(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 Apl 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.")
```