ST. JOSEPH'S COLLEGE OF ENGINEERING AND TECHNOLOGY, PALAI

(An ISO 9001:2015 Certified College Affiliated to APJ Abdul Kalam Technological University)

INSTRUCTION MANUAL

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

AIL 411 DEEP LEARNING LAB

Seventh Semester

2019 Regulations

ST. JOSEPH'S COLLEGE OF ENGINEERING AND TECHNOLOGY, PALAI

(An ISO 9001:2015 Certified College Affiliated to APJ Abdul Kalam Technological University)

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

INSTRUCTION MANUAL

AIL 411 DEEP LEARNING LAB

Prepared by

Ms. Lakshmi G

Assistant Professor, AD

Approved by

Dr. Deepa V.

HOD, AD

Verified by

Ms. Jabin Mathew

Assistant Professor, AD

LIST OF EXPERIMENTS

Prepared by: Verified by: Approved by:

EXPERIMENT - 1

LINEAR REGRESSION

AIM:
Implement Simple Linear Regression with Synthetic Data

ALGORITHM

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

PROGRAM

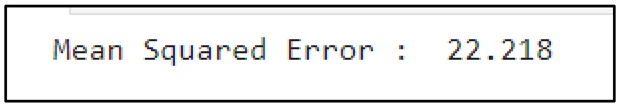
```python
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

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()
```

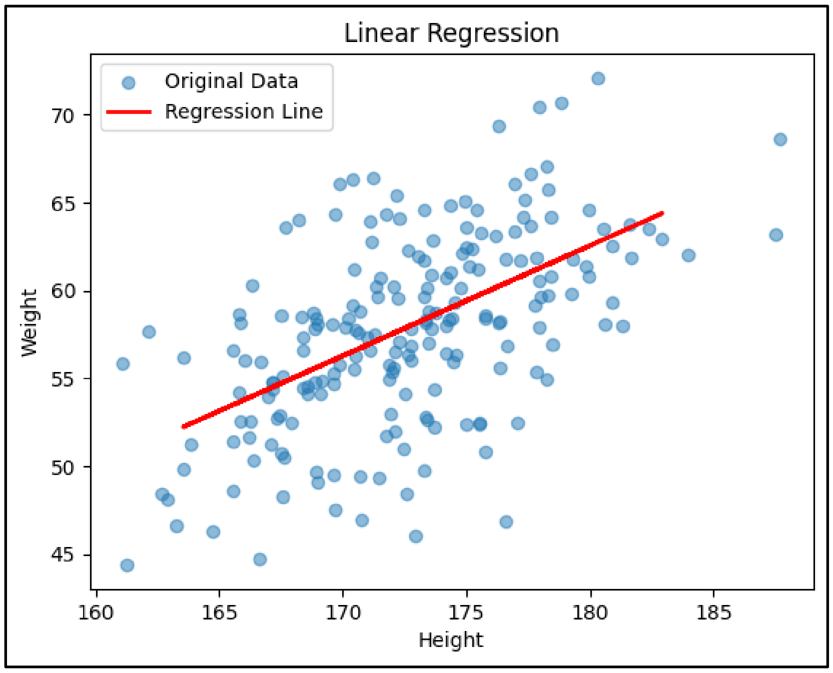
RESULT

The program for linear regression has been completed and the output is verified.

OUTPUT



```
Mean Squared Error :  22.218
```



EXPERIMENT - 2

IMAGE ENHANCEMENT

AIM:
Implement basic image enhancement operations such as histogram equalization, and morphological operations.

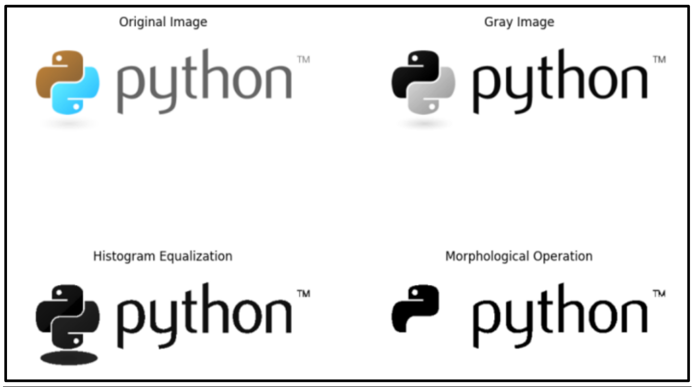
ALGORITHM
```
1: Start
2: Import the necessary libraries and read the image file.
3: Convert the image to grayscale.
4: Apply histogram equalization to the grayscale image.
5: Apply Otsu's thresholding to the grayscale image to binarize it.
6: Define a kernel for morphological opening.
7: Apply morphological opening to the binary image.
8: Display the original and enhanced images.
9: Stop
```

PROGRAM
```
import cv2
import numpy as np
import matplotlib.pyplot as plt
img = cv2.imread("py.png")

gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

HistEq = cv2.equalizeHist(gray)
binr = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY+cv2.THRESH_OTSU)[1] # binarize the
image

kernel = np.ones((3, 3), np.uint8) # define the kernel

opening = cv2.morphologyEx(binr, cv2.MORPH_OPEN, kernel, iterations=1) # opening
cv2.imwrite("GrayImg.jpg", gray)
cv2.imwrite("HistogramEqualization.jpg", HistEq)
cv2.imwrite("MorphologicalOperation.jpg", opening)
plt.figure(figsize=(10, 8))  # Adjust the figure size if needed
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()  # Adjust layout spacing
plt.show()
```

RESULT

The program for basic image enhancement operations has been completed and the output is verified.

OUTPUT



EXPERIMENT - 3

CIFAR-10 CLASSIFICATION

AIM:

Implement a Feedforward neural network with three hidden layers for classification on the CIFAR-10 dataset. Design and train a neural network that achieves high accuracy in classifying the images into their respective classes. Test different hyper-parameters.

ALGORITHM

```
1: Start
2: Load and preprocess 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 test 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

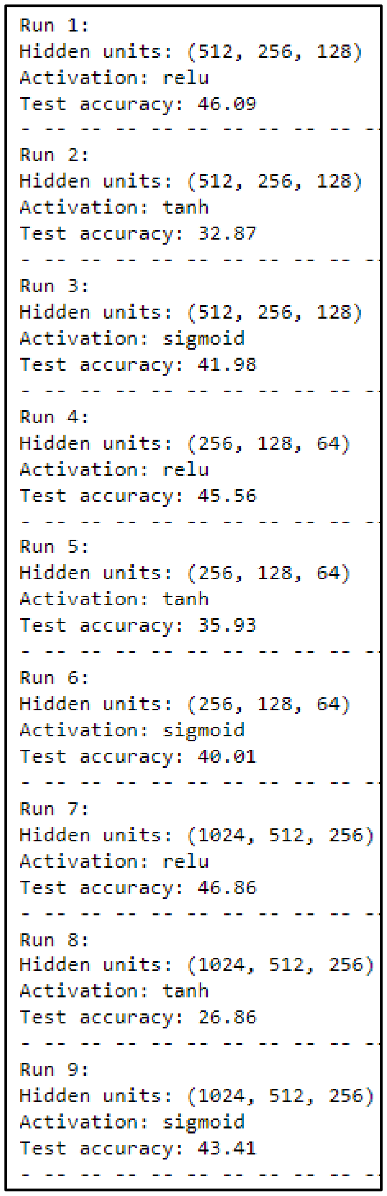
```
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.utils import to_categorical
import numpy as np
import matplotlib.pyplot as plt

(x_train, y_train), (x_test, y_test) = cifar10.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0
y_train, y_test = to_categorical(y_train), to_categorical(y_test) # Convert labels to one-hot encoding
```

```
def create_model(hidden_units=None, activation=None):
    model = models.Sequential([
        layers.Flatten(input_shape=(32, 32, 3)),
        layers.Dense(hidden_units[0], activation=activation), # Hidden Layer 1
        layers.Dense(hidden_units[1], activation=activation), # Hidden Layer 2
        layers.Dense(hidden_units[2], activation=activation), # Hidden Layer 3
        layers.Dense(10, activation='softmax')
    ])
    return model
hidden_units_list = [(512, 256, 128), (256, 128, 64), (1024, 512, 256)]
activation_list = ['relu', 'tanh', 'sigmoid']

results_dict = {}
counter = 1

# Loop through combinations of hidden units and activations
for hidden_units in hidden_units_list:
    for activation in activation_list:
        model = create_model(hidden_units=hidden_units, activation=activation)
        model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
        history = model.fit(x_train, y_train, epochs=5, batch_size=64, validation_data=(x_test,
y_test))

        _, test_acc = model.evaluate(x_test, y_test)

        model_info = { # Create a dictionary for the current iteration
            "Hidden units": hidden_units,
            "Activation": activation,
            "Test accuracy": round(test_acc * 100, 4)
        }
        results_dict[counter] = model_info  # Add the current dictionary to the results dictionary

        counter += 1

for key, value in results_dict.items():
    print(f"Run {key}:")
    for info_key, info_value in value.items():
        print(f"{info_key}: {info_value}")
    print("- -" * 15)  # Dict prints Separator
print("\n")

max_accuracy_run = max(results_dict, key=lambda k: results_dict[k]["Test accuracy"])
max_accuracy_info = results_dict[max_accuracy_run]
print("Run with the highest test accuracy:")
print(f"Run {max_accuracy_run}:")
for info_key, info_value in max_accuracy_info.items():
    print(f"{info_key}: {info_value}")
num_images = 3
sample_images = x_train[:num_images]
predictions = model.predict(sample_images)

def plot_probability_meter(predictions, image):
    class_labels = ["airplane", "automobile", "bird", "cat", "deer", "dog", "frog", "horse", "ship",
"truck"]

    fig, axs = plt.subplots(1, 2, figsize=(10, 2))

    # Plot the image
    axs[0].imshow(image)
    axs[0].axis('off')

    # Plot the probability meter
    axs[1].barh(class_labels, predictions[0], color='blue')
    axs[1].set_xlim([0, 1])
    # axs[1].set_xlabel('Probability')

    plt.tight_layout()
    plt.show()

for i in range(num_images):
    plot_probability_meter(predictions[i:i+1], sample_images[i])
```

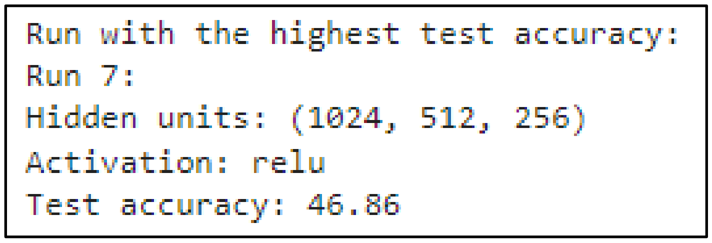
RESULT

The program for CIFAR-10 image classification has been completed and the output is verified.

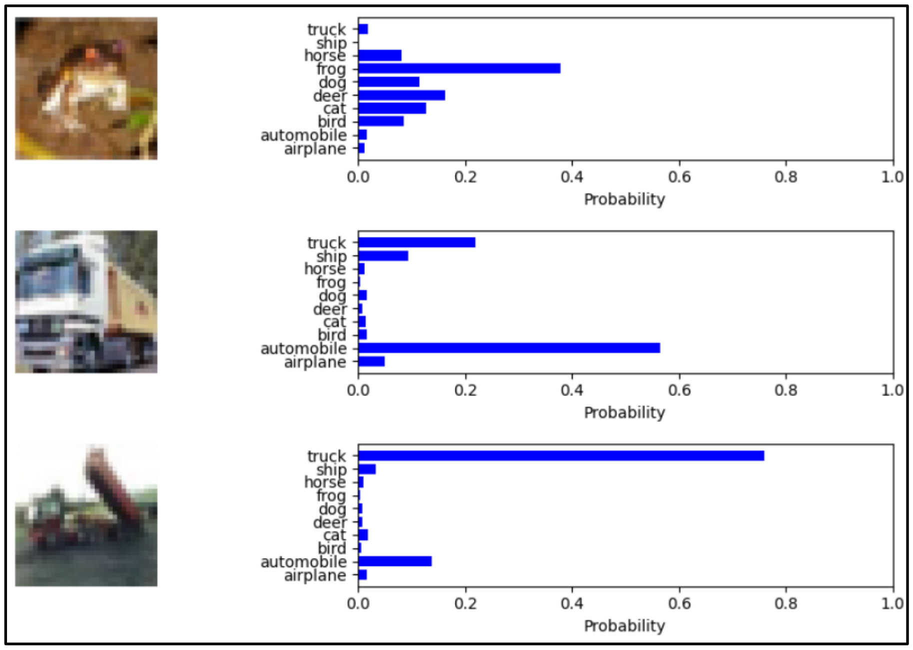
OUTPUT



```
Run 1:
Hidden units: (512, 256, 128)
Activation: relu
Test accuracy: 46.09
- -- -- -- -- -- -- -- -- -- -
Run 2:
Hidden units: (512, 256, 128)
Activation: tanh
Test accuracy: 32.87
- -- -- -- -- -- -- -- -- -- -
Run 3:
Hidden units: (512, 256, 128)
Activation: sigmoid
Test accuracy: 41.98
- -- -- -- -- -- -- -- -- -- -
Run 4:
Hidden units: (256, 128, 64)
Activation: relu
Test accuracy: 45.56
- -- -- -- -- -- -- -- -- -- -
Run 5:
Hidden units: (256, 128, 64)
Activation: tanh
Test accuracy: 35.93
- -- -- -- -- -- -- -- -- -- -
Run 6:
Hidden units: (256, 128, 64)
Activation: sigmoid
Test accuracy: 40.01
- -- -- -- -- -- -- -- -- -- -
Run 7:
Hidden units: (1024, 512, 256)
Activation: relu
Test accuracy: 46.86
- -- -- -- -- -- -- -- -- -- -
Run 8:
Hidden units: (1024, 512, 256)
Activation: tanh
Test accuracy: 26.86
- -- -- -- -- -- -- -- -- -- -
Run 9:
Hidden units: (1024, 512, 256)
Activation: sigmoid
Test accuracy: 43.41
- -- -- -- -- -- -- -- -- -- -
```



```
Run with the highest test accuracy:
Run 7:
Hidden units: (1024, 512, 256)
Activation: relu
Test accuracy: 46.86
```



EXPERIMENT - 4

CIFAR-10: TESTING WITH DIFFERENT HYPERPARAMETERS

AIM:

Implement a feed forward neural network with three hidden layers for the CIFAR-10 dataset.Train the network using a baseline optimization algorithm, such as stochastic gradient descent (SGD) or Adam, without any specific weight initialization technique or regularization technique. Record the accuracy and loss during training.

(a) Repeat the training process with Xavier initialization for weight initialization. Compare the convergence speed and accuracy of the network with the baseline results. Analyze the impact of Xavier initialization on the network's performance.

(b) Repeat the training process with Kaiming initialization for weight initialization. Compare the convergence speed and accuracy of the network with the baseline results. Analyze the impact of Kaiming initialization on the network's performance.

(c) Implement dropout regularization by applying dropout to the hidden layers of the network. Train the network with dropout regularization and compare its performance with the baseline results. Analyze the impact of dropout on the network's performance in terms of accuracy and overfitting

(d) Implement L1 or L2 regularization techniques by adding a regularization term to the loss function during training. Train the network with regularization and compare its performance with the baseline results. Analyze the impact of regularization on the network's performance in terms of accuracy and prevention of overfitting.

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 or SGD optimizer.
6: Evaluate the baseline model on the test set.
7: Repeat steps 3-5 using Xavier and Kaiming weight initializers.
8: Repeat steps 3-5 using dropout and L2 kernel regularizer.
9: Compare the results of the different models.
10: Stop
```

PROGRAM

```
# BASE MODEL
import tensorflow as tf
from tensorflow.keras import layers, models, regularizers
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.utils import to_categorical
import numpy as np
import time

(x_train, y_train), (x_test, y_test) = cifar10.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0
y_train, y_test = to_categorical(y_train), to_categorical(y_test) # Convert labels to one-hot encoding

base_model = models.Sequential([
    layers.Flatten(input_shape=(32, 32, 3)),
    layers.Dense(256, activation='relu'),
    layers.Dense(128, activation='relu'),
    layers.Dense(64, activation='relu'),
    layers.Dense(10, activation='softmax')
])

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

st = time.time()
base_model_history = base_model.fit(x_train, y_train, epochs=5, batch_size=64,
validation_data=(x_test, y_test))
sp = time.time()

print("- -" * 40)
```

```
base_test_loss, base_test_acc = base_model.evaluate(x_test, y_test)
base_time_taken = round(sp-st, 2)
print(f"Base Model Results:\n\nTest Loss: {round(base_test_loss, 4)}\nTest Accuracy: {round(base_test_acc * 100, 2)}%\nTime Taken: {base_time_taken} seconds")

# MODELS WITH KERNEL INITIALIZERS
# Xavier & Kaiming Weight Initializers
def create_model(initializer=None):
    model_with_kernel = models.Sequential([
        layers.Flatten(input_shape=(32, 32, 3)),
        layers.Dense(256, activation='relu', kernel_initializer=initializer),
        layers.Dense(128, activation='relu', kernel_initializer=initializer),
        layers.Dense(64, activation='relu', kernel_initializer=initializer),
        layers.Dense(10, activation='softmax')
    ])
    return model_with_kernel
res = []

# glorot_uniform - Xavier Weight Initializer, he_normal - Kaiming Weight Initializer
weight_initializers = ['glorot_uniform', 'he_normal']

for init in weight_initializers:
    print(f"Training with initializer: {init}\n")

    model = create_model(initializer=init)
    model.compile(optimizer='sgd',  loss='categorical_crossentropy', metrics=['accuracy'])

    wei_st = time.time()
    model.fit(x_train, y_train, epochs=5)
    wei_sp = time.time()
    test_loss, test_accuracy = model.evaluate(x_test, y_test)

    time_taken = wei_sp - wei_st
    res.append((time_taken, test_accuracy))
    print(f"\nTest Accuracy (using '{init}' weight initialization): {round(test_accuracy * 100, 2)}%\n\nTime Taken: {round(time_taken, 2)} seconds\n")
    print("- -" * 40)

# Comparison with baseline results
base_glorot_acc = round(base_test_acc / res[0][1], 2)
base_glorot_time = round(base_time_taken / res[0][0], 3)
base_he_acc = round(base_test_acc / res[1][1])
base_he_time = round(base_time_taken / res[1][0], 3)

print(f"Baseline Results Comparison:\n\nAccuracy comparison:\n{weight_initializers[0]}: {base_glorot_acc}% accurate than base model\n{weight_initializers[1]}: {base_he_acc}% accurate than base model")
print(f"\nTime comparison:\n{weight_initializers[0]}: {base_glorot_time} seconds faster than base model\n\n{weight_initializers[1]}: {base_he_time} seconds faster than base model")

# MODEL WITH DROPOUT LAYER
# Dropout Rate - 0.2
model_with_dropout = models.Sequential([
    layers.Flatten(input_shape=(32, 32, 3)),
    layers.Dense(256, activation='relu'),
    layers.Dropout(0.2),
    layers.Dense(128, activation='relu'),
    layers.Dropout(0.2),
    layers.Dense(64, activation='relu'),
    layers.Dropout(0.2),
    layers.Dense(10, activation='softmax')])
```

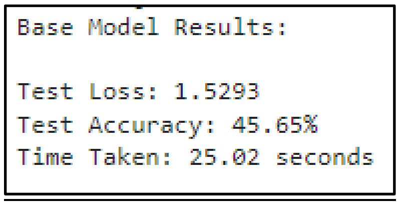
```
# Load and preprocess CIFAR-10 dataset
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

model_with_dropout.compile(optimizer='sgd', loss='sparse_categorical_crossentropy',
metrics=['accuracy'])
model_with_dropout.fit(x_train, y_train, epochs=5)
dropout_test_loss, dropout_test_accuracy = model_with_dropout.evaluate(x_test, y_test)

print(f"\nTest Accuracy: {round(dropout_test_accuracy * 100, 4)}%")
print("- -" * 40)
# Comparison with baseline results
base_dropout_acc = round(base_test_acc / dropout_test_accuracy , 2)

print(f"Baseline Results Comparison:\nAccuracy: {base_dropout_acc}% faster than base model")

# MODEL WITH KERNEL REGULARIZER
# L2 Kernel Regularizer
model_with_reg = models.Sequential([
    layers.Flatten(input_shape=(32, 32, 3)),
    layers.Dense(256, activation='relu', kernel_regularizer=regularizers.l2(0.01)),
    layers.Dense(128, activation='relu', kernel_regularizer=regularizers.l2(0.01)),
    layers.Dense(64, activation='relu', kernel_regularizer=regularizers.l2(0.01)),
    layers.Dense(10, activation='softmax')
])

(x_train, y_train), (x_test, y_test) = cifar10.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0
```

```
model_with_reg.compile(optimizer='sgd',                    loss='sparse_categorical_crossentropy',
metrics=['accuracy'])
model_with_reg.fit(x_train, y_train, epochs=5)

reg_test_loss, reg_test_accuracy = model_with_reg.evaluate(x_test, y_test)
print(f"\nTest Accuracy : {round(reg_test_accuracy * 100, 4)}%")

print("- -" * 40)

# Comparison with baseline results
base_reg_acc = round(base_test_acc / reg_test_accuracy , 2)
print(f"Baseline Results Comparison:\nAccuracy: {base_reg_acc}% faster than base model")
```

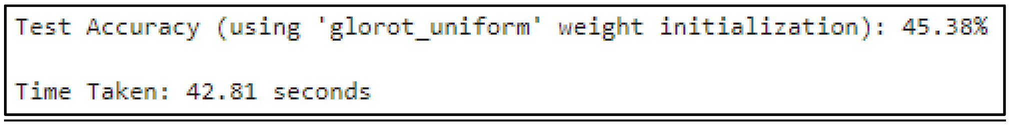
RESULT

The program for CIFAR-10 images classification using different weight initialization and regularization techniques has been completed and the output is verified.

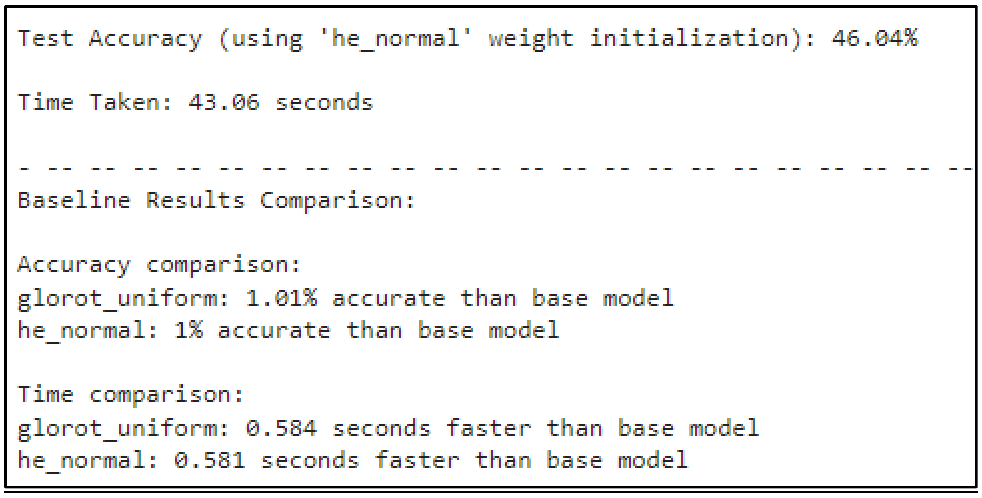
OUTPUT



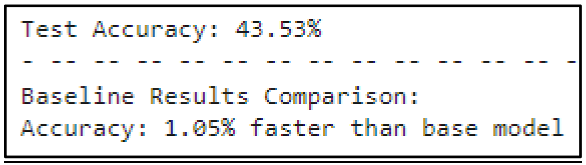
```
Base Model Results:

Test Loss: 1.5293
Test Accuracy: 45.65%
Time Taken: 25.02 seconds
```



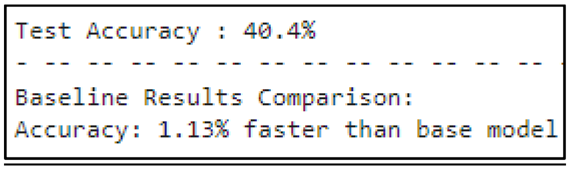
```
Test Accuracy (using 'glorot_uniform' weight initialization): 45.38%

Time Taken: 42.81 seconds
```



```
Test Accuracy (using 'he_normal' weight initialization): 46.04%

Time Taken: 43.06 seconds

- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
Baseline Results Comparison:

Accuracy comparison:
glorot_uniform: 1.01% accurate than base model
he_normal: 1% accurate than base model

Time comparison:
glorot_uniform: 0.584 seconds faster than base model
he_normal: 0.581 seconds faster than base model
```



```
Test Accuracy: 43.53%
- -- -- -- -- -- -- -- -- -- -- -- -- -
Baseline Results Comparison:
Accuracy: 1.05% faster than base model
```



```
Test Accuracy : 40.4%
- -- -- -- -- -- -- -- -- -- -- -- -- -
Baseline Results Comparison:
Accuracy: 1.13% faster than base model
```

EXPERIMENT - 5

MNIST CLASSIFICATION USING CNN

AIM:

Implement a Convolutional Neural Network (CNN) architecture for digit classification on the MNIST dataset. Design and train a CNN model that achieves high accuracy in recognizing handwritten digits.

ALGORITHM

```
1: Start
2: Load the MNIST dataset.
3: Preprocess the dataset and normalize it.
4: Create a CNN model with appropriate specifications for image classification.
5: Compile the model.
6: Evaluate the model on the test set and print the test accuracy.
7: Stop
```

PROGRAM

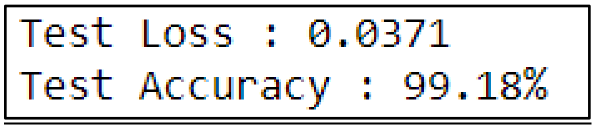
```
import tensorflow as tf
from tensorflow.keras.datasets import mnist
from tensorflow.keras.utils import to_categorical
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
train_images, test_images = train_images / 255.0, test_images / 255.0
train_labels, test_labels = to_categorical(train_labels), to_categorical(test_labels)
model = tf.keras.Sequential([
    tf.keras.layers.Conv2D(32, 3, activation='relu', input_shape=(28, 28, 1)),
    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=30, batch_size=64, validation_split=0.2)
```

```
test_loss, test_acc = model.evaluate(test_images, test_labels)
print(f"Test Loss : {round(test_loss, 4)}\nTest Accuracy : {round(test_acc * 100, 4)}%")
```

RESULT

The program for implementing CNN architecture for digit classification in the MNIST Dataset has been completed and the output is verified.

OUTPUT



```
Test Loss : 0.0371
Test Accuracy : 99.18%
```

EXPERIMENT - 6

MNIST CLASSIFICATION USING VGGNET-19

AIM:

Digit classification using pre-trained networks like VGGnet-19 for MNIST dataset and analyze and visualize performance improvement. Explore transfer learning using Convolutional Neural Networks (ConvNets) as fixed feature extractors and fine-tuning for image classification. Analyze their performance on a new image classification task while comparing the fixed feature extractor approach with fine-tuning.

ALGORITHM

```
1: Start
2: Load the MNIST dataset.
3: Convert the grayscale images to RGB (VGG requires 3 channels).
4: Resize the images to 224x224px to match the VGG input size.
5: Normalize pixel values to [0, 1].
6: Convert labels to one-hot encoding.
7: Load VGG19 pre-trained model.
8: Create a fixed feature extractor model by adding a dense layer on top of VGG19 model.
9: Train the fixed feature extractor model.
10: Create a fine-tuned model by adding custom classification layers on top of VGG19 model.
11: Freeze the layers of the base VGG model.
12: Train the fine-tuned model.
13: Evaluate the performance of both models.
14: Stop
```

PROGRAM

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

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

# Load a subset of the dataset (5000 training images and 500 testing images)
X_train = X_train[:5000]
y_train = y_train[:5000]
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)

# Load VGG19 pre-trained model
base_model = VGG19(include_top=False, weights='imagenet', input_shape=(224, 224, 3))

# Fixed feature extractor approach
model_frozen = Sequential()
model_frozen.add(Flatten(input_shape=X_train.shape[1:]))
model_frozen.add(Dense(256, activation='relu'))
model_frozen.add(Dense(10, activation='softmax'))

model_frozen.compile(optimizer=Adam(learning_rate=0.001), loss='categorical_crossentropy',
metrics=['accuracy'])

# Training the frozen model
history_frozen = model_frozen.fit(X_train_froz, y_train_froz, epochs=5, batch_size=32,
validation_data=(X_test_froz, y_test_froz))

# Convert grayscale images to RGB (VGG requires 3 channels)
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))

# Resize images to match VGG input size (224x224)
X_train_resized = tf.image.resize(X_train_rgb, (224, 224))
X_test_resized = tf.image.resize(X_test_rgb, (224, 224))

# Normalize pixel values to [0, 1]
```

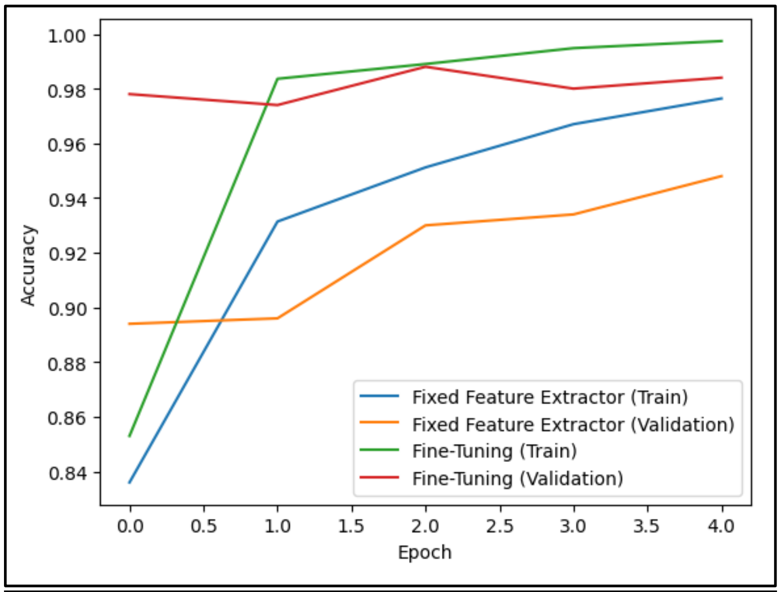
```
X_train_resized = X_train_resized / 255.0
X_test_resized = X_test_resized / 255.0

# Convert labels to one-hot encoding
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))

# Add custom classification layers on top
x = Flatten()(base_model.output)
x = Dense(256, activation='relu')(x)
output = Dense(10, activation='softmax')(x)
model_finetuned = Model(inputs=base_model.input, outputs=output) # Create the fine-tuned
model
# Freeze the layers of the base VGG model
for layer in base_model.layers:
    layer.trainable = False

model_finetuned.compile(optimizer=Adam(learning_rate=0.001),
loss='categorical_crossentropy', metrics=['accuracy'])

# Train the model
history_finetuned = model_finetuned.fit(X_train_resized, y_train_rgb, batch_size=64, epochs=5,
validation_data=(X_test_resized, y_test_rgb))

# Evaluate the model
loss, accuracy = model_finetuned.evaluate(X_test_resized, y_test_rgb)
print("Test accuracy:", accuracy)

plt.plot(history_frozen.history['accuracy'], label='Fixed Feature Extractor (Train)')
plt.plot(history_frozen.history['val_accuracy'], label='Fixed Feature Extractor (Validation)')
plt.plot(history_finetuned.history['accuracy'], label='Fine-Tuning (Train)')
plt.plot(history_finetuned.history['val_accuracy'], label='Fine-Tuning (Validation)')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

RESULT

The program for implementing VGGnet-19 for the MNIST dataset has been completed and the output is verified.

OUTPUT

```
Test Accuracy: 98.4%
```



EXPERIMENT - 7

IMDB DATASET REVIEW CLASSIFICATION USING RNN

AIM:
Implement a Recurrent Neural Network (RNN) for review classification on the IMDB dataset. Design and train an RNN model to classify movie reviews as positive or negative based on their sentiment.

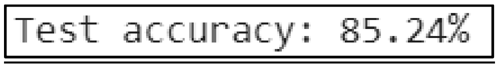
ALGORITHM
1: Start
2: Load the IMDB dataset.
3: Preprocess the dataset by padding the sequences to a maximum length of words.
4: Compile the model.
5: Evaluate the model on the test data and print the test accuracy.
6: Predict the sentiment of a new review by passing the review as input to the model. Applea
7: 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
(X_train, y_train), (X_test, y_test) = imdb.load_data(num_words=num_words)
X_train = pad_sequences(X_train, maxlen=max_length)
X_test = pad_sequences(X_test, maxlen=max_length)

model = Sequential()
model.add(Embedding(input_dim=num_words, output_dim=128, input_length=max_length))
model.add(LSTM(128))
model.add(Dense(1, activation='sigmoid'))

model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
model.fit(X_train, y_train, validation_split=0.2, epochs=5, batch_size=64)
loss, accuracy = model.evaluate(X_test, y_test)
print(f"Test accuracy: {round(accuracy * 100, 2)}%")

# Reshape the sequence to match the batch size expected by the model
test_seq = np.reshape(X_test[7], (1, -1))

pred = model.predict(test_seq)[0]
print('Positive Review') if int(pred[0]) == 1 else print(‘Negative Review’)

# Checking the correctness of prediction
print(y_test[7]) # 0 means Negative Review ; 1 means Positive Review
```

RESULT

The program for implementing RNN architecture for review classification on the IMDB dataset has been completed and the output is verified.

OUTPUT



```
Test accuracy: 85.24%
```

Input Sentence: “If only to avoid making this type of film in the future. This film is interesting as an experiment but tells no convincing story.”

Prediction: “Negative Review”

EXPERIMENT - 8

IMDB DATASET REVIEW CLASSIFICATION USING LSTM-GRU

AIM:

Analyze and visualize the performance change while using LSTM (Long Short-Term Memory) and GRU (Gated Recurrent Unit) instead of a standard RNN (Recurrent Neural Network) for sentiment analysis on the IMDB dataset. Compare the performance of different RNN architectures and understand their impact on sentiment classification.

ALGORITHM

```
1: Start
2: Load the IMDB dataset.
3: Preprocess the dataset by padding the sequences to a maximum length of words.
4: Create two models: an LSTM model and a GRU model.
5: Compile each model.
7: Evaluate each model on the test data and print the test accuracy.
8: Plot the training and validation loss and accuracy for each model.
9: Stop
```

PROGRAM

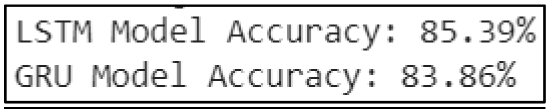
```
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras.datasets import imdb
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, GRU, Dense

num_words = 10000
max_length = 200
(X_train, y_train), (X_test, y_test) = imdb.load_data(num_words=num_words)
X_train = pad_sequences(X_train, maxlen=max_length)
X_test = pad_sequences(X_test, maxlen=max_length)

# LSTM model
lstm_model = Sequential()
lstm_model.add(Embedding(input_dim=num_words, output_dim=128,
input_length=max_length))
```

```
lstm_model.add(LSTM(128))
lstm_model.add(Dense(1, activation='sigmoid'))
lstm_model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

#GRU model
gru_model = Sequential()
gru_model.add(Embedding(input_dim=num_words, output_dim=128,
input_length=max_length))
gru_model.add(GRU(128))
gru_model.add(Dense(1, activation='sigmoid'))

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

lstm_history = lstm_model.fit(X_train, y_train, validation_split=0.2, epochs=5, batch_size=64)
gru_history = gru_model.fit(X_train, y_train, validation_split=0.2, epochs=5, batch_size=64)

lstm_loss, lstm_acc = lstm_model.evaluate(X_test, y_test)
gru_loss, gru_acc = gru_model.evaluate(X_test, y_test)

print(f"LSTM Model Accuracy: {round(lstm_acc * 100, 2)}%\nGRU Model Accuracy:
{round(gru_acc * 100, 2)}%")

# Plotting training history
plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)
plt.plot(lstm_history.history['loss'], label='LSTM Training Loss')
plt.plot(lstm_history.history['val_loss'], label='LSTM Validation Loss')
plt.plot(gru_history.history['loss'], label='GRU Training Loss')
plt.plot(gru_history.history['val_loss'], label='GRU Validation Loss')
plt.legend()
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Training and Validation Loss')
plt.subplot(1, 2, 2)
plt.plot(lstm_history.history['accuracy'], label='LSTM Training Accuracy')
plt.plot(lstm_history.history['val_accuracy'], label='LSTM Validation Accuracy')
plt.plot(gru_history.history['accuracy'], label='GRU Training Accuracy')
plt.plot(gru_history.history['val_accuracy'], label='GRU Validation Accuracy')
plt.legend()
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.title('Training and Validation Accuracy')
plt.tight_layout()
plt.show()
```

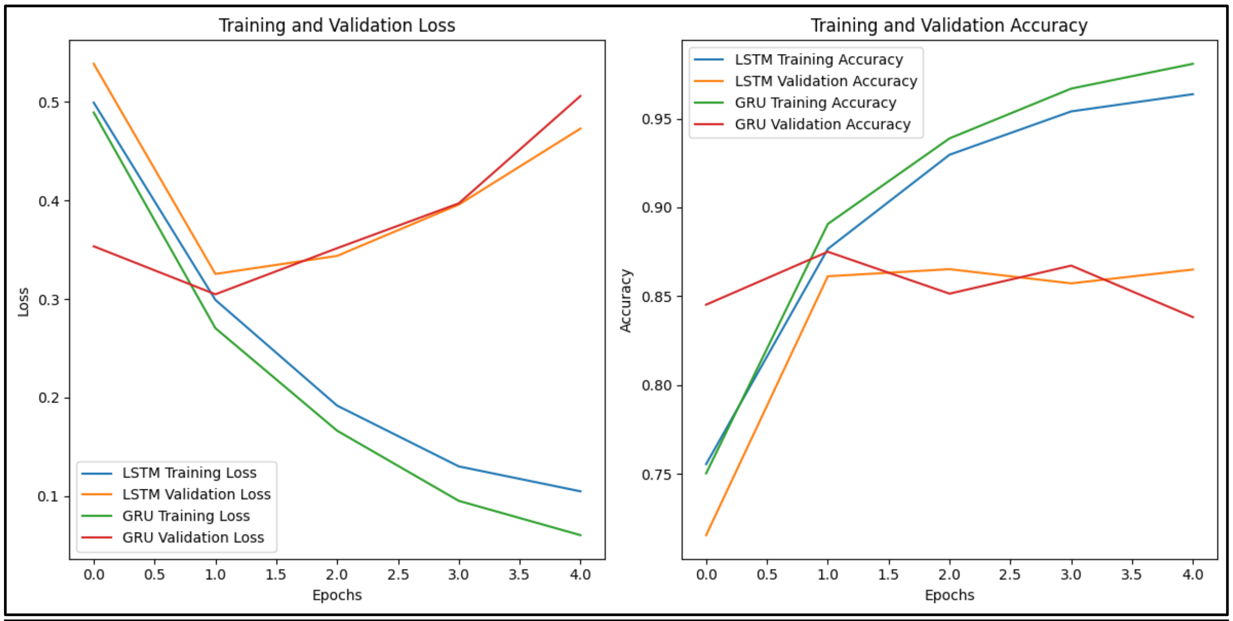
RESULT

The program for finding the performance change while using LSTM and GRU instead of RNN (Recurrent Neural Network) for sentiment analysis on the IMDB dataset has been completed and the output is verified.

OUTPUT



```
LSTM Model Accuracy: 85.39%
GRU Model Accuracy: 83.86%
```



EXPERIMENT - 9

REGRESSION ON NIFTY-50 DATA

AIM:

Implement time series forecasting for the NIFTY-50 dataset. Design and train a model to predict future values of the NIFTY-50 stock market index based on historical data.

ALGORITHM

```
1: Start
2: Load the NIFTY-50 dataset.
3: Preprocess the dataset – Normalize numerical OHLC Columns.
4: Split the dataset into training and test sets with no shuffling.
5: Create  a simple deep learning model.
6. Compile the model with MSE loss function.
7: Fit the model on OHLC columns against the Close(C) column.
8. Make predictions of future close values and record the observations.
8: Plot a graph of predictions and original close values and display the chart.
9: Stop
```

PROGRAM

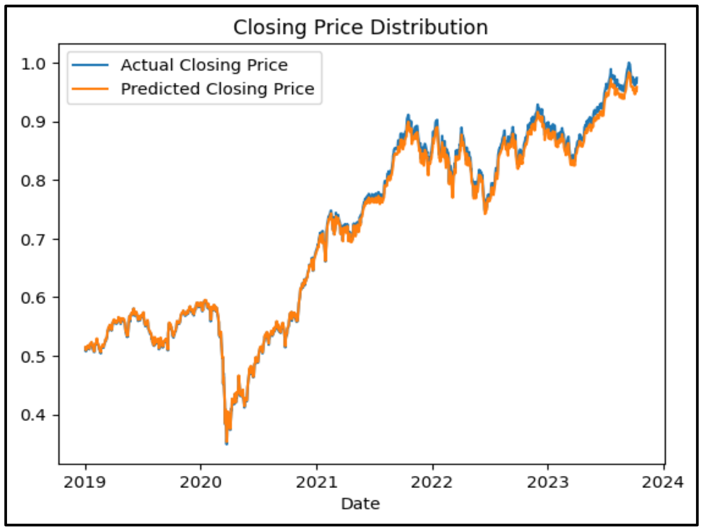
```
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
import tensorflow as tf
from tensorflow.keras.layers import Dense
from sklearn.metrics import mean_absolute_error
import matplotlib.pyplot as plt

nifty = "/content/drive/MyDrive/Colab Notebooks/nifty.csv"
data = pd.read_csv(nifty, index_col="Date", parse_dates=True)

# Normalize numerical columns
scaler = MinMaxScaler()
data[['Open', 'High', 'Low', 'Close']] = scaler.fit_transform(data[['Open', 'High', 'Low', 'Close']])

# Ensure data is not shuffled while splitting
train_data, test_data = train_test_split(data, test_size=0.2, shuffle=False)
```

```
model = tf.keras.Sequential([
    Dense(64, activation='relu'),
    Dense(32, activation='relu'),
    Dense(1, activation='linear')
])

model.compile(loss='mse', optimizer='adam')
model.fit(train_data[['Open', 'High', 'Low', 'Close']], train_data['Close'], epochs=70)
predicted_closing_prices = model.predict(test_data[['Open', 'High', 'Low', 'Close']])

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)}")
```

RESULT

The program for predicting future values of the NIFTY-50 stock market index based on historical data has been completed and the output is verified.

OUTPUT



```
Mean Absolute Error : 0.00581
```

Prepared by: Verified by: Approved by:

EXPERIMENT - 10

ENGLISH TO HINDI TRANSLATION USING AUTOENCODER

AIM:

Implement a shallow autoencoder and decoder network for machine translation using the Kaggle English to Hindi Neural Translation Dataset. Design and train a model to translate English sentences to Hindi by leveraging the power of autoencoders and decoders.

ALGORITHM

```
1: Start
2: Load the dataset and limit sentence lengths.
3: Assign numerical tokens to words and create token-word mappings.
4: Split data into training and testing sets and define batch size.
5: Define input, embedding, LSTM, and dense layers for encoder and decoder.
6: Compile and train the model using generated batches of data.
7: Extract the encoder model from the trained model.
8: Create a separate decoder model using extracted states.
9: Translate an English sentence and compare the predicted and actual Hindi translations.
10: Stop
```

PROGRAM

```
import numpy as np # USE COLAB
import pandas as pd
from sklearn.model_selection import train_test_split
from keras.layers import Input, LSTM, Embedding, Dense
from keras.models import Model

data=pd.read_csv("/content/drive/MyDrive/Colab Notebooks/English_Hindi_Clean.csv",
encoding='utf-8')

all_eng_words=set()
for eng in data['English']:
    for word in eng.split():
        if word not in all_eng_words:
            all_eng_words.add(word)

all_hin_words=set()
for hin in data['Hindi']:
    for word in hin.split():
        if word not in all_hin_words:
            all_hin_words.add(word)

data['len_eng_sen']=data['English'].apply(lambda x:len(x.split(" ")))
data['len_hin_sen']=data['Hindi'].apply(lambda x:len(x.split(" ")))

data=data[data['len_eng_sen']<=20]
data=data[data['len_hin_sen']<=20]

max_len_src=max(data['len_hin_sen'])
max_len_tar=max(data['len_eng_sen'])

inp_words = sorted(list(all_eng_words))
tar_words = sorted(list(all_hin_words))
num_enc_toks = len(all_eng_words)
num_dec_toks = len(all_hin_words)
num_enc_toks, num_dec_toks

num_dec_toks += 1 #for zero padding

inp_tok_idx = dict([(word, i+1) for i, word in enumerate(inp_words)])
tar_tok_idx = dict([(word, i+1) for i, word in enumerate(tar_words)])

rev_inp_char_idx = dict((i, word) for word, i in inp_tok_idx.items())
rev_tar_char_idx = dict((i, word) for word, i in tar_tok_idx.items())

X, y = data['English'], data['Hindi']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2,random_state=42)
X_train.shape, X_test.shape

def generate_batch(X = X_train, y = y_train, batch_size = 128):
    while True:
        for j in range(0, len(X), batch_size):
            enc_inp_data = np.zeros((batch_size, max_len_src),dtype='float32')
            dec_inp_data = np.zeros((batch_size, max_len_tar),dtype='float32')
            dec_tar_data = np.zeros((batch_size, max_len_tar, num_dec_toks),dtype='float32')
            for i, (inp_text, tar_text) in enumerate(zip(X[j:j+batch_size], y[j:j+batch_size])):
                for t, word in enumerate(inp_text.split()):
```

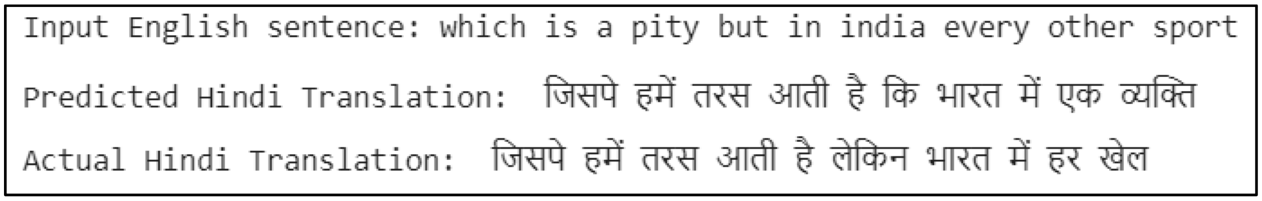
```
                enc_inp_data[i, t] = inp_tok_idx[word] # enc input seq
            for t, word in enumerate(tar_text.split()):
                if t<len(tar_text.split())-1:
                    dec_inp_data[i, t] = tar_tok_idx[word] # decoder input seq
                if t>0:
                    # decoder target sequence (one hot encoded)
                    # does not include the START_ token
                    # Offset by one timestep
                    dec_tar_data[i, t - 1, tar_tok_idx[word]] = 1.
        yield([enc_inp_data, dec_inp_data], dec_tar_data)

latent_dim=300

enc_inps = Input(shape=(None,))
enc_emb =  Embedding(num_enc_toks, latent_dim, mask_zero = True)(enc_inps)
enc_lstm = LSTM(latent_dim, return_state=True)
enc_outputs, st_h, st_c = enc_lstm(enc_emb)
enc_states = [st_h, st_c]

dec_inps = Input(shape=(None,))
dec_emb_layer = Embedding(num_dec_toks, latent_dim, mask_zero = True)
dec_emb = dec_emb_layer(dec_inps)
dec_lstm = LSTM(latent_dim, return_sequences=True, return_state=True)
dec_outputs, _, _ = dec_lstm(dec_emb, initial_state=enc_states)
dec_dense = Dense(num_dec_toks, activation='softmax')
dec_outputs = dec_dense(dec_outputs)

model = Model([enc_inps, dec_inps], dec_outputs)
model.compile(optimizer='rmsprop', loss='categorical_crossentropy')

train_samples = len(X_train)
val_samples = len(X_test)
batch_size = 128

model.fit_generator(generator = generate_batch(X_train, y_train, batch_size = batch_size),
                steps_per_epoch = train_samples//batch_size, epochs=100,
                validation_data = generate_batch(X_test, y_test, batch_size = batch_size),
                validation_steps = val_samples//batch_size)
```

```
enc_model = Model(enc_inps, enc_states)

dec_st_inp_h = Input(shape=(latent_dim,))
dec_st_inp_c = Input(shape=(latent_dim,))
dec_states_inps = [dec_st_inp_h, dec_st_inp_c]

dec_emb2= dec_emb_layer(dec_inps) # Get the embeddings of the decoder sequence

dec_outputs2, st_h2, st_c2 = dec_lstm(dec_emb2, initial_state=dec_states_inps)
dec_states2 = [st_h2, st_c2]
dec_outputs2 = dec_dense(dec_outputs2) # A dense softmax layer to generate prob dist. over
the target vocabulary

dec_model = Model(
    [dec_inps] + dec_states_inps,
    [dec_outputs2] + dec_states2)

def translate(inp_seq):
    states_value = enc_model.predict(inp_seq)
    tar_seq = np.zeros((1,1))
    # Populate the first character of target sequence with the start character.
    tar_seq[0, 0] = tar_tok_idx['START_']

    stop_cond = False
    dec_sen = ''
    while not stop_cond:
        output_toks, h, c = dec_model.predict([tar_seq] + states_value)

        sampled_tok_idx = np.argmax(output_toks[0, -1, :])
        sampled_char = rev_tar_char_idx[sampled_tok_idx]
        dec_sen += ' '+sampled_char

        if (sampled_char == '_END' or
           len(dec_sen) > 50):
            stop_cond = True

        tar_seq = np.zeros((1,1))
        tar_seq[0, 0] = sampled_tok_idx
        states_value = [h, c]

    return dec_sen

train_gen = generate_batch(X_train, y_train, batch_size = 1)
k=0
(inp_seq, actual_output), _ = next(train_gen)
hin_sen = translate(inp_seq)
print(f'''Input English sentence: {X_train[k:k+1].values[0]}\n
          Predicted Hindi Translation: {hin_sen[:-4]}\n
          Actual Hindi Translation: {y_train[k:k+1].values[0][6:-4]}''')
```

RESULT

The program for implementing a model to translate English sentences to Hindi using autoencoders and decoder architecture has been completed and the output is verified.

OUTPUT



```
Input English sentence: which is a pity but in india every other sport

Predicted Hindi Translation:  जिसपे हमें तरस आती है कि भारत में एक व्यक्ति

Actual Hindi Translation:  जिसपे हमें तरस आती है लेकिन भारत में हर खेल
```