1. PROGRAM TO IMPLEMENT LINEAR REGRESSION

ALGORITHM

1. Start
2. Import Necessary Python Libraries
3. Read the CSV file into Pandas DataFrame
4. Extract the height and weight columns from the dataframe
5. Split the data into training and testing data
6. Reshape the training and testing data
7. Create a linear regression model and fit it to training data
8. Predict the weight for the test data using the fitted model
9. Calculate and print the mean squared error using predicted and actual weight
10. Plot the graph
11. Stop

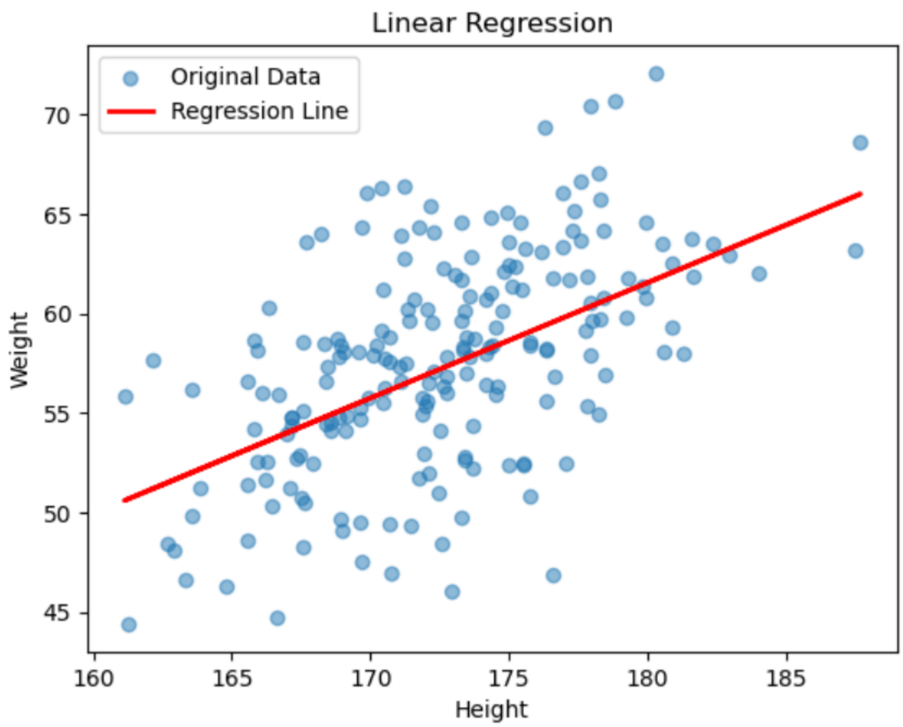
PROGRAM

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

df = pd.read_csv("LinReg_syn_data.csv")
X = df.loc[:, 'height'].values
y = df.loc[:, 'weight'].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=123)
X_train = X_train.reshape(-1,1)
y_train = y_train.reshape(-1, 1)
X_test = X_test.reshape(-1,1)
y_test = y_test.reshape(-1,1)

model = LinearRegression().fit(X_train, y_train)
y_pred = model.predict(X_test)
mse = mean_squared_error(y_true=y_test, y_pred=y_pred)
print("Mean Squared Error : ", round(mse, 3))
plt.scatter(X, y, label="Original Data", alpha=0.5)
plt.plot(X_test, y_pred, color='red', linewidth=2, label="Regression Line")

plt.title("Linear Regression")
plt.xlabel("Height")
plt.ylabel("Weight")
plt.legend()
```

```
plt.show()
```

OUTPUT

Mean Squared Error : 22.218



2. PROGRAM TO IMPLEMENT IMAGE ENHANCEMENT OPERATIONS

ALGORITHM

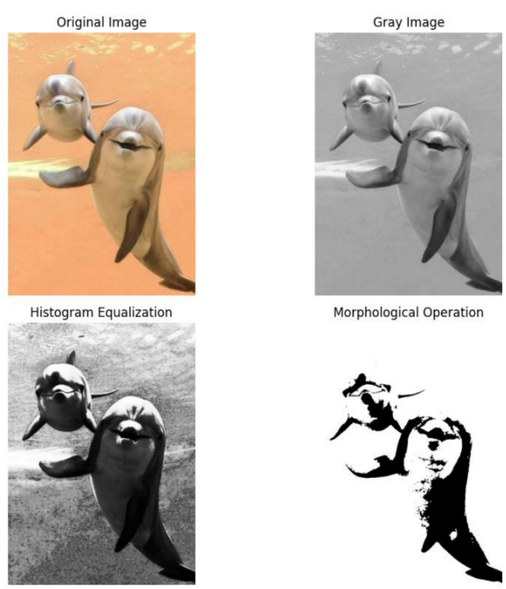
1. Start
2. Import the necessary python libraries
3. Load the image to be enhanced using imhead()
4. Convert the image to grayscale
5. Perform histogram equalization on image using equalizeHist to enhance contrast of image.
6. Binarize the image using threshold function.
7. DEfine 3x3 kernel for morphological operations.
8. Perform morphological opening on image using morphologyEx function to remove noise and small objects.
9. Save grayscale image,histogram equalized and result of morphological operation.
10. Create a figure for plotting with specified size.
11. Plot the original image , grayscale image , histogram equalized image and morphological operation result.
12. Display the plots.
13. Stop

PROGRAM

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

img = cv2.imread("Maggie.jpg")
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
HistEq = cv2.equalizeHist(gray)


binr = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY+cv2.THRESH_OTSU)[1]
kernel = np.ones((3, 3), np.uint8)
opening = cv2.morphologyEx(binr, cv2.MORPH_OPEN, kernel, iterations=1) #
opening the image



cv2.imwrite("GrayImg.jpg", gray)
cv2.imwrite("HistogramEqualization.jpg", HistEq)
cv2.imwrite("MorphologicalOperation.jpg", opening)
```

```
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 8))
plt.subplot(2, 2, 1)
plt.imshow(img)
plt.title("Original Image")
plt.axis('off')

plt.subplot(2, 2, 2)
plt.imshow(gray, cmap='gray')
plt.title("Gray Image")
plt.axis('off')

plt.subplot(2, 2, 3)
plt.imshow(HistEq, cmap='gray')
plt.title("Histogram Equalization")
plt.axis('off')

plt.subplot(2, 2, 4)
plt.imshow(opening, cmap='gray')
plt.title("Morphological Operation")
plt.axis('off')

plt.tight_layout()
plt.show()
```

OUTPUT



3. PROGRAM TO IMPLEMENT Y=X

ALGORITHM

1. Start
2. Import the necessary python libraries.
3. Create arrays X and Y with a single element.
4. Create a sequential model.
5. Add a dense layer to model with 1 neuron named D1, input dimension of 1 and no bias.
6. Compute the model using SGD as optimizer and MSE as loss function.
7. Train the model for 100 epochs using X and Y.
8. Predict output of new input using model.
9. Print [predicted output.
10. Stop.

PROGRAM

```
import tensorflow as tf
import numpy as np
from tensorflow import keras
from tensorflow.keras import layers,models,Sequential
from tensorflow.keras.layers import Dense
x=np.array([3.0])
y=np.array([3.0])
model=Sequential()
model.add(Dense(1,name='D1', input_dim=1, use_bias=False))
model.compile(optimizer='sgd', loss='mse')
model.fit(x,y,epochs=100)
l=model.predict(np.array([12]))
print(l)
```

OUTPUT

```
Epoch 1/100
1/1 [==============================] - 0s 131ms/step - loss: 23.6124
Epoch 2/100
1/1 [==============================] - 0s 8ms/step - loss: 15.8770
Epoch 3/100
1/1 [==============================] - 0s 20ms/step - loss: 10.6757
Epoch 4/100
1/1 [==============================] - 0s 4ms/step - loss: 7.1783
Epoch 5/100
1/1 [==============================] - 0s 1ms/step - loss: 4.8267
```

```
Epoch 6/100
1/1 [==============================] - 0s 2ms/step - loss: 3.2455
Epoch 7/100
1/1 [==============================] - 0s 1ms/step - loss: 2.1823
Epoch 8/100
1/1 [==============================] - 0s 1ms/step - loss: 1.4674
Epoch 9/100
1/1 [==============================] - 0s 3ms/step - loss: 0.9866
Epoch 10/100
1/1 [==============================] - 0s 8ms/step - loss: 0.6634
...
...
...
[[11.999998]]
```

4. PROGRAM TO IMPLEMENT AND GATE

ALGORITHM

1. Start
2. IMport necessary python libraries.
3. Create arrays X and Y representing input output pairs.
4. Initialize a sequential model.
5. Add two dense layers to the model.
6. Compile the model using Adam optimizer and MSE as loss function.
7. TRain the model using input(X) and target output(Y).
8. Predict output of new input using model.
9. Print the predicted output.
10. Stop.

PROGRAM

```
import tensorflow as tf
import numpy as np
from tensorflow import keras
from keras.models import Sequential
#from tensorflow.keras import layers,models,Sequential
from keras.layers import Dense,Activation

x=np.array([[0,0],[0,1],[1,0],[1,1]])
y=np.array([[0],[0],[0],[1]])
model=Sequential()

model.add(Dense(16,input_dim=2,activation='relu',use_bias=False))
model.add(Dense(1, activation='sigmoid',use_bias=False))
model.compile(optimizer='adam',loss='mse')
model.fit(x,y,epochs=350)

l=model.predict([[1,1]])
print(l)
```

OUTPUT

```
Epoch 1/350
1/1 [==============================] - 0s 175ms/step - loss: 0.2566
Epoch 2/350
1/1 [==============================] - 0s 13ms/step - loss: 0.2562
Epoch 3/350
1/1 [==============================] - 0s 5ms/step - loss: 0.2558
Epoch 4/350
```

```
1/1 [==============================] - 0s 4ms/step - loss: 0.2554
Epoch 5/350
1/1 [==============================] - 0s 8ms/step - loss: 0.2549
Epoch 6/350
1/1 [==============================] - 0s 4ms/step - loss: 0.2545
Epoch 7/350
1/1 [==============================] - 0s 4ms/step - loss: 0.2541
Epoch 8/350
1/1 [==============================] - 0s 5ms/step - loss: 0.2537
Epoch 9/350
1/1 [==============================] - 0s 3ms/step - loss: 0.2533
Epoch 10/350
1/1 [==============================] - 0s 3ms/step - loss: 0.2529


[[0.603781]]
```

5.PROGRAM TO IMPLEMENT CIFAR 10 IMAGE CLASSIFICATION

ALGORITHM

1. Start
2. Load and prepare the CIFAR-10 dataset.
3. Define the neural network architecture.
4. DEfine the hyperparameters to test.
5. Create a model for each combination of hidden units and activations.
6. Train the model on the training set and evaluate the model on the testing set.
7. Print the results of all the models.
8. Find the model with the highest test accuracy.
9. Print the details of the model with the highest test accuracy.
10. Display the probability of predictions of each image in a meter graph.
11. Stop.

PROGRAM

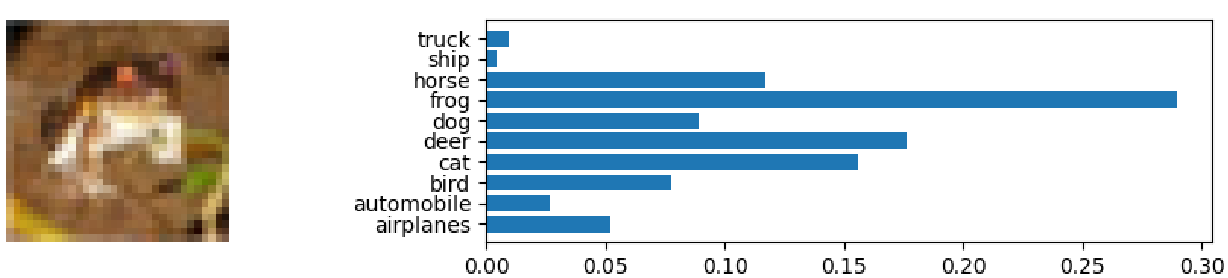
```
import tensorflow as tf
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.utils import to_categorical
import numpy as np
import matplotlib.pyplot as plt
(xtr,ytr),(xte,yte)=cifar10.load_data()

xtr,xte=xtr/255.0,xte/255.0
ytr,yte=to_categorical(ytr), to_categorical(yte)

model=Sequential([
    layers.Flatten(input_shape=(32,32,3)),
    layers.Dense(512,'relu'),
    layers.Dense(256,'relu'),
    layers.Dense(128,'relu'),
    layers.Dense(10,'softmax')
])
model.compile(optimizer='adam',
loss='categorical_crossentropy',metrics=['accuracy'])
history=model.fit(xtr,ytr,epochs=5,batch_size=64,validation_data=(xte,yte))
```

```
_,acc=model.evaluate(xte,yte)
print("Test accuracy:",round(acc*100,4))

sample_img=xtr[:1]
pred=model.predict(sample_img)

class_lab=['airplanes','automobile','bird','cat','deer','dog','frog','horse','ship','truck']
fig,axs=plt.subplots(1,2,figsize=(10,2))
axs[0].imshow(sample_img[0])
axs[0].axis('off')
axs[1].barh(class_lab,pred[0])
plt.tight_layout()
plt.show()
```

OUTPUT

Epoch 1/5
782/782 [==============================] - 28s 35ms/step - loss: 1.8667 - accuracy: 0.3238 - val_loss: 1.7437 - val_accuracy: 0.3773
Epoch 2/5
782/782 [==============================] - 27s 35ms/step - loss: 1.6780 - accuracy: 0.3962 - val_loss: 1.6917 - val_accuracy: 0.3923
Epoch 3/5
782/782 [==============================] - 27s 35ms/step - loss: 1.5900 - accuracy: 0.4282 - val_loss: 1.5692 - val_accuracy: 0.4400
Epoch 4/5
782/782 [==============================] - 26s 33ms/step - loss: 1.5274 - accuracy: 0.4507 - val_loss: 1.5329 - val_accuracy: 0.4539
Epoch 5/5
782/782 [==============================] - 26s 34ms/step - loss: 1.4897 - accuracy: 0.4686 - val_loss: 1.5332 - val_accuracy: 0.4587

313/313 [==============================] - 3s 9ms/step - loss: 1.5332 - accuracy: 0.4587
Test accuracy: 45.87



6. CIFAR 10 IMAGE CLASSIFICATION USING DIFFERENT WEIGHT INITIALIZATION AND REGULARIZATION TECHNIQUES

ALGORITHM

1. Start
2. Load the CIFAR-10 dataset.
3. Preprocess the dataset and normalize it.
4. Create a baseline model with three hidden layers.
5. Train the baseline model using the adam optimizer or SGD optimizer.
6. Evaluate the baseline model on the test set.
7. Repeat steps 3-5 using Xavier and Kaiming weight initialization.
8. Repeat steps 3-5 using dropout.
9. Repeat steps 3-5 using an L2 kernel regularization.
10. Compare the results of the different models.
11. Stop.

PROGRAM

```
import tensorflow as tf
from tensorflow.keras import layers,models,initializers
from tensorflow.keras.models import Sequential
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.utils import to_categorical
import matplotlib.pyplot as plt
(xtr,ytr),(xte,yte)=cifar10.load_data()
xtr,xte=xtr/255.0,xte/255.0
ytr,yte=to_categorical(ytr),to_categorical(yte)
def model_create(ini,drop=0.0,l2=None):
  model=Sequential([
     layers.Flatten(input_shape=(32,32,3)),
     layers.Dense(512,kernel_initializer=ini,kernel_regularizer=l2, activation='relu'),
     layers.Dense(256,kernel_initializer=ini,kernel_regularizer=l2, activation='relu'),
     layers.Dense(128,kernel_initializer=ini,kernel_regularizer=l2, activation='relu'),
     layers.Dense(64,kernel_initializer=ini,kernel_regularizer=l2, activation='relu'),
     layers.Dense(32,kernel_initializer=ini,kernel_regularizer=l2, activation='relu'),
     layers.Dense(10,activation='softmax')
  ])
  return model
x_ini=initializers.glorot_normal()
```

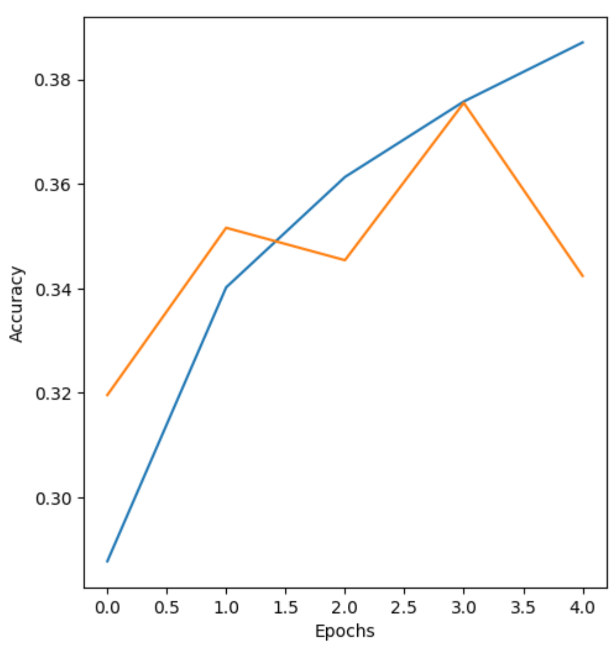
```
k_ini=initializers.he_normal()
x_model=model_create(x_ini,0.3,tf.keras.regularizers.l2(0.001))
k_model=model_create(k_ini,0.3,tf.keras.regularizers.l2(0.001))
x_model.compile(optimizer='adam',loss='categorical_crossentropy',metrics=['accura
cy'])
k_model.compile(optimizer='adam',loss='categorical_crossentropy',metrics=['accura
cy'])
x_history=x_model.fit(xtr,ytr,epochs=5,validation_data=(xte,yte))
k_history=k_model.fit(xtr,ytr,epochs=5,validation_data=(xte,yte))
_,acc=x_model.evaluate(xte,yte)
print("Test accuracy of xavier initializer:",round(acc*100,4))
_,acc=k_model.evaluate(xte,yte)
print("Test accuracy of Kaiming Initializer:",round(acc*100,4))
plt.figure(figsize=(12,6))
plt.subplot(1,2,1)
plt.plot(x_history.history['accuracy'],label='Xavier(train)')
plt.plot(x_history.history['val_accuracy'],label='Xavier(validation)')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.subplot(1,2,2)
plt.plot(k_history.history['accuracy'],label='Kaiming(train)')
plt.plot(k_history.history['val_accuracy'],label='Kaiming(train)')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

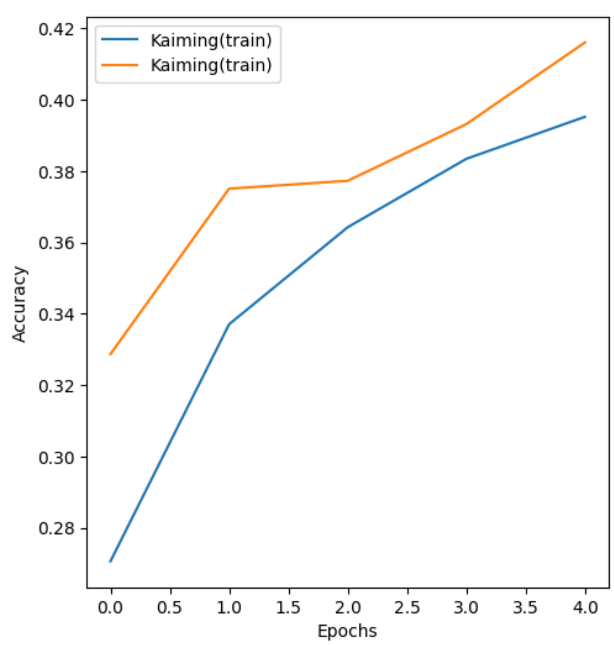
OUTPUT

```
Epoch 1/5
1563/1563 [==============================] - 20s 13ms/step - loss: 2.0428 -
accuracy: 0.2878 - val_loss: 1.9429 - val_accuracy: 0.3196
Epoch 2/5
1563/1563 [==============================] - 20s 13ms/step - loss: 1.9024 -
accuracy: 0.3402 - val_loss: 1.8539 - val_accuracy: 0.3516
Epoch 3/5
1563/1563 [==============================] - 21s 14ms/step - loss: 1.8568 -
accuracy: 0.3613 - val_loss: 1.8842 - val_accuracy: 0.3454
Epoch 4/5
1563/1563 [==============================] - 20s 13ms/step - loss: 1.8203 -
accuracy: 0.3758 - val_loss: 1.8183 - val_accuracy: 0.3755
```

Epoch 5/5
1563/1563 [==============================] - 20s 13ms/step - loss: 1.7931 - accuracy: 0.3871 - val_loss: 1.9021 - val_accuracy: 0.3424
Epoch 1/5
1563/1563 [==============================] - 30s 18ms/step - loss: 2.3737 - accuracy: 0.2706 - val_loss: 2.0077 - val_accuracy: 0.3287
Epoch 2/5
1563/1563 [==============================] - 27s 17ms/step - loss: 1.9410 - accuracy: 0.3371 - val_loss: 1.8325 - val_accuracy: 0.3751
Epoch 3/5
1563/1563 [==============================] - 24s 15ms/step - loss: 1.8540 - accuracy: 0.3643 - val_loss: 1.8061 - val_accuracy: 0.3773
Epoch 4/5
1563/1563 [==============================] - 22s 14ms/step - loss: 1.8010 - accuracy: 0.3835 - val_loss: 1.7863 - val_accuracy: 0.3932
Epoch 5/5
1563/1563 [==============================] - 29s 18ms/step - loss: 1.7728 - accuracy: 0.3952 - val_loss: 1.7315 - val_accuracy: 0.4161

313/313 [==============================] - 1s 3ms/step - loss: 1.9021 - accuracy: 0.3424
Test accuracy of xavier initializer: 34.24
313/313 [==============================] - 1s 4ms/step - loss: 1.7315 - accuracy: 0.4161
Test accuracy of Kaiming Initializer: 41.61





7. PROGRAM TO IMPLEMENT A CNN FOR DIGIT CLASSIFICATION USING MNIST DATASET

ALGORITHM

1. Start
2. Import the necessary python libraries and MNIST dataset.
3. Load the MNIST dataset separating it into training and testing images and labels.
4. Normalize the pixel values of training and testing images and labels.
5. Reshape the training and testing images.
6. Convert the training and testing labels to one hot encoding using to_categorical function.
7. Define a sequential model using keras, adding convolutional layers, pooling layers, flatten and dense layers.
8. Compile the model using adam optimizer and categorical_crossentropy as loss function.
9. Train the model using training images and labels.
10. Evaluate the trained model using testing images and labels and calculate the test accuracy and print it.
11. Stop

PROGRAM

```
import tensorflow as tf
from tensorflow.keras.datasets import mnist
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import Sequential
(train_images,train_labels),(test_images,test_labels)=mnist.load_data()
train_images,test_images=train_images/255.0,test_images/255.0
train_images,test_images=train_images.reshape(-1,28,28,1),test_images.reshape(-
1,28,28,1)
train_labels,test_labels=to_categorical(train_labels),to_categorical(test_labels)
model=Sequential([tf.keras.layers.Reshape((28,28,1)),
                  tf.keras.layers.Conv2D(32,3,activation='relu'),
                  tf.keras.layers.MaxPooling2D(),
                  tf.keras.layers.Conv2D(64,3,activation='relu'),
                  tf.keras.layers.MaxPooling2D(),
                  tf.keras.layers.Conv2D(64,3,activation='relu'),
                  tf.keras.layers.Flatten(),
                  tf.keras.layers.Dense(64,activation='relu'),
```

```
                        tf.keras.layers.Dense(10,activation='softmax')
                       ])
model.compile(optimizer='adam',loss='categorical_crossentropy',metrics=['accuracy
'])
model.fit(train_images,train_labels,epochs=5,batch_size=64,validation_split=0.2)
_,test_acc=model.evaluate(test_images,test_labels)
print(f"\nTest Accuracy : {round(test_acc * 100, 4)}%")
```

OUTPUT

```
Epoch 1/5
750/750 [==============================] - 55s 72ms/step - loss: 0.2121 -
accuracy: 0.9359 - val_loss: 0.0741 - val_accuracy: 0.9779
Epoch 2/5
750/750 [==============================] - 52s 69ms/step - loss: 0.0584 -
accuracy: 0.9815 - val_loss: 0.0579 - val_accuracy: 0.9832
Epoch 3/5
750/750 [==============================] - 52s 69ms/step - loss: 0.0427 -
accuracy: 0.9866 - val_loss: 0.0441 - val_accuracy: 0.9883
Epoch 4/5
750/750 [==============================] - 51s 68ms/step - loss: 0.0322 -
accuracy: 0.9899 - val_loss: 0.0402 - val_accuracy: 0.9878
Epoch 5/5
750/750 [==============================] - 49s 65ms/step - loss: 0.0248 -
accuracy: 0.9923 - val_loss: 0.0382 - val_accuracy: 0.9886
313/313 [==============================] - 5s 16ms/step - loss: 0.0316 -
accuracy: 0.9900

Test Accuracy : 99.0%
```

8. DIGIT CLASSIFICATION USING VGGnet-19 FOR MNIST

ALGORITHM

1. Start
2. Import python libraries, VGG19 model and MNIST dataset.
3. Load MNIST dataset into training and testing set and limit to 5000 and 500 samples.
4. Normalize pixel values in training and testing set to range [0,1].
5. Convert labels to one hot encoding.
6. Load VGG19 pre-trained model excluding top layers.
7. Create a sequential model and compile the model using adam optimizer and categorical_crossentropy loss.
8. Train the model using training loss.
9. Convert grayscale images to RGB and resize them.
10. Load VGG19 model for fine tuning.
11. Create a new model with custom classification layers on top of the pre-trained VGG19 base model.
12. Compile and train the fine tuned model.
13. Evaluate the model on test data.
14. Plot the training and validation accuracy.
15. Stop.

PROGRAM

```
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras.applications import VGG19
from tensorflow.keras.datasets import mnist
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import Dense, Flatten, Input


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

X_train = X_train[:1500]
y_train = y_train[:1500]
```

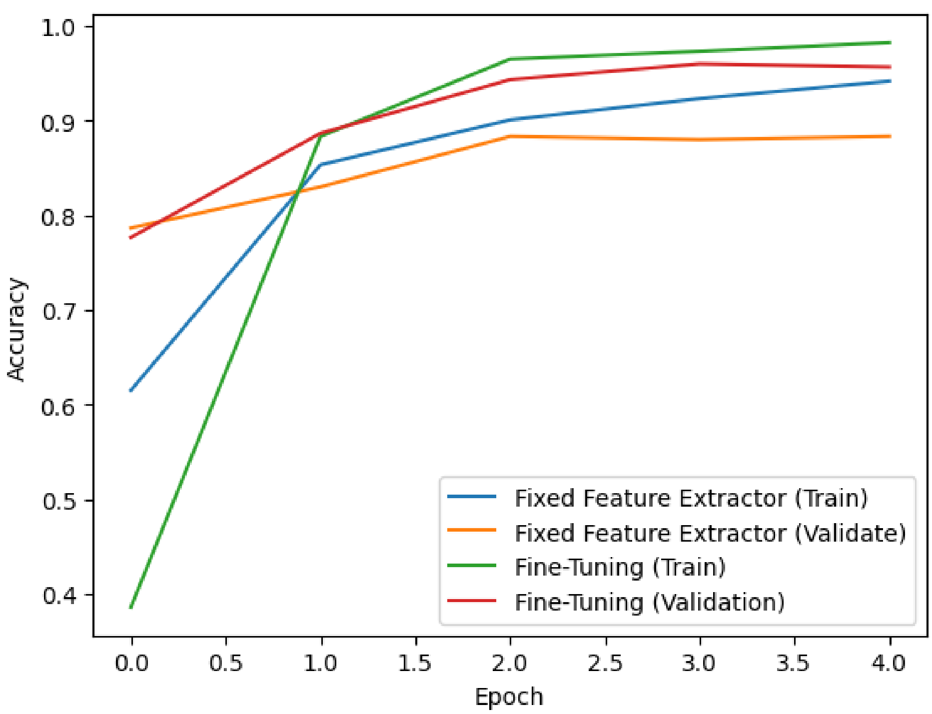
```
X_test = X_test[:500]
y_test = y_test[:500]

X_train_froz, X_test_froz = X_train / 255.0, X_test / 255.0
y_train_froz, y_test_froz = to_categorical(y_train), to_categorical(y_test)

model = Sequential([
   Flatten(input_shape=X_train.shape[1:]),
   Dense(256, activation='relu'),
   Dense(10, activation='softmax')
])
model.compile(optimizer='adam',
loss='categorical_crossentropy',metrics=['accuracy'])
hist = model.fit(X_train_froz, y_train_froz, epochs=5,
batch_size=64,validation_split=0.2)

X_train_rgb = tf.image.grayscale_to_rgb(tf.expand_dims(X_train, axis=-1))
X_test_rgb = tf.image.grayscale_to_rgb(tf.expand_dims(X_test, axis=-1))
X_train_resized = tf.image.resize(X_train_rgb, (224, 224))
X_test_resized = tf.image.resize(X_test_rgb, (224, 224))

X_train_resized = X_train_resized / 255.0
X_test_resized = X_test_resized / 255.0

y_train_rgb = to_categorical(y_train, num_classes=10)
y_test_rgb = to_categorical(y_test, num_classes=10)


base_model = VGG19(weights='imagenet', include_top=False, input_shape=(224,
224, 3))

x = Flatten()(base_model.output)
x = Dense(256, activation='relu')(x)
output = Dense(10, activation='softmax')(x)
model_2 = Model(inputs=base_model.input, outputs=output)

for layer in base_model.layers:
  layer.trainable = False
```

```
model_2.compile(optimizer='adam',
loss='categorical_crossentropy',metrics=['accuracy'])

histo = model_2.fit(X_train_resized, y_train_rgb, epochs=5,
batch_size=64,validation_split=0.2)

loss, accuracy = model_2.evaluate(X_test_resized, y_test_rgb)
print("Test accuracy:", accuracy)

plt.plot(hist.history['accuracy'], label='Fixed Feature Extractor (Train)')
plt.plot(hist.history['val_accuracy'], label='Fixed Feature Extractor (Validate)')

plt.plot(histo.history['accuracy'], label='Fine-Tuning (Train)')
plt.plot(histo.history['val_accuracy'], label='Fine-Tuning (Validation)')

plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

OUTPUT

```
Epoch 1/5
19/19 [==============================] - 1s 13ms/step - loss: 1.5030 -
accuracy: 0.6150 - val_loss: 0.8416 - val_accuracy: 0.7867
Epoch 2/5
19/19 [==============================] - 0s 5ms/step - loss: 0.6141 - accuracy:
0.8533 - val_loss: 0.5596 - val_accuracy: 0.8300
Epoch 3/5
19/19 [==============================] - 0s 6ms/step - loss: 0.4023 - accuracy:
0.9008 - val_loss: 0.4355 - val_accuracy: 0.8833
Epoch 4/5
19/19 [==============================] - 0s 6ms/step - loss: 0.3027 - accuracy:
0.9233 - val_loss: 0.4455 - val_accuracy: 0.8800
Epoch 5/5
19/19 [==============================] - 0s 5ms/step - loss: 0.2449 - accuracy:
0.9417 - val_loss: 0.4104 - val_accuracy: 0.8833

Epoch 1/5
```

19/19 [==============================] - 9s 434ms/step - loss: 4.0634 - accuracy: 0.3858 - val_loss: 0.9605 - val_accuracy: 0.7767
Epoch 2/5
19/19 [==============================] - 6s 333ms/step - loss: 0.4747 - accuracy: 0.8833 - val_loss: 0.4567 - val_accuracy: 0.8867
Epoch 3/5
19/19 [==============================] - 6s 344ms/step - loss: 0.1940 - accuracy: 0.9650 - val_loss: 0.2778 - val_accuracy: 0.9433
Epoch 4/5
19/19 [==============================] - 6s 340ms/step - loss: 0.1318 - accuracy: 0.9733 - val_loss: 0.2372 - val_accuracy: 0.9600
Epoch 5/5
19/19 [==============================] - 6s 338ms/step - loss: 0.0946 - accuracy: 0.9825 - val_loss: 0.2172 - val_accuracy: 0.9567
16/16 [==============================] - 2s 139ms/step - loss: 0.1532 - accuracy: 0.9620
Test accuracy: 0.9620000123977661



9. PROGRAM TO IMPLEMENT AN RNN FOR REVIEW CLASSIFICATION ON IMDB DATASET

ALGORITHM

1. Start
2. Import necessary python libraries and IMDB dataset.
3. Assign num_words as 1000 and max_length as 200.
4. Split IMDB dataset into training set and testing set.
5. Pad training set and testing set to have maximum length 200.
6. Initialize a sequential model and add embedding,LSTM and dense layer to model.
7. Compile the model with adam optimizer, binary_crossentropy as loss function.
8. Train the model with the training set and evaluate the model with the testing set.
9. Calculate accuracy and print it.
10. Reshape the testing set.
11. Predict sentiment with the test set.
12. If the model predicts the sentiment score as 1, then print 'Positive review' else if 0 then print 'Negative review'.
13. Check the actual sentiment score of the 8th review on the testing set and print.
14. Stop.

PROGRAM

```
import numpy as np
import tensorflow as tf
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
num_words=10000
max_length=200
(xtr,ytr),(xte,yte)=imdb.load_data(num_words=num_words)
xtr,xte=pad_sequences(xtr,maxlen=max_length),pad_sequences(xte,maxlen=max_l
ength)
model=Sequential([
   Embedding(input_dim=num_words,output_dim=128,input_length=max_length),
   LSTM(128),
   Dense(1,activation=sigmoid)
])
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

```
model.fit(xtr,ytr,validation_split=0.2,epochs=5,batch_size=64)
loss,acc=model.evaluate(xte,yte)
print("Test accuracy:",round(acc*100,4))
test_seq=np.reshape(xte[7],(1,-1))
pred=model.predict(test_seq)[0]
if int(pred[0])==1:
  print('Positive Review')
else:
  print('Negative Review')
yte[7]
```

OUTPUT

```
Epoch 1/5
313/313 [==============================] - 34s 84ms/step - loss: 0.4233 - accuracy: 0.7994 - val_loss: 0.3109 - val_accuracy: 0.8664
Epoch 2/5
313/313 [==============================] - 11s 35ms/step - loss: 0.2421 - accuracy: 0.9068 - val_loss: 0.3698 - val_accuracy: 0.8630
Epoch 3/5
313/313 [==============================] - 8s 25ms/step - loss: 0.1669 - accuracy: 0.9378 - val_loss: 0.3609 - val_accuracy: 0.8724
Epoch 4/5
313/313 [==============================] - 8s 24ms/step - loss: 0.1365 - accuracy: 0.9496 - val_loss: 0.4191 - val_accuracy: 0.8176
Epoch 5/5
313/313 [==============================] - 5s 17ms/step - loss: 0.1143 - accuracy: 0.9580 - val_loss: 0.4396 - val_accuracy: 0.8372

782/782 [==============================] - 4s 5ms/step - loss: 0.4435 - accuracy: 0.8329
Test accuracy: 83.288

1/1 [==============================] - 0s 342ms/step
Negative Review

0
```

10. COMPARATIVE ANALYSIS OF LSTM AND GRU FOR SENTIMENT ANALYSIS AND IMDB DATASET.

ALGORITHM

1. Start.
2. Import necessary python libraries and IMDB dataset.
3. Assign num_words as 10000 and max_length as 200.
4. Split the dataset into testing and training sets with sequences padded to max_length.
5. Create an LSTM model and add LSTM layers.
6. Compile the model with binary_crossentropy loss and adam optimizer and accuracy as metrics.
7. Create a GRU model and add GRU layers.
8. Compile the model with binary_crossentropy loss and adam optimizer and accuracy as metrics.
9. Fit LSTM and GRU model on train set.
10. Calculate accuracy of both models on test and print the values.
11. Plot the training history of both models.
12. Display the plots.
13. Stop.

PROGRAM

```
import numpy as np
import tensorflow as tf
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,GRU
import matplotlib.pyplot as plt

num_words=10000
max_length=200

(xtr,ytr),(xte,yte)=imdb.load_data(num_words=num_words)
xtr,xte=pad_sequences(xtr,maxlen=max_length),pad_sequences(xte,maxlen=max_l
ength)

l_model=Sequential([
```

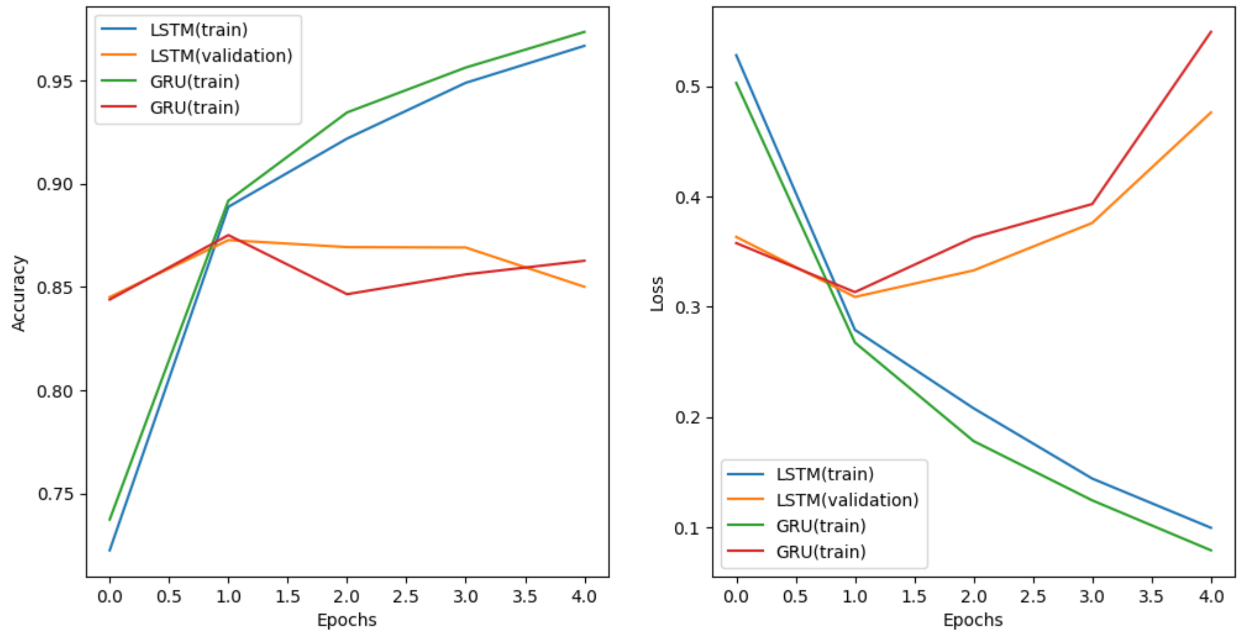
```
    Embedding(input_dim=num_words,output_dim=128,input_length=max_length),
    LSTM(128),
    Dense(1,activation='sigmoid')
])

l_model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
l_history= l_model.fit(xtr,ytr,validation_split=0.2,epochs=5,batch_size=64)

loss,acc=l_model.evaluate(xte,yte)
print("Test accuracy:",round(acc*100,4))

g_model=Sequential([
    Embedding(input_dim=num_words,output_dim=128,input_length=max_length),
    GRU(128),
    Dense(1,activation='sigmoid')
])

g_model.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])
g_history=g_model.fit(xtr,ytr,validation_split=0.2,epochs=5,batch_size=64)

loss,acc=g_model.evaluate(xte,yte)
print("Test accuracy:",round(acc*100,4))

plt.figure(figsize=(12,6))
plt.subplot(1,2,1)
plt.plot(l_history.history['accuracy'],label='LSTM(train)')
plt.plot(l_history.history['val_accuracy'],label='LSTM(validation)')
plt.plot(g_history.history['accuracy'],label='GRU(train)')
plt.plot(g_history.history['val_accuracy'],label='GRU(train)')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.subplot(1,2,2)
plt.plot(l_history.history['loss'],label='LSTM(train)')
plt.plot(l_history.history['val_loss'],label='LSTM(validation)')
plt.plot(g_history.history['loss'],label='GRU(train)')
plt.plot(g_history.history['val_loss'],label='GRU(train)')
plt.xlabel('Epochs')
plt.ylabel('Loss')
```

```
plt.legend()
plt.show()
```

OUTPUT

```
Epoch 1/5
313/313 [==============================] - 36s 91ms/step - loss: 0.4276 - accuracy: 0.8009 - val_loss: 0.3477 - val_accuracy: 0.8658
Epoch 2/5
313/313 [==============================] - 11s 36ms/step - loss: 0.2385 - accuracy: 0.9087 - val_loss: 0.3164 - val_accuracy: 0.8726
Epoch 3/5
313/313 [==============================] - 7s 24ms/step - loss: 0.1745 - accuracy: 0.9351 - val_loss: 0.3558 - val_accuracy: 0.8640
Epoch 4/5
313/313 [==============================] - 8s 25ms/step - loss: 0.1287 - accuracy: 0.9554 - val_loss: 0.3972 - val_accuracy: 0.8568
Epoch 5/5
313/313 [==============================] - 5s 17ms/step - loss: 0.0855 - accuracy: 0.9712 - val_loss: 0.6033 - val_accuracy: 0.8408
output
782/782 [==============================] - 4s 5ms/step - loss: 0.6279 - accuracy: 0.8338
Test accuracy: 83.384
Epoch 1/5
313/313 [==============================] - 31s 89ms/step - loss: 0.4668 - accuracy: 0.7686 - val_loss: 0.3458 - val_accuracy: 0.8588
Epoch 2/5
313/313 [==============================] - 12s 38ms/step - loss: 0.2465 - accuracy: 0.9026 - val_loss: 0.3038 - val_accuracy: 0.8714
Epoch 3/5
313/313 [==============================] - 8s 27ms/step - loss: 0.1607 - accuracy: 0.9404 - val_loss: 0.3611 - val_accuracy: 0.8470
Epoch 4/5
313/313 [==============================] - 8s 25ms/step - loss: 0.1019 - accuracy: 0.9636 - val_loss: 0.3745 - val_accuracy: 0.8636
Epoch 5/5
313/313 [==============================] - 5s 17ms/step - loss: 0.0620 - accuracy: 0.9796 - val_loss: 0.4538 - val_accuracy: 0.8540
782/782 [==============================] - 4s 5ms/step - loss: 0.4660 - accuracy: 0.8498
```

Test accuracy: 84.976



11. PROGRAM TO IMPLEMENT TIME SERIES FORECASTING FOR NIFTY-50 DATASET

ALGORITHM

1. Start.
2. Import necessary python libraries.
3. Read CSV file ‘nifty.csv’ as a dataframe.
4. Normalize the numerical columns ‘Open’,’High’,’Low’,’Close’ using the MinMax Scaler.
5. Split the data into the training and testing set.
6. Define a sequential model with 3 dense layers , 2 hidden layers using ReLu activation and output layers using linear activation.
7. Compile the model using MSE loss function and adam optimizer.
8. Train the model on training data.
9. Predict the closing prices using the trained model.
10. Visualize actual and predicted closing prices with a line plot.
11. Calculate MAW between actual and predicted closing prices and print.
12. Stop.

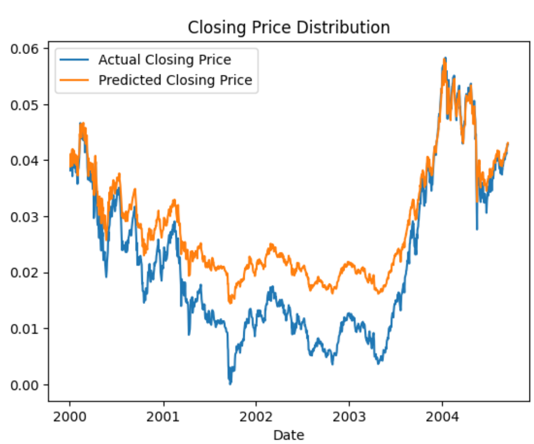
PROGRAM

```
import pandas as pd
import tensorflow as tf
from tensorflow.keras.layers import Dense
from tensorflow.keras.models import Sequential
from sklearn.metrics import mean_absolute_error
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt

data = pd.read_csv("nifty.csv", index_col="Date", parse_dates=True)
scaler = MinMaxScaler()
data[['Open', 'High', 'Low', 'Close']] = scaler.fit_transform(data[['Open', 'High', 'Low',
'Close']] )
train_data, test_data = train_test_split(data, test_size=0.2, shuffle=False)
model =Sequential([
   Dense(64, activation='relu'),
   Dense(32, activation='relu'),
   Dense(1, activation='linear')
```

```
])
model.compile(loss='mean_squared_error', optimizer='adam')
model.fit(train_data[['Open', 'High', 'Low']], train_data['Close'], epochs=100)
predicted_closing_prices = model.predict(test_data[['Open', 'High', 'Low']])

plt.plot(test_data.index, test_data['Close'], label='Actual Closing Price')
plt.plot(test_data.index, predicted_closing_prices, label='Predicted Closing Price')
plt.title("Closing Price Distribution")
plt.xlabel("Date")
plt.legend()
plt.show()

mae = mean_absolute_error(test_data['Close'], predicted_closing_prices)
print(f"\n\nMean Absolute Error: {round(mae, 5)}")
```

OUTPUT

```
Epoch 1/100
148/148 [==============================] - 7s 4ms/step - loss: 0.0076
Epoch 2/100
148/148 [==============================] - 0s 3ms/step - loss: 9.5517e-06
Epoch 3/100
148/148 [==============================] - 0s 3ms/step - loss: 9.1067e-06
Epoch 4/100
148/148 [==============================] - 0s 3ms/step - loss: 8.8282e-06
Epoch 5/100
148/148 [==============================] - 0s 3ms/step - loss: 8.6149e-06
```



Mean Absolute Error: 0.00662

12. PROGRAM TO IMPLEMENT ENGLISH TO HINDI MACHINE TRANSLATION

ALGORITHM

1. Start.
2. Import the necessary python libraries.
3. Read CSV file and pre-process data.
4. Do text preprocessing like lowercasing.
5. Vectorize the data.
6. Create zero matrices for encoder and decoder data.
7. Populate the matrices with one hot encoding.
8. Set up the model architecture.
9. Compile and train the model.
10. Predict text using the trained model and print.
11. Stop.

PROGRAM

```
import numpy as np
import pandas as pd
import tensorflow as tf
from tensorflow.keras.models import Model, load_model
from tensorflow.keras.layers import Input, LSTM, Embedding, Dense
from sklearn.utils import shuffle
from sklearn.model_selection import train_test_split
import re
import string
from string import digits
import nltk
from nltk.corpus import stopwords
from nltk import word_tokenize
import matplotlib.pyplot as plt
import os

# Reading CSV
lines = pd.read_csv("MyDrive/MyDrive/Hindi_English_Truncated_Corpus.csv")

# Selecting only a specific source
```

```
lines=lines[lines['source']=='ted']

# Deleting duplicate values
lines.drop_duplicates(inplace=True)

# Selecting 25000 out of the 39881 values
lines=lines.sample(n=25000, random_state=101)

# Lowercase all characters
lines['english_sentence']=lines['english_sentence'].apply(lambda x: x.lower())
lines['hindi_sentence']=lines['hindi_sentence'].apply(lambda x: x.lower())

# Remove quotes
lines['english_sentence']=lines['english_sentence'].apply(lambda x: re.sub("'", '', x))
lines['hindi_sentence']=lines['hindi_sentence'].apply(lambda x: re.sub("'", '', x))

# Set of all special characters
exclude = set(string.punctuation)

# Remove all the special characters
lines['english_sentence']=lines['english_sentence'].apply(lambda x: ''.join(ch for ch
in x if ch not in exclude))
lines['hindi_sentence']=lines['hindi_sentence'].apply(lambda x: ''.join(ch for ch in x if
ch not in exclude))

# Remove all numbers from text
remove_digits = str.maketrans('', '', digits)
lines['english_sentence']=lines['english_sentence'].apply(lambda x:
x.translate(remove_digits))
lines['hindi_sentence']=lines['hindi_sentence'].apply(lambda x:
x.translate(remove_digits))
lines['hindi_sentence'] = lines['hindi_sentence'].apply(lambda x:
re.sub("[२३०८१५७९४६]", "", x))

# Remove extra spaces
lines['english_sentence']=lines['english_sentence'].apply(lambda x: x.strip())
lines['hindi_sentence']=lines['hindi_sentence'].apply(lambda x: x.strip())
lines['english_sentence']=lines['english_sentence'].apply(lambda x: re.sub(" +", " ",
x))
lines['hindi_sentence']=lines['hindi_sentence'].apply(lambda x: re.sub(" +", " ", x))
```

```
# Replacing English Alphabets that may occur in Hindi Text
lines['hindi_sentence']=lines['hindi_sentence'].apply(lambda x: re.sub("[a-zA-Z]", " ",
x))

# Vectorize the data

input_texts = []
target_texts = []

input_characters = set()
target_characters = set()

# Isolating individual characters of English and Hindi respectively
for line in lines['english_sentence']:
    input_texts.append(line)
    for char in line:
        if re.findall("[a-zA-Z]", char) or char == ' ':
            if char not in input_characters:
                input_characters.add(char)

for line in lines['hindi_sentence']:

    target_text = '%' + line + '$'

    target_texts.append(line)
    for char in target_text:
        if char not in target_characters:
            target_characters.add(char)

# Sorting the lists
input_characters = sorted(list(input_characters))
target_characters = sorted(list(target_characters))

num_encoder_tokens = len(input_characters)
num_decoder_tokens = len(target_characters)

max_encoder_seq_length = max([len(txt) for txt in input_texts]) # Finding largest
sequence in English and setting it as max length
```

```
max_decoder_seq_length = max([len(txt) for txt in target_texts]) # Finding largest
sequence in Hindi and setting it as max length

# Indexing each token using enumerate
input_token_index = dict([(char, i) for i, char in enumerate(input_characters)])
target_token_index = dict([(char, i) for i, char in enumerate(target_characters)])

# Creating Zero Matrix with max length sizes
encoder_input_data = np.zeros((len(input_texts), max_encoder_seq_length,
num_encoder_tokens), dtype='float32')
decoder_input_data = np.zeros((len(input_texts), max_decoder_seq_length,
num_decoder_tokens), dtype='float32')
decoder_target_data = np.zeros((len(input_texts), max_decoder_seq_length,
num_decoder_tokens), dtype='float32')

for i, (input_text, target_text) in enumerate(zip(input_texts, target_texts)):
    for t, char in enumerate(input_text):
        if re.findall("[a-zA-Z]", char) or char == ' ':
            encoder_input_data[i,t, input_token_index[char]] = 1
    encoder_input_data[i,t+1:, input_token_index[' ']] = 1

    for t, char in enumerate(target_text):
        decoder_input_data[i, t, target_token_index[char]] = 1

        if t > 0:
            decoder_target_data[i, t-1, target_token_index[char]] = 1

    decoder_input_data[i,t+1:, target_token_index[' ']] = 1
    decoder_target_data[i, t:, target_token_index[' ']] = 1

# Initializing Hyperparameters
batch_size = 128 # Batch size for training
epochs = 50 # Number of epochs to train for
latent_dim = 256

# Set up the decoder, using `encoder_states` as initial state.
decoder_inputs = Input(shape=(None, num_decoder_tokens))

decoder_lstm = LSTM(latent_dim, return_sequences=True, return_state=True)
```

```
decoder_outputs, _, _ = decoder_lstm(decoder_inputs,
initial_state=encoder_states)

decoder_dense = Dense(num_decoder_tokens, activation='softmax')
decoder_outputs = decoder_dense(decoder_outputs)

# Define the model that will turn `encoder_input_data` & `decoder_input_data` into
`decoder_target_data`
model = Model([encoder_inputs, decoder_inputs], decoder_outputs)

# Run training
model.compile(optimizer='rmsprop', loss='categorical_crossentropy',
metrics=['accuracy'])

# Fitting data
model.fit([encoder_input_data, decoder_input_data], decoder_target_data,
          batch_size=batch_size,
          epochs=epochs,
          validation_split=0.2)

# Predicting Value
text_to_predict="So there is some sort of justice"
prediction=model.predict(text_to_predict)
print("The predicted text is: ", prediction)
```

OUTPUT

The predicted text is: तो वहाँ न्याय है