_train[0]**

**#1**

**(25000,)**

**(25000,)**

**Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/imdb.npz**

**17464789/17464789 [==============================] - 0s 0us/step**

**[1,**

**13,**

**244,**

**6,**

**87,**

**337,**

**7,**

**628,**

**2219,**

**5,**

**28,**

**285,**

**15,**

**240,**

**93,**

**23,**

**288,**

**549,**

**18,**

**1455,**

**673,**

**4,**

**241,**

**534,**

**3635,**

**...**

**14,**

**241,**

**46,**

**7,**

**158]**

**Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...**

**(25000, 50)**

**array([2071, 56, 26, 141, 6, 194, 7486, 18, 4, 226, 22,**

**21, 134, 476, 26, 480, 5, 144, 30, 5535, 18, 51,**

**36, 28, 224, 92, 25, 104, 4, 226, 65, 16, 38,**

**1334, 88, 12, 16, 283, 5, 16, 4472, 113, 103, 32,**

**15, 16, 5345, 19, 178, 32], dtype=int32)**

**model = Sequential()**

**#model.add(Embedding(10000,2,50))**

**model.add(LSTM(32,input\_shape=(50,1),return\_sequences=False))**

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

**model.summary()**

**#1**

**(25000,)**

**(25000,)**

**Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/imdb.npz**

**17464789/17464789 [==============================] - 0s 0us/step**

**[1,**

**13,**

**244,**

**6,**

**87,**

**337,**

**7,**

**628,**

**2219,**

**5,**

**28,**

**285,**

**15,**

**240,**

**93,**

**23,**

**288,**

**549,**

**18,**

**1455,**

**673,**

**4,**

**241,**

**534,**

**3635,**

**...**

**14,**

**241,**

**46,**

**7,**

**158]**

**Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...**

**(25000, 50)**

**array([2071, 56, 26, 141, 6, 194, 7486, 18, 4, 226, 22,**

**21, 134, 476, 26, 480, 5, 144, 30, 5535, 18, 51,**

**36, 28, 224, 92, 25, 104, 4, 226, 65, 16, 38,**

**1334, 88, 12, 16, 283, 5, 16, 4472, 113, 103, 32,**

**15, 16, 5345, 19, 178, 32], dtype=int32)**

**Model: "sequential"**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Layer (type) Output Shape Param #**

**=================================================================**

**lstm (LSTM) (None, 32) 4352**

**dense (Dense) (None, 1) 33**

**=================================================================**

**Total params: 4385 (17.13 KB)**

**Trainable params: 4385 (17.13 KB)**

**Non-trainable params: 0 (0.00 Byte)**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**model.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['acc'])**

**history = model.fit(X\_train, y\_train,epochs=10,validation\_data=(X\_test,y\_test))**

**##1**

**(25000,)**

**(25000,)**

**Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/imdb.npz**

**17464789/17464789 [==============================] - 0s 0us/step**

**[1,**

**13,**

**244,**

**6,**

**87,**

**337,**

**7,**

**628,**

**2219,**

**5,**

**28,**

**285,**

**15,**

**240,**

**93,**

**23,**

**288,**

**549,**

**18,**

**1455,**

**673,**

**4,**

**241,**

**534,**

**3635,**

**...**

**14,**

**241,**

**46,**

**7,**

**158]**

**Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...**

**(25000, 50)**

**array([2071, 56, 26, 141, 6, 194, 7486, 18, 4, 226, 22,**

**21, 134, 476, 26, 480, 5, 144, 30, 5535, 18, 51,**

**36, 28, 224, 92, 25, 104, 4, 226, 65, 16, 38,**

**1334, 88, 12, 16, 283, 5, 16, 4472, 113, 103, 32,**

**15, 16, 5345, 19, 178, 32], dtype=int32)**

**Model: "sequential"**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Layer (type) Output Shape Param #**

**=================================================================**

**lstm (LSTM) (None, 32) 4352**

**dense (Dense) (None, 1) 33**

**=================================================================**

**Total params: 4385 (17.13 KB)**

**Trainable params: 4385 (17.13 KB)**

**Non-trainable params: 0 (0.00 Byte)**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Epoch 1/10**

**782/782 [==============================] - 31s 36ms/step - loss: 0.6859 - acc: 0.5484 - val\_loss: 0.6879 - val\_acc: 0.5429**

**Epoch 2/10**

**782/782 [==============================] - 23s 29ms/step - loss: 0.6852 - acc: 0.5492 - val\_loss: 0.6872 - val\_acc: 0.5483**

**Epoch 3/10**

**782/782 [==============================] - 27s 34ms/step - loss: 0.6856 - acc: 0.5479 - val\_loss: 0.6866 - val\_acc: 0.5510**

**Epoch 4/10**

**782/782 [==============================] - 23s 29ms/step - loss: 0.6846 - acc: 0.5502 - val\_loss: 0.6859 - val\_acc: 0.5521**

**Epoch 5/10**

**782/782 [==============================] - 23s 29ms/step - loss: 0.6846 - acc: 0.5492 - val\_loss: 0.6860 - val\_acc: 0.5533**

**Epoch 6/10**

**782/782 [==============================] - 22s 28ms/step - loss: 0.6843 - acc: 0.5544 - val\_loss: 0.6878 - val\_acc: 0.5413**

**Epoch 7/10**

**782/782 [==============================] - 23s 30ms/step - loss: 0.6848 - acc: 0.5544 - val\_loss: 0.6896 - val\_acc: 0.5391**

**Epoch 8/10**

**782/782 [==============================] - 23s 29ms/step - loss: 0.6846 - acc: 0.5523 - val\_loss: 0.6899 - val\_acc: 0.5360**

**Epoch 9/10**

**782/782 [==============================] - 23s 30ms/step - loss: 0.6839 - acc: 0.5528 - val\_loss: 0.6872 - val\_acc: 0.5509**

**Epoch 10/10**

**782/782 [==============================] - 27s 35ms/step - loss: 0.6834 - acc: 0.5534 - val\_loss: 0.6857 - val\_acc: 0.5559**

**results=model.predict(X\_test)**

**#782/782 [==============================] - 11s 12ms/step**

**y\_test.shape**

**results=results.reshape(-1)**

**results.shape**

**#(25000,)**

**if results.shape[0] != y\_test.shape[0]:**

**raise ValueError("results and y\_test must have the same number of elements")**

**plt.scatter(range(len(results)), results, c="r")**

**plt.scatter(range(len(y\_test)), y\_test, c="g")**

**plt.show()**

**plt.plot(history.history['loss'])**

**plt.show()**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* IMPLEMENT SEQ2SEQ MODEL FOR NEURAL MACHINE TRANSLATION\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**from keras.models import Model**

**from keras.layers import Input, LSTM, Dense,Embedding,RepeatVector**

**import numpy as np**

**from keras.preprocessing.sequence import pad\_sequences**

**from keras.models import load\_model**

**from keras import optimizers**

**from numpy import array,random,take**

**import string**

**import re**

**from keras.preprocessing.text import Tokenizer**

**from keras.models import Sequential**

**from numpy import argmax**

**data\_path='/content/fra.txt'**

**with open(data\_path, 'r', encoding='utf-8') as f:**

**lines = f.read()**

**lines**

**def to\_lines(text):**

**sents=text.strip().split("\n")**

**sents=[i.split('\t') for i in sents]**

**return sents**

**fra\_eng=to\_lines(lines)**

**fra\_eng[:5]**

**fra\_eng=array(fra\_eng)**

**fra\_eng[:5]**

**fra\_eng.shape**

**fra\_eng=fra\_eng[:50000,:]**

**fra\_eng=fra\_eng[:,[0,1]]**

**fra\_eng[:5]**

**#REMOVE PUNCTUATION**

**fra\_eng[:,0]=[s.translate(str.maketrans('','',string.punctuation))for s in fra\_eng[:,0] ]**

**fra\_eng[:,1]=[s.translate(str.maketrans('','',string.punctuation))for s in fra\_eng[:,1] ]**

**fra\_eng[:5]**

**#convert text to lower case**

**for i in range(len(fra\_eng)):**

**fra\_eng[i,0]=fra\_eng[i,0].lower()**

**fra\_eng[i,1]=fra\_eng[i,1].lower()**

**fra\_eng**

**#function to build a tokenizer**

**def tokenization(lines):**

**tokenizer=Tokenizer()**

**tokenizer.fit\_on\_texts(lines)**

**return tokenizer**

**#prepare english tokenizer**

**eng\_tokenizer=tokenization(fra\_eng[:,0])**

**eng\_vocab\_size=len(eng\_tokenizer.word\_index)+1**

**eng\_length=8**

**print(eng\_vocab\_size)**

**#prepare english tokenizer**

**fra\_tokenizer=tokenization(fra\_eng[:,1])**

**fra\_vocab\_size=len(fra\_tokenizer.word\_index)+1**

**fra\_length=8**

**print(fra\_vocab\_size)**

**#encode and pad sequences**

**def encode\_sequences(tokenizer,length,lines):**

**#integer encode sequences**

**seq=tokenizer.texts\_to\_sequences(lines)**

**#pad sequences with 0**

**seq=pad\_sequences(seq,maxlen=length,padding='post')**

**return seq**

**from sklearn.model\_selection import train\_test\_split**

**train,test=train\_test\_split(fra\_eng,test\_size=0.2,random\_state=3)**

**#prepare training data**

**trainX=encode\_sequences(fra\_tokenizer,fra\_length,train[:,1])**

**trainY=encode\_sequences(eng\_tokenizer,eng\_length,train[:,0])**

**#prepare validation data**

**testX=encode\_sequences(fra\_tokenizer,fra\_length,test[:,1])**

**testY=encode\_sequences(eng\_tokenizer,eng\_length,test[:,0])**

**def define\_model(in\_vocab,out\_vocab,in\_timesteps,out\_timesteps,units):**

**model=Sequential() model.add(Embedding(in\_vocab,units,input\_length=in\_timesteps,mask\_zero='True'))**

**model.add(LSTM(units))**

**model.add(RepeatVector(out\_timesteps))**

**model.add(LSTM(units,return\_sequences='True'))**

**model.add(Dense(out\_vocab,activation="softmax"))**

**return model**

**model=define\_model(fra\_vocab\_size,eng\_vocab\_size,fra\_length,eng\_length,512)**

**rms=optimizers.RMSprop(learning\_rate=0.001)**

**model.compile(optimizer=rms,loss='sparse\_categorical\_crossentropy')**

**#train the model**

**model.fit(trainX,trainY.reshape(trainY.shape[0],trainY.shape[1],1),**

**batch\_size=512,**

**epochs=50,**

**validation\_split=0.2)**

**pred=model.predict(testX.reshape(testX.shape[0],testX.shape[1],1))**

**predicted\_classes = pred.argmax(axis=1)**

**predicted\_classes**

**def get\_words(n,tokenizer):**

**for word,index in tokenizer.word\_index.items():**

**if index==n:**

**return word**

**return None**

**pred\_text=[]**

**for i in predicted\_classes:**

**temp=[]**

**for j in range(len(i)):**

**t=get\_words(i[j],eng\_tokenizer)**

**if j>0:**

**if((t==get\_words(i[j-1],eng\_tokenizer)) or (t==None)):**

**temp.append('')**

**else:**

**temp.append(t)**

**else:**

**if(t==None):**

**temp.append('')**

**else:**

**temp.append(t)**

**pred\_text.append(' '.join(temp))**

**pred\_text[5]**

**import pandas as pd**

**df=pd.DataFrame({"actual": test[:,0], "predicted": pred\_text})**

**df.sample(15)**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* NEXT WORD PREDICTION USING LSTM \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**import numpy as np**

**import tensorflow as tf**

**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**

**# Reading corpus the text file**

**with open("/content/sample\_data/IndiaUS.txt", 'r', encoding='utf-8') as myfile:**

**mytext = myfile.read()**

**mytext**

**mytokenizer = Tokenizer()**

**mytokenizer.fit\_on\_texts([mytext])**

**total\_words = len(mytokenizer.word\_index) + 1**

**print(total\_words)**

**mytokenizer.word\_index**

**my\_input\_sequences = []**

**for line in mytext.split('\n'):**

**#print(line)**

**token\_list = mytokenizer.texts\_to\_sequences([line])[0]**

**#print(token\_list)**

**for i in range(1, len(token\_list)):**

**my\_n\_gram\_sequence = token\_list[:i+1]**

**#print(my\_n\_gram\_sequence)**

**my\_input\_sequences.append(my\_n\_gram\_sequence)**

**print(input\_sequences)**

**max\_sequence\_len = max([len(seq) for seq in my\_input\_sequences])**

**input\_sequences = np.array(pad\_sequences(my\_input\_sequences, maxlen=max\_sequence\_len, padding='pre'))**

**input\_sequences[1]**

**#array([ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,**

**0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,**

**0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,**

**0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,**

**0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,**

**0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,**

**0, 0, 99, 4, 177], dtype=int32)**

**X = input\_sequences[:, :-1]**

**y = input\_sequences[:, -1]**

**X[1]**

**#array([ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,**

**0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,**

**0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,**

**0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,**

**0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 99, 4],**

**dtype=int32)**

**y[1]**

**177**

**y = np.array(tf.keras.utils.to\_categorical(y, num\_classes=total\_words))**

**y[0]**

**model = Sequential()**

**model.add(Embedding(total\_words, 100, input\_length=max\_sequence\_len-1))**

**model.add(LSTM(150))**

**model.add(Dense(total\_words, activation='softmax'))**

**print(model.summary())**

**##Model: "sequential"**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Layer (type) Output Shape Param #**

**=================================================================**

**embedding (Embedding) (None, 82, 100) 59900**

**lstm (LSTM) (None, 150) 150600**

**dense (Dense) (None, 599) 90449**

**=================================================================**

**Total params: 300949 (1.15 MB)**

**Trainable params: 300949 (1.15 MB)**

**Non-trainable params: 0 (0.00 Byte)**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**model.compile(loss='categorical\_crossentropy', optimizer='adam', metrics=['accuracy'])**

**model.fit(X, y, epochs=100, verbose=1)**

**input\_text = "Joe biden"**

**predict\_next\_words= 6**

**for \_ in range(predict\_next\_words):**

**token\_list = mytokenizer.texts\_to\_sequences([input\_text])[0]**

**print(token\_list)**

**token\_list = pad\_sequences([token\_list], maxlen=max\_sequence\_len-1, padding='pre')**

**predicted = np.argmax(model.predict(token\_list), axis=-1)**

**output\_word = ""**

**for word, index in mytokenizer.word\_index.items():**

**if index == predicted:**

**output\_word = word**

**break**

**input\_text += " " + output\_word**

**print(input\_text)**

**input\_text = "Joe biden"**

**predict\_next\_words= 6**

**for \_ in range(predict\_next\_words):**

**token\_list = mytokenizer.texts\_to\_sequences([input\_text])[0]**

**print(token\_list)**

**token\_list = pad\_sequences([token\_list], maxlen=max\_sequence\_len-1, padding='pre')**

**predicted = model.predict(token\_list)**

**predicted.argmax()**

***(2 sequential ApI 3 parts)***

**# (i) Import necessary libraries**

**import tensorflow as tf**

**from tensorflow.keras.models import Sequential**

**from tensorflow.keras.layers import Dense**

**# Assuming 10 features in your input data**

**num\_features = 10**

**# (ii) Construct the model**

**model = Sequential([**

**Dense(4, input\_shape=(num\_features,), activation='tanh', name='layer1\_tanh'),**

**Dense(2, activation='tanh', name='layer2\_tanh'),**

**Dense(1, activation='sigmoid', name='layer3\_sigmoid')**

**])a**

**# (iii) Compile the model**

**model.compile(optimizer='adam',**

**loss='binary\_crossentropy', # Appropriate for binary classification**

**metrics=['accuracy']) # Binary classification metric**

**# Print model summary**

**model.summary()**

***(3 3bits 24 integers )***

**import numpy as np**

**import tensorflow as tf**

**# (i) Construct a vector consisting of the first 24 integers using NumPy**

**numpy\_vector = np.arange(1, 25)**

**# (ii) Convert the NumPy vector into a Tensor of rank 3**

**# Reshape the vector into a rank 3 tensor**

**rank\_3\_tensor = tf.constant(numpy\_vector.reshape((2, 3, 4)))**

**print("NumPy Vector:")**

**print(numpy\_vector)**

**print("\nTensor of Rank 3:")**

**print(rank\_3\_tensor.numpy())**

***(4 FUNCTIONAL API 3 bits)***

**# (i) Importing necessary libraries**

**from tensorflow.keras.models import Model**

**from tensorflow.keras.layers import Input, Conv2D, MaxPooling2D, Flatten, Dense**

**from tensorflow.keras.optimizers import Adam**

**# Assuming the input shape is 28x28x1**

**input\_shape = (28, 28, 1)**

**# (ii) Constructing the model using Functional API**

**inputs = Input(shape=input\_shape)**

**# Convolutional layer 1**

**conv1 = Conv2D(32, kernel\_size=(3, 3), activation='relu', padding='same')(inputs)**

**pool1 = MaxPooling2D(pool\_size=(2, 2))(conv1)**

**# Convolutional layer 2**

**conv2 = Conv2D(64, kernel\_size=(3, 3), activation='relu', padding='same')(pool1)**

**pool2 = MaxPooling2D(pool\_size=(2, 2))(conv2)**

**# Flatten layer**

**flatten = Flatten()(pool2)**

**# Fully connected layer**

**dense1 = Dense(128, activation='relu')(flatten)**

**# Output layer**

**outputs = Dense(num\_classes, activation='softmax')(dense1) # Assuming num\_classes is defined**

**# Creating the model**

**model = Model(inputs=inputs, outputs=outputs)**

**# (iii) Compiling the model**

**model.compile(optimizer=Adam(),**

**loss='categorical\_crossentropy', # Appropriate for multiclass classification**

**metrics=['accuracy']) # Considering multiclass classification**

**# Printing the model summary**

**print(model.summary())**

***(5 LSTM RMSProp )***

**import numpy as np**

**from tensorflow.keras.datasets import imdb**

**from tensorflow.keras.preprocessing.sequence import pad\_sequences**

**from tensorflow.keras.models import Sequential**

**from tensorflow.keras.layers import Embedding, LSTM, Dense**

**# Load the IMDB dataset**

**num\_words = 10000**

**maxlen = 200**

**(X\_train, y\_train), (X\_test, y\_test) = imdb.load\_data(num\_words=num\_words)**

**# Preprocess the data**

**X\_train = pad\_sequences(X\_train, maxlen=maxlen)**

**X\_test = pad\_sequences(X\_test, maxlen=maxlen)**

**# Define the LSTM model**

**model = Sequential()**

**model.add(Embedding(num\_words, 128, input\_length=maxlen))**

**model.add(LSTM(64, dropout=0.2, recurrent\_dropout=0.2))**

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

**# Compile the model with RMSProp optimizer**

**model.compile(loss='binary\_crossentropy', optimizer='rmsprop', metrics=['accuracy'])**

**# Train the model**

**batch\_size = 128**

**epochs = 5**

**model.fit(X\_train, y\_train, batch\_size=batch\_size, epochs=epochs, validation\_data=(X\_test, y\_test))**

**# Evaluate the model**

**score, acc = model.evaluate(X\_test, y\_test, batch\_size=batch\_size)**

**print("Test score:", score)**

**print("Test accuracy:", acc)**

***(6 CNN Dataset)***

**import numpy as np**

**import matplotlib.pyplot as plt**

**import tensorflow as tf**

**from tensorflow.keras.preprocessing.image import ImageDataGenerator**

**from tensorflow.keras.models import Sequential**

**from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout**

**from tensorflow.keras.optimizers import Adam**

**# Load data**

**train\_dir = 'path/to/train' # Path to training directory**

**test\_dir = 'path/to/test'# Path to test directory**

**# Data augmentation and normalization**

**train\_datagen = ImageDataGenerator(**

**rescale=1./255,**

**rotation\_range=40,**

**width\_shift\_range=0.2,**

**height\_shift\_range=0.2,**

**shear\_range=0.2,**

**zoom\_range=0.2,**

**horizontal\_flip=True,**

**fill\_mode='nearest')**

**test\_datagen=ImageDataGenerator(rescale=1./255)**

**# Batch size**

**batch\_size = 32**

**# Load and augment training data**

**train\_generator = train\_datagen.flow\_from\_directory(**

**train\_dir,**

**target\_size=(150, 150),**

**batch\_size=batch\_size,**

**class\_mode='binary')**

**# Load and augment test data**

**test\_generator =test\_datagen.flow\_from\_directory(**

**test\_dir,**

**target\_size=(150, 150),**

**batch\_size=batch\_size,**

**class\_mode='binary')**

**# CNN model**

**model = Sequential([**

**Conv2D(32, (3, 3), activation='relu', input\_shape=(150, 150, 3)),**

**MaxPooling2D(2, 2),**

**Conv2D(64, (3, 3), activation='relu'),**

**MaxPooling2D(2, 2),**

**Conv2D(128, (3, 3), activation='relu'),**

**MaxPooling2D(2, 2),**

**Conv2D(128, (3, 3), activation='relu'),**

**MaxPooling2D(2, 2),**

**Flatten(),**

**Dropout(0.5),**

**Dense(512, activation='relu'),**

**Dense(1, activation='sigmoid')])**

**# Compile model**

**model.compile(optimizer=Adam(lr=1e-4),**

**loss='binary\_crossentropy',**

**metrics=['accuracy'])**

**# Train model**

**history = model.fit(**

**train\_generator,**

**steps\_per\_epoch=train\_generator.samples // batch\_size,**

**epochs=20,**

**validation\_data=test\_generator,**

**validation\_steps=test\_generator.samples // batch\_size)**

**# Plot training history**

**plt.plot(history.history['accuracy'], label='accuracy')**

**plt.plot(history.history['val\_accuracy'], label = 'val\_accuracy')**

**plt.xlabel('Epoch')**

**plt.ylabel('Accuracy')**

**plt.ylim([0, 1])**

**plt.legend(loc='lower right')**

**plt.show()**

**(7 MNIST, CIFAR-10 datasets)**

**import numpy as np**

**import tensorflow as tf**

**from tensorflow.keras.datasets import mnist, fashion\_mnist, cifar10**

**from tensorflow.keras.models import Sequential**

**from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense**

**# Load datasets**

**(x\_train\_mnist, y\_train\_mnist), (x\_test\_mnist, y\_test\_mnist) = mnist.load\_data()**

**(x\_train\_fashion, y\_train\_fashion), (x\_test\_fashion, y\_test\_fashion) = fashion\_mnist.load\_data()**

**(x\_train\_cifar, y\_train\_cifar), (x\_test\_cifar, y\_test\_cifar) = cifar10.load\_data()**

**# Normalize pixel values to be between 0 and 1**

**x\_train\_mnist, x\_test\_mnist = x\_train\_mnist / 255.0, x\_test\_mnist / 255.0**

**x\_train\_fashion, x\_test\_fashion = x\_train\_fashion / 255.0, x\_test\_fashion / 255.0**

**x\_train\_cifar, x\_test\_cifar = x\_train\_cifar / 255.0, x\_test\_cifar / 255.0**

**# Add a channel dimension for CNN**

**x\_train\_mnist = np.expand\_dims(x\_train\_mnist, axis=-1)**

**x\_test\_mnist = np.expand\_dims(x\_test\_mnist, axis=-1)**

**x\_train\_fashion = np.expand\_dims(x\_train\_fashion, axis=-1)**

**x\_test\_fashion = np.expand\_dims(x\_test\_fashion, axis=-1)**

**# Define CNN model**

**def create\_model(input\_shape, num\_classes):**

**model = Sequential([**

**Conv2D(32, (3, 3), activation='relu', input\_shape=input\_shape),**

**MaxPooling2D((2, 2)),**

**Conv2D(64, (3, 3), activation='relu'),**

**MaxPooling2D((2, 2)),**

**Conv2D(128, (3, 3), activation='relu'),**

**MaxPooling2D((2, 2)),**

**Flatten(),**

**Dense(128, activation='relu'),**

**Dense(num\_classes, activation='softmax')**

**])**

**return model**

**# Compile and train model for MNIST**

**model\_mnist = create\_model((28, 28, 1), 10)**

**model\_mnist.compile(optimizer='adam', loss='sparse\_categorical\_crossentropy', metrics=['accuracy'])**

**print("\nTraining MNIST model for 5 epochs...")**

**model\_mnist.fit(x\_train\_mnist, y\_train\_mnist, epochs=5, validation\_data=(x\_test\_mnist, y\_test\_mnist))**

**print("\nTraining MNIST model for 10 epochs...")**

**model\_mnist.fit(x\_train\_mnist, y\_train\_mnist, epochs=10, validation\_data=(x\_test\_mnist, y\_test\_mnist))**

**print("\nTraining MNIST model for 20 epochs...")**

**model\_mnist.fit(x\_train\_mnist, y\_train\_mnist, epochs=20, validation\_data=(x\_test\_mnist, y\_test\_mnist))**

**# Compile and train model for Fashion MNIST**

**model\_fashion = create\_model((28, 28, 1), 10)**

**model\_fashion.compile(optimizer='adam', loss='sparse\_categorical\_crossentropy', metrics=['accuracy'])**

**print("\nTraining Fashion MNIST model for 5 epochs...")**

**model\_fashion.fit(x\_train\_fashion, y\_train\_fashion, epochs=5, validation\_data=(x\_test\_fashion, y\_test\_fashion))**

**print("\nTraining Fashion MNIST model for 10 epochs...")**

**model\_fashion.fit(x\_train\_fashion, y\_train\_fashion, epochs=10, validation\_data=(x\_test\_fashion, y\_test\_fashion))**

**print("\nTraining Fashion MNIST model for 20 epochs...")**

**model\_fashion.fit(x\_train\_fashion, y\_train\_fashion, epochs=20, validation\_data=(x\_test\_fashion, y\_test\_fashion))**

**# Compile and train model for CIFAR-10**

**model\_cifar = create\_model((32, 32, 3), 10)**

**model\_cifar.compile(optimizer='adam', loss='sparse\_categorical\_crossentropy', metrics=['accuracy'])**

**print("\nTraining CIFAR-10 model for 5 epochs...")**

**model\_cifar.fit(x\_train\_cifar, y\_train\_cifar, epochs=5, validation\_data=(x\_test\_cifar, y\_test\_cifar))**

**print("\nTraining CIFAR-10 model for 10 epochs...")**

**model\_cifar.fit(x\_train\_cifar, y\_train\_cifar, epochs=10, validation\_data=(x\_test\_cifar, y\_test\_cifar))**

**print("\nTraining CIFAR-10 model for 20 epochs...")**

**model\_cifar.fit(x\_train\_cifar, y\_train\_cifar, epochs=20, validation\_data=(x\_test\_cifar, y\_test\_cifar))**

***(8 VGG-16 & 19 CNN )***

**import numpy as np**

**import tensorflow as tf**

**from tensorflow.keras.models import Sequential**

**from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense**

**from tensorflow.keras.applications import VGG16, VGG19**

**from tensorflow.keras.datasets import cifar10**

**from tensorflow.keras.utils import to\_categorical**

**# Load and preprocess CIFAR-10 dataset**

**(X\_train, y\_train), (X\_test, y\_test) = cifar10.load\_data()**

**X\_train = X\_train.astype('float32') / 255**

**X\_test = X\_test.astype('float32') / 255**

**y\_train = to\_categorical(y\_train, num\_classes=10)**

**y\_test = to\_categorical(y\_test, num\_classes=10)**

**# VGG16 model**

**def vgg16\_model():**

**model = Sequential([**

**VGG16(weights='imagenet', include\_top=False, input\_shape=(32, 32, 3)),**

**Flatten(),**

**Dense(512, activation='relu'),**

**Dense(10, activation='softmax')**

**])**

**return model**

**# VGG19 model**

**def vgg19\_model():**

**model = Sequential([**

**VGG19(weights='imagenet', include\_top=False, input\_shape=(32, 32, 3)),**

**Flatten(),**

**Dense(512, activation='relu'),**

**Dense(10, activation='softmax')**

**])**

**return model**

**# Compile and train VGG16 model**

**model\_vgg16 = vgg16\_model()**

**model\_vgg16.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])**

**model\_vgg16.fit(X\_train, y\_train, batch\_size=64, epochs=10, validation\_data=(X\_test, y\_test))**

**# Evaluate VGG16 model**

**vgg16\_loss, vgg16\_acc = model\_vgg16.evaluate(X\_test, y\_test)**

**print("VGG16 Test Accuracy:", vgg16\_acc)**

**# Compile and train VGG19 model**

**model\_vgg19 = vgg19\_model()**

**model\_vgg19.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])**

**model\_vgg19.fit(X\_train, y\_train, batch\_size=64, epochs=10, validation\_data=(X\_test, y\_test))**

**# Evaluate VGG19 model**

**vgg19\_loss, vgg19\_acc = model\_vgg19.evaluate(X\_test, y\_test)**

**print("VGG19 Test Accuracy:", vgg19\_acc)**

***(9 Dropout layer.)***

**import numpy as np**

**import tensorflow as tf**

**from tensorflow.keras.models import Sequential**

**from tensorflow.keras.layers import Dense, Dropout**

**# Example data**

**X = np.random.random((1000, 20)) # 1000 samples with 20 features**

**y = np.random.randint(2, size=(1000,)) # Binary classification labels**

**# Define a sequential model**

**model = Sequential([**

**Dense(64, activation='relu', input\_shape=(20,)),**

**Dropout(0.5), # Dropout layer with dropout rate of 0.5 (50%)**

**Dense(64, activation='relu'),**

**Dropout(0.5),**

**Dense(1, activation='sigmoid')**

**])**

**# Compile the model**

**model.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])**

**# Train the model**

**model.fit(X, y, epochs=10, batch\_size=32, validation\_split=0.2)**

***(10 Tranfer learning)***

**import numpy as np**

**import tensorflow as tf**

**from tensorflow.keras.applications import VGG16**

**from tensorflow.keras.models import Sequential**

**from tensorflow.keras.layers import Dense, Flatten**

**from tensorflow.keras.optimizers import Adam**

**from tensorflow.keras.datasets import cifar10**

**from tensorflow.keras.utils import to\_categorical**

**# Load and preprocess CIFAR-10 dataset**

**(x\_train, y\_train), (x\_test, y\_test) = cifar10.load\_data()**

**x\_train = x\_train.astype('float32') / 255.0**

**x\_test = x\_test.astype('float32') / 255.0**

**y\_train = to\_categorical(y\_train, num\_classes=10)**

**y\_test = to\_categorical(y\_test, num\_classes=10)**

**# Load pre-trained VGG16 model (excluding the top dense layers)**

**base\_model = VGG16(weights='imagenet', include\_top=False, input\_shape=(32, 32, 3))**

**# Freeze convolutional layers**

**for layer in base\_model.layers:**

**layer.trainable = False**

**# Create a new model on top of the pre-trained base model**

**model = Sequential([**

**base\_model,**

**Flatten(),**

**Dense(512, activation='relu'),**

**Dense(10, activation='softmax')**

**])**

**# Compile the model**

**model.compile(optimizer=Adam(lr=0.001), loss='categorical\_crossentropy', metrics=['accuracy'])**

**# Train the model**

**history = model.fit(x\_train, y\_train, epochs=10, batch\_size=128, validation\_data=(x\_test, y\_test))**

**# Evaluate the model**

**test\_loss, test\_acc = model.evaluate(x\_test, y\_test)**

**print("Test accuracy:", test\_acc)**

***(11 Early stoping.)***

**import numpy as np**

**import tensorflow as tf**

**from tensorflow.keras.models import Sequential**

**from tensorflow.keras.layers import Dense**

**from tensorflow.keras.callbacks import EarlyStopping**

**from sklearn.model\_selection import train\_test\_split**

**# Generate some example data**

**np.random.seed(0)**

**X = np.random.rand(1000, 10)**

**y = np.random.randint(2, size=1000)**

**# Split the data into training and testing sets**

**X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)**

**# Define the deep learning model**

**model = Sequential([**

**Dense(64, activation='relu', input\_shape=(10,)),**

**Dense(32, activation='relu'),**

**Dense(1, activation='sigmoid')**

**])**

**# Compile the model**

**model.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])**

**# Define early stopping criteria**

**early\_stopping = EarlyStopping(monitor='val\_loss', patience=3, restore\_best\_weights=True)**

**# Train the model with early stopping**

**history = model.fit(X\_train, y\_train, epochs=100, batch\_size=32, validation\_data=(X\_test, y\_test), callbacks=[early\_stopping])**

**# Evaluate the model**

**test\_loss, test\_acc = model.evaluate(X\_test, y\_test)**

**print("Test accuracy:", test\_acc)**

***(12 Data Augmentation Techique)***

**import numpy as np**

**import tensorflow as tf**

**from tensorflow.keras.models import Sequential**

**from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense**

**from tensorflow.keras.preprocessing.image import ImageDataGenerator**

**from tensorflow.keras.datasets import cifar10**

**from tensorflow.keras.utils import to\_categorical**

**# Load CIFAR-10 dataset**

**(x\_train, y\_train), (x\_test, y\_test) = cifar10.load\_data()**

**# Normalize pixel values to be between 0 and 1**

**x\_train = x\_train.astype('float32') / 255.0**

**x\_test = x\_test.astype('float32') / 255.0**

**# Convert class vectors to binary class matrices**

**num\_classes = 10**

**y\_train = to\_categorical(y\_train, num\_classes)**

**y\_test = to\_categorical(y\_test, num\_classes)**

**# Define the CNN model**

**model = Sequential([**

**Conv2D(32, (3, 3), activation='relu', input\_shape=(32, 32, 3)),**

**MaxPooling2D((2, 2)),**

**Conv2D(64, (3, 3), activation='relu'),**

**MaxPooling2D((2, 2)),**

**Conv2D(128, (3, 3), activation='relu'),**

**MaxPooling2D((2, 2)),**

**Flatten(),**

**Dense(512, activation='relu'),**

**Dense(num\_classes, activation='softmax')**

**])**

**# Compile the model**

**model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])**

**# Define data augmentation generator**

**datagen = ImageDataGenerator(**

**rotation\_range=20, # randomly rotate images in the range (degrees, 0 to 180)**

**width\_shift\_range=0.1, # randomly shift images horizontally (fraction of total width)**

**height\_shift\_range=0.1, # randomly shift images vertically (fraction of total height)**

**horizontal\_flip=True # randomly flip images horizontally**

**)**

**# Fit the model using data augmentation generator**

**batch\_size = 32**

**epochs = 50**

**datagen.fit(x\_train)**

**model.fit(datagen.flow(x\_train, y\_train, batch\_size=batch\_size), epochs=epochs, validation\_data=(x\_test, y\_test))**

***(13 predict next word..)***

**import numpy as np**

**import tensorflow as tf**

**from tensorflow.keras.models import Sequential**

**from tensorflow.keras.layers import LSTM, Dense, Embedding**

**from tensorflow.keras.preprocessing.text import Tokenizer**

**from tensorflow.keras.preprocessing.sequence import pad\_sequences**

**# Example text data**

**text\_data = [**

**"The quick brown fox jumps over the lazy dog",**

**"The cat in the hat",**

**"A bird in the hand is worth two in the bush"**

**]**

**# Tokenize the text data**

**tokenizer = Tokenizer()**

**tokenizer.fit\_on\_texts(text\_data)**

**vocab\_size = len(tokenizer.word\_index) + 1**

**# Convert text data to sequences of integers**

**sequences = tokenizer.texts\_to\_sequences(text\_data)**

**# Generate input sequences and labels for training**

**input\_sequences = []**

**next\_words = []**

**for sequence in sequences:**

**for i in range(1, len(sequence)):**

**input\_sequence = sequence[:i]**

**label = sequence[i] input\_sequences.append(input\_sequence)**

**next\_words.append(label)**

**# Pad input sequences to have the same length**

**max\_sequence\_length = max([len(seq) for seq in input\_sequences])**

**input\_sequences = pad\_sequences(input\_sequences, maxlen=max\_sequence\_length, padding='pre')**

**# Convert to numpy arrays**

**input\_sequences = np.array(input\_sequences)**

**next\_words = np.array(next\_words)**

**# Define the LSTM model**

**model = Sequential([**

**Embedding(vocab\_size, 100, input\_length=max\_sequence\_length - 1),**

**LSTM(100),**

**Dense(vocab\_size, activation='softmax')**

**])**

**# Compile the model**

**model.compile(loss='sparse\_categorical\_crossentropy', optimizer='adam', metrics=['accuracy'])**

**# Train the model**

**model.fit(input\_sequences, next\_words, epochs=100, verbose=2)**

**# Function to generate the next word given a seed text**

**def generate\_next\_word(seed\_text):**

**token\_list = tokenizer.texts\_to\_sequences([seed\_text])[0]**

**token\_list = pad\_sequences([token\_list], maxlen=max\_sequence\_length - 1, padding='pre')**

**predicted\_probs = model.predict(token\_list)[0]**

**predicted\_index = np.argmax(predicted\_probs)**

**predicted\_word = tokenizer.index\_word[predicted\_index]**

**return predicted\_word**

**# Example usage**

**seed\_text = "The cat"**

**next\_word = generate\_next\_word(seed\_text)**

**print("Next word prediction:", next\_word)**

***(14 TensorFlow running 3 bits)***

**import tensorflow as tf**

**# (i) Get the version of TensorFlow running on your machine**

**print("TensorFlow version:", tf.\_\_version\_\_)**

**# (ii) Get the type & number of physical devices available on your machine**

**physical\_devices = tf.config.list\_physical\_devices()**

**print("Number of physical devices:", len(physical\_devices))**

**for device in physical\_devices:**

**print("Device type:", device.device\_type)**

**# Test whether GPU is available**

**gpu\_available = tf.config.list\_physical\_devices('GPU')**

**if gpu\_available:**

**print("GPU is available!")**

**else:**

**print("No GPU available.")**