KARPAGAM ACADEMY OF HIGHER EDUCATION

(Deemed to be University)

(Established Under Section 3 of UGC Act, 1956)

(Accredited with A+ Grade by NAAC in the Second Cycle)

Eachanari (Post), Coimbatore – 641 021.

DEPARTMENTOF COMPUTER APPLICATIONS

BACHELOR OF COMPUTER SCIENCE

(ARTIFICIAL INTELLIGENCE AND DATA SCIENCE)

DEEP LEARNING - PRACTICALS

(21ADU611A)

III B.Sc CS (AI & DS)

SEMESTER: VI

OCTOBER – APRIL 2024

NAME:___________________________

REG.NO.:__________________________

KARPAGAM ACADEMY OF HIGHER EDUCATION

(Deemed to be University)

(Established Under Section 3 of UGC Act, 1956)

(Accredited with A+ Grade by NAAC in the Second Cycle)

Eachanari (Post), Coimbatore – 641 021.

DEPARTMENT OF COMPUTER APPLICATIONS

CERTIFICATE

This is to certify that this is a bonafide record of work done by___________________________ Register No: __________________ III year / VI Semester **BACHELOR OF COMPUTER SCIENCE(ARTIFICIAL INTELLIGENCE AND DATA SCIENCE)**for the practical Examination in **DEEP LEARNING - PRACTICALS (21ADU611A)** held on ________.

Staff in-charge **Head of the Department**

(Internal Examiner) **(External Examiner)**

INDEX

S.NO.	DATE	TITLE OF THE EXPERIMENT	PAGE NO.	SIGNATURE
1.				
2.				
3.				
4.				
5.				
6.				
7.				
8.				
9.				
10.				

EXP: 1

VECTOR ADDITION USING TENSORFLOW

AIM:

To implement vector addition, subtraction, multiplication and division in tensorflow.

ALGORITHM:

Step 1 : Start the process.

Step 2 : Open the Google colab

Step 3 : Import tensorflow library

Step 4 : Create two vectors

Step 5 : check the dimensions of the vectors

Step 6 : Perform addition, subtraction, multiplication and division operations of matrices.

Step 7 : Print the results.

Step 8 : Close the application and stop the process.

PROGRAM:

```
# importing packages
import tensorflow as tf

# creating two tensors
matrix = tf.constant([[10, 2], [10, 4]])
matrix1 = tf.constant([[12, 4], [12, 8]])

# create a vector
vector = tf.constant([10, 10])

# checking the dimensions of vector
vector.ndim

print(matrix)
print('the number of dimensions of a matrix is :\
'+str(matrix.ndim))

# addition of two matrices
print('Addition of Two Matrix :');
print(matrix+matrix1)
```

```
# subtraction of two matrices
print('Subtraction of two matrix :');
print(matrix1 - matrix)

# multiplication of two matrices
print('Multiplication of two matrix :');
print(matrix1 * matrix)

# division of two matrices
print('Division of two matrix :');
print(matrix1 / matrix)
```

OUTPUT:

```
tf.Tensor(
[[10  2]
 [10  4]], shape=(2, 2), dtype=int32)
the number of dimensions of a matrix is :2


Addition of Two Matrix :
tf.Tensor(
[[22  6]
 [22 12]], shape=(2, 2), dtype=int32)
Subtraction of two matrix :
tf.Tensor(
[[2 2]
 [2 4]], shape=(2, 2), dtype=int32)
Multiplication of two matrix :
tf.Tensor(
[[120   8]
 [120  32]], shape=(2, 2), dtype=int32)
Division of two matrix :
tf.Tensor(
[[1.2 2. ]
 [1.2 2. ]], shape=(2, 2), dtype=float64)
```

RESULT:

The above program has been executed successfully and its output has been verified.

Exp: 2

REGRESSION MODEL IN KERAS

AIM:

To implement a simple problem like regression model in Keras.

ALGORITHM:

Step 1 : Start the process.

Step 2 : Open the Google colab

Step 3 : Import required libraries.

Step 4 : Load the dataset.

Step 5 : Preprocess the data

Step 6 : Define and compile the regression model

Step 7 : Train the model

Step 8 : Evaluate the model

Step 9 : Plot the training history.

Step 10: Close the application and stop the process.

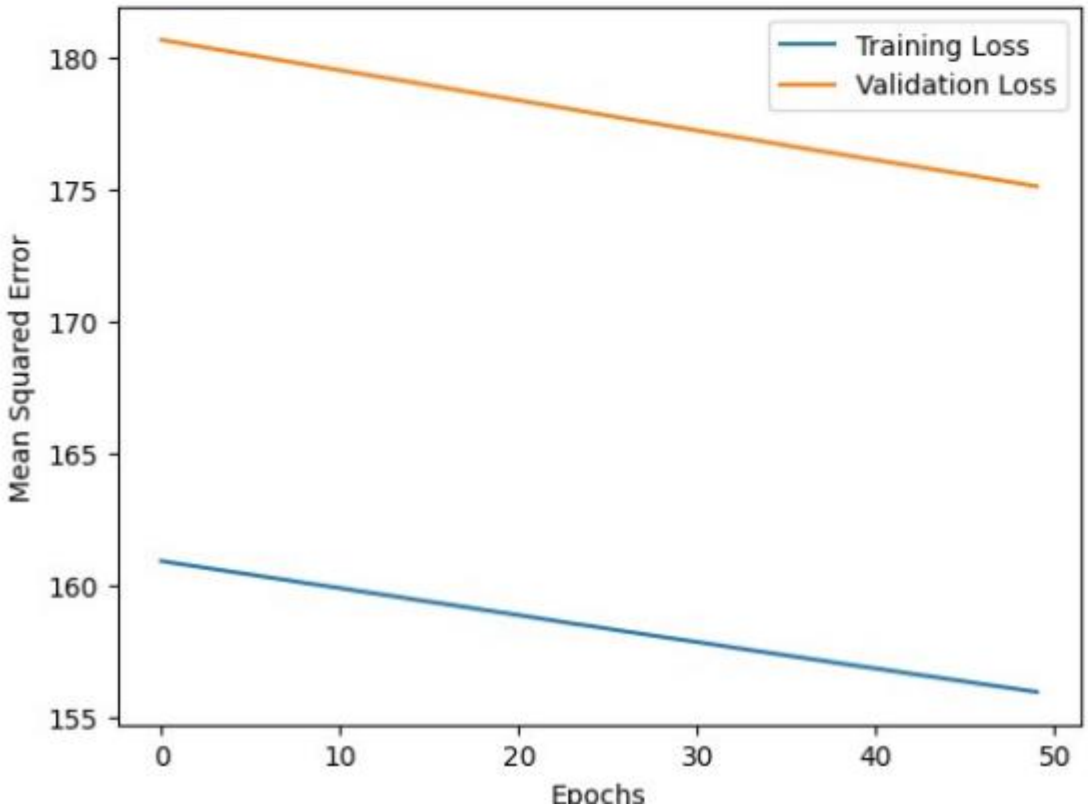
PROGRAM:

```
!pip install tensorflow


import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
import matplotlib.pyplot as plt


np.random.seed(42)
X = np.random.rand(100, 1) * 10
y = 2 * X + 1 + np.random.randn(100, 1) * 2
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

model = Sequential()
model.add(Dense(1, input_dim=1, activation='linear')) # Simple linear
regression with one input and one output
model.compile(optimizer='adam', loss='mean_squared_error')

history = model.fit(X_train_scaled, y_train, epochs=50, batch_size=32,
validation_data=(X_test_scaled, y_test), verbose=1)

loss = model.evaluate(X_test_scaled, y_test, verbose=0)
print(f'Mean Squared Error on Test Data: {loss}')

plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Mean Squared Error')
plt.legend()
plt.show()
```

OUTPUT:

```
Epoch 50/50
3/3 [==============================] - 0s 16ms/step - loss: 155.96
val_loss: 175.1071
Mean Squared Error on Test Data: 175.10708618164062
```



RESULT:

The above program has been executed successfully and its output has been verified.

Exp: 3

MULTILAYER PERCEPTRON

AIM:

To implement a perceptron in TensorFlow/Keras Environment.

ALGORITHM:

Step 1 : Start the process.

Step 2 : Open the Google colab

Step 3 : Import required libraries.

Step 4 : create mutli-layer perceptron classifier

Step 5 : Train the model

Step 6 : Evaluate the performance and make prediction

Step 7 : Print the accuracy.

Step 8: Close the application and stop the process.

PROGRAM:

```
from sklearn.neural_network import MLPClassifier
```

```
X = [[0, 0], [1, 1]]
y = [0, 1]
```

```
# create mutli-layer perceptron classifier
clf = MLPClassifier(solver='lbfgs', alpha=1e-5,
                    hidden_layer_sizes=(5, 2), random_state=1)
```

```
# train
clf.fit(X, y)

# make predictions
print( clf.predict([[2., 2.]]) )
print( clf.predict([[0, -1]]) )
print( clf.predict([[1, 2]]) )
```

OUTPUT:

RESULT:

The above program has been executed successfully and its output has been verified.

Exp: 4

FEED-FORWARD NETWORK IN TENSORFLOW/KERAS

AIM:

To implement a Feed-Forward Network in TensorFlow/Keras.

ALGORITHM:

Step 1 : Start the process.

Step 2 : Open the Google colab

Step 3 : Import required libraries.

Step 4 : Set the hyperparameters

Step 5 : Load the MNIST fashion dataset

Step 6 : Create and train the model

Step 7 : Test the model

Step 8 : Print the accuracy.

Step 9: Close the application and stop the process.

PROGRAM:

```
import torch
import torchvision
import torch.nn as nn
import torchvision.transforms as transforms
import numpy as np
```

```
#hyperparamter
input_size = 28*28
n_classes = 10 #output size
learning_rate = 0.001
hidden_size = 512
batch_size = 32
num_epochs = 5
```

```
# dataset -FAshionMNIST
```

```
train_dataset = torchvision.datasets.FashionMNIST(root="data/data/",
                                                  transform=transforms.ToTensor(),
                                                  train=True,
                                                  download=True)
```

```
# dataset -FAshionMNIST


test_dataset = torchvision.datasets.FashionMNIST(root="data/data/",
                                                 transform=transforms.ToTensor(),
                                                 train=False)
```

```
# data loader
train_loader = torch.utils.data.DataLoader(dataset=train_dataset,
                                           shuffle=True,
                                           batch_size=batch_size)
```

```
# test data loader
test_loader = torch.utils.data.DataLoader(dataset=test_dataset,
                                          batch_size=batch_size)
```

```
# model
class FeedForward(nn.Module):
    def __init__(self, input_size, n_classes, hidden_size):
        super(FeedForward, self).__init__()
        self.fc1 = nn.Linear(input_size, hidden_size)
        self.relu = nn.ReLU()
        self.fc2 = nn.Linear(hidden_size, n_classes)
    def forward(self,x):
        out = self.fc1(x)
        out = self.relu(out)
        out = self.fc2(out)
        return out
```

```
model = FeedForward(input_size, n_classes, hidden_size)
```

```
#loss and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(),lr=learning_rate)
```

```
total_size = len(train_loader)
# Train the model
for epoch in range(num_epochs):
    for i,(images,labels) in enumerate(train_loader,0):
        images = images.reshape(-1,input_size)

        #forward
        outputs = model(images)
        loss = criterion(outputs,labels)

        optimizer.zero_grad()

        #backpropagation
        loss.backward()
        optimizer.step()

        if (i+1)%500==0:
            print("Epoch {}/{} Step {}/{} : Loss {:.4f}".format(epoch+1,num_epochs,i+1,
total_size,loss))
```

```
# test the model
with torch.no_grad():
    correct = 0.0
    total = 0.0
    for images, labels in test_loader:
        images = images.reshape(-1,input_size)
        outputs = model(images)
        _,prediction = torch.max(outputs,1)
        total += labels.size(0)
        correct += (prediction==labels).sum()
    print("Accuracy : {:.2f}".format((correct*100.0)/total))
```

OUPUT:

Feed Forward Neural Network (1).ipynb

File Edit View Insert Runtime Tools Help Last saved at 12:38 PM

\+ Code + Text

```python
        #backpropagation
        loss.backward()
        optimizer.step()

        if (i+1)%500==0:
            print("Epoch {}/{} Step {}/{} : Loss {:.4f}".format(epoch+1,num_epochs,i+1, total_size,loss))
```

```
Epoch 1/5 Step 500/1875 : Loss 0.5962
Epoch 1/5 Step 1000/1875 : Loss 0.3362
Epoch 1/5 Step 1500/1875 : Loss 0.2992
Epoch 2/5 Step 500/1875 : Loss 0.2429
Epoch 2/5 Step 1000/1875 : Loss 0.5426
Epoch 2/5 Step 1500/1875 : Loss 0.2334
Epoch 3/5 Step 500/1875 : Loss 0.3212
Epoch 3/5 Step 1000/1875 : Loss 0.6971
Epoch 3/5 Step 1500/1875 : Loss 0.3913
Epoch 4/5 Step 500/1875 : Loss 0.2047
Epoch 4/5 Step 1000/1875 : Loss 0.2900
Epoch 4/5 Step 1500/1875 : Loss 0.4703
Epoch 5/5 Step 500/1875 : Loss 0.2819
Epoch 5/5 Step 1000/1875 : Loss 0.1987
Epoch 5/5 Step 1500/1875 : Loss 0.2045
```

```python
# test the model
with torch.no_grad():
    correct = 0.0
    total = 0.0
    for images, labels in test_loader:
        images = images.reshape(-1,input_size)
        outputs = model(images)
        _,prediction = torch.max(outputs,1)
        total += labels.size(0)
        correct += (prediction==labels).sum()
    print("Accuracy : {:.2f}".format((correct*100.0)/total))
```

```
Accuracy : 88.00
```

RESULT:

The above program has been executed successfully and its output has been verified.

Exp: 5

IMAGE CLASSIFIER USING CNN IN TENSORFLOW/KERAS

AIM:

To implement a Transfer Learning concept in Image Classification.

ALGORITHM:

Step 1 : Start the process.

Step 2 : Open the Google colab

Step 3 : Import required libraries.

Step 4 : Load CIFAR-10 Dataset.

Step 5 : Build the CNN model

Step 6 : Train the model

Step 7 : Compile the model and display the model summary.

Step 8 : Evaluate the model on test dataset.

Step 9 : Print the accuracy.

Step 10: Close the application and stop the process.

PROGRAM:

```
# Import necessary libraries
import tensorflow as tf
from tensorflow.keras import layers, models, datasets
from tensorflow.keras.utils import to_categorical

# Load the CIFAR-10 dataset
(train_images, train_labels), (test_images, test_labels) = datasets.cifar10.load_data()

# Normalize pixel values to be between 0 and 1
train_images, test_images = train_images / 255.0, test_images / 255.0

# One-hot encode the labels
train_labels = to_categorical(train_labels, num_classes=10)
test_labels = to_categorical(test_labels, num_classes=10)
```

```python
# Build the CNN model
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))

# Compile the model
model.compile(optimizer='adam',
         loss='categorical_crossentropy',
         metrics=['accuracy'])

# Display model summary
model.summary()

# Train the model
history = model.fit(train_images, train_labels, epochs=10,
              validation_data=(test_images, test_labels))

# Evaluate the model on the test set
test_loss, test_acc = model.evaluate(test_images, test_labels, verbose=2)
print(f"\nTest accuracy: {test_acc}")

# Plot training history
import matplotlib.pyplot as plt

plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label='val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(loc='lower right')
plt.show()
```

OUTPUT:

colab.research.google.com/drive/1PFE_TnlA0GdktwBPqfgGC1YiJXRSidH9#scrollTo=ck2nG_Oikbn5

imgclassifierusingcnn.ipynb

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

```
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(loc='lower right')
plt.show()
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 [==============================] - 3s 0us/step
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 30, 30, 32)        896

 max_pooling2d (MaxPooling2  (None, 15, 15, 32)        0
 D)

 conv2d_1 (Conv2D)           (None, 13, 13, 64)        18496

 max_pooling2d_1 (MaxPoolin  (None, 6, 6, 64)          0
 g2D)

 conv2d_2 (Conv2D)           (None, 4, 4, 64)          36928

 flatten (Flatten)           (None, 1024)              0

 dense (Dense)               (None, 64)                65600

 dense_1 (Dense)             (None, 10)                650

=================================================================
Total params: 122570 (478.79 KB)
Trainable params: 122570 (478.79 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Epoch 1/10
 776/1563 [=============>................] - ETA: 33s - loss: 1.7344 - accuracy: 0.3553
```

RESULT:

The above program has been executed successfully and its output has been verified.

Exp: 6

TRANSFER LEARNING CONCEPT IN IMAGE CLASSIFICATION

AIM:

To implement Transfer Learning concept in Image Classification

ALGORITHM:

Step 1 : Start the process.

Step 2 : Open the Google colab

Step 3 : Import required libraries.

Step 4 : Load CIFAR-10 Public dataset.

Step 5 : Define the ResNet50 model with pre-trained weights (excluding the top layer)

Step 6 : Freeze the layers of the pre-trained model

Step 7 : Create a new model on top of the pre-trained model

Step 8 : Freeze the layers of the pre-trained model

Step 9 : Compile the model and print the accuracy.

Step 10: Close the application and stop the process.

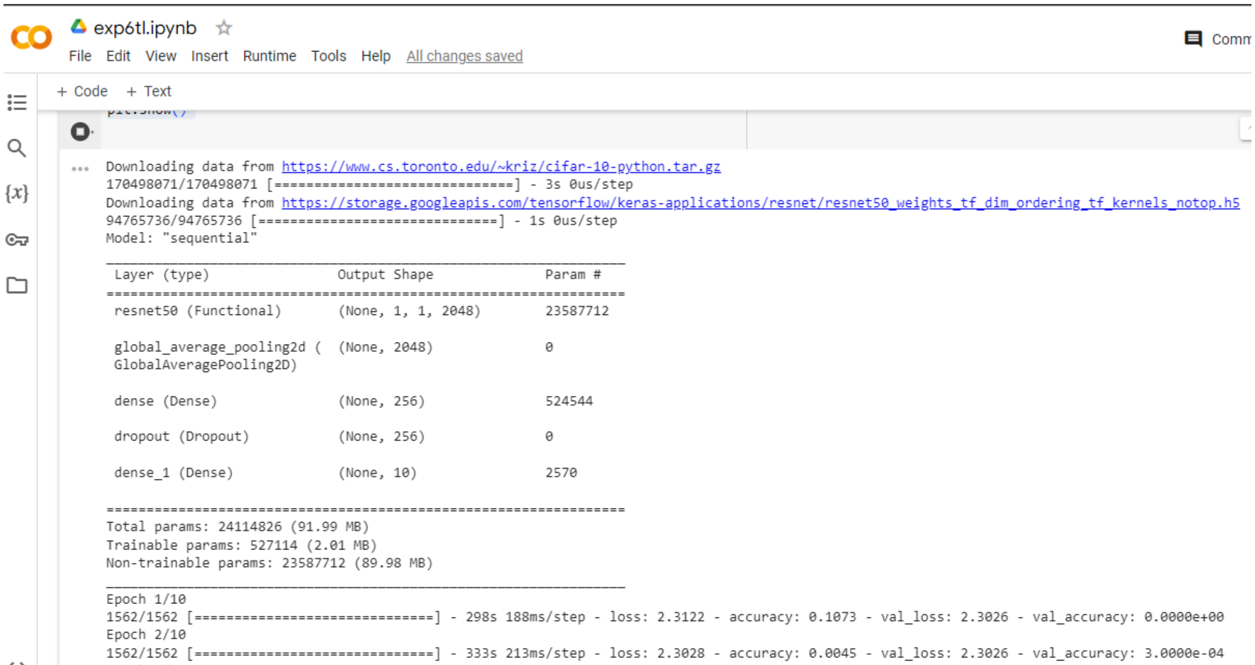
PROGRAM:

```
# Import necessary libraries
import tensorflow as tf
from tensorflow.keras import layers, models, datasets
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Load the CIFAR-10 dataset
(train_images, train_labels), (test_images, test_labels) = datasets.cifar10.load_data()

# Normalize pixel values to be between 0 and 1
train_images, test_images = train_images / 255.0, test_images / 255.0

# Define the ResNet50 model with pre-trained weights (excluding the top layer)
base_model = ResNet50(weights='imagenet', include_top=False, input_shape=(32, 32, 3))

# Freeze the layers of the pre-trained model
for layer in base_model.layers:
```

```
    layer.trainable = False

# Create a new model on top of the pre-trained model
model = models.Sequential([
    base_model,
    layers.GlobalAveragePooling2D(),
    layers.Dense(256, activation='relu'),
    layers.Dropout(0.5),
    layers.Dense(10, activation='softmax')
])

# Compile the model
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

# Display model summary
model.summary()

# Data augmentation to improve generalization
datagen = ImageDataGenerator(
    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'
)

# Fit the model with data augmentation
history = model.fit(datagen.flow(train_images, train_labels, batch_size=32),
                    steps_per_epoch=len(train_images) // 32, epochs=10,
                    validation_data=(test_images, test_labels))

# Evaluate the model on the test set
test_loss, test_acc = model.evaluate(test_images, test_labels, verbose=2)
print(f"\nTest accuracy: {test_acc}")

# Plot training history
import matplotlib.pyplot as plt

plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label='val_accuracy')
plt.xlabel('Epoch')
```

```
plt.ylabel('Accuracy')
plt.legend(loc='lower right')
plt.show()
```

OUTPUT:




```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 [==============================] - 3s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet50_weights_tf_dim_ordering_tf_kernels_notop.h5
94765736/94765736 [==============================] - 1s 0us/step
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 resnet50 (Functional)       (None, 1, 1, 2048)        23587712

 global_average_pooling2d (  (None, 2048)              0
 GlobalAveragePooling2D)

 dense (Dense)               (None, 256)               524544

 dropout (Dropout)           (None, 256)               0

 dense_1 (Dense)             (None, 10)                2570

=================================================================
Total params: 24114826 (91.99 MB)
Trainable params: 527114 (2.01 MB)
Non-trainable params: 23587712 (89.98 MB)
_________________________________________________________________
Epoch 1/10
1562/1562 [==============================] - 298s 188ms/step - loss: 2.3122 - accuracy: 0.1073 - val_loss: 2.3026 - val_accuracy: 0.0000e+00
Epoch 2/10
1562/1562 [==============================] - 333s 213ms/step - loss: 2.3028 - accuracy: 0.0045 - val_loss: 2.3026 - val_accuracy: 3.0000e-04
```

RESULT:

The above program has been executed successfully and its output has been verified.

Exp: 7

AUTOENCODER IN TENSORFLOW/KERAS.

AIM:

To implement an Autoencoder in TensorFlow/Keras.

ALGORITHM:

Step 1 : Start the process.

Step 2 : Open the Google colab

Step 3 : Import required libraries.

Step 4 : Import MNIST public dataset.

Step 5 : Build the encoder and decoder

Step 4 : Create model and start training

Step 5 : Test the model and print the accuracy.

Step 6: Close the application and stop the process.

PROGRAM:

```
from __future__ import division, print_function, absolute_import

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

```
# Import MNIST data
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets("/tmp/data/", one_hot=True)
```

```
# Training Parameters
learning_rate = 0.01
num_steps = 30000
batch_size = 256

display_step = 1000
examples_to_show = 10

# Network Parameters
num_hidden_1 = 256 # 1st layer num features
num_hidden_2 = 128 # 2nd layer num features (the latent dim)
num_input = 784 # MNIST data input (img shape: 28*28)
```

```
# tf Graph input (only pictures)
X = tf.placeholder("float", [None, num_input])

weights = {
    'encoder_h1': tf.Variable(tf.random_normal([num_input, num_hidden_1])),
    'encoder_h2': tf.Variable(tf.random_normal([num_hidden_1, num_hidden_2])),
    'decoder_h1': tf.Variable(tf.random_normal([num_hidden_2, num_hidden_1])),
    'decoder_h2': tf.Variable(tf.random_normal([num_hidden_1, num_input])),
}
biases = {
    'encoder_b1': tf.Variable(tf.random_normal([num_hidden_1])),
    'encoder_b2': tf.Variable(tf.random_normal([num_hidden_2])),
    'decoder_b1': tf.Variable(tf.random_normal([num_hidden_1])),
    'decoder_b2': tf.Variable(tf.random_normal([num_input])),
}
```

```
# Building the encoder
def encoder(x):
    # Encoder Hidden layer with sigmoid activation #1
    layer_1 = tf.nn.sigmoid(tf.add(tf.matmul(x, weights['encoder_h1']),
                                   biases['encoder_b1']))
    # Encoder Hidden layer with sigmoid activation #2
    layer_2 = tf.nn.sigmoid(tf.add(tf.matmul(layer_1, weights['encoder_h2']),
                                   biases['encoder_b2']))
    return layer_2


# Building the decoder
def decoder(x):
    # Decoder Hidden layer with sigmoid activation #1
    layer_1 = tf.nn.sigmoid(tf.add(tf.matmul(x, weights['decoder_h1']),
                                   biases['decoder_b1']))
    # Decoder Hidden layer with sigmoid activation #2
    layer_2 = tf.nn.sigmoid(tf.add(tf.matmul(layer_1, weights['decoder_h2']),
                                   biases['decoder_b2']))
    return layer_2

# Construct model
encoder_op = encoder(X)
decoder_op = decoder(encoder_op)
# Prediction
y_pred = decoder_op
# Targets (Labels) are the input data.
y_true = X
```

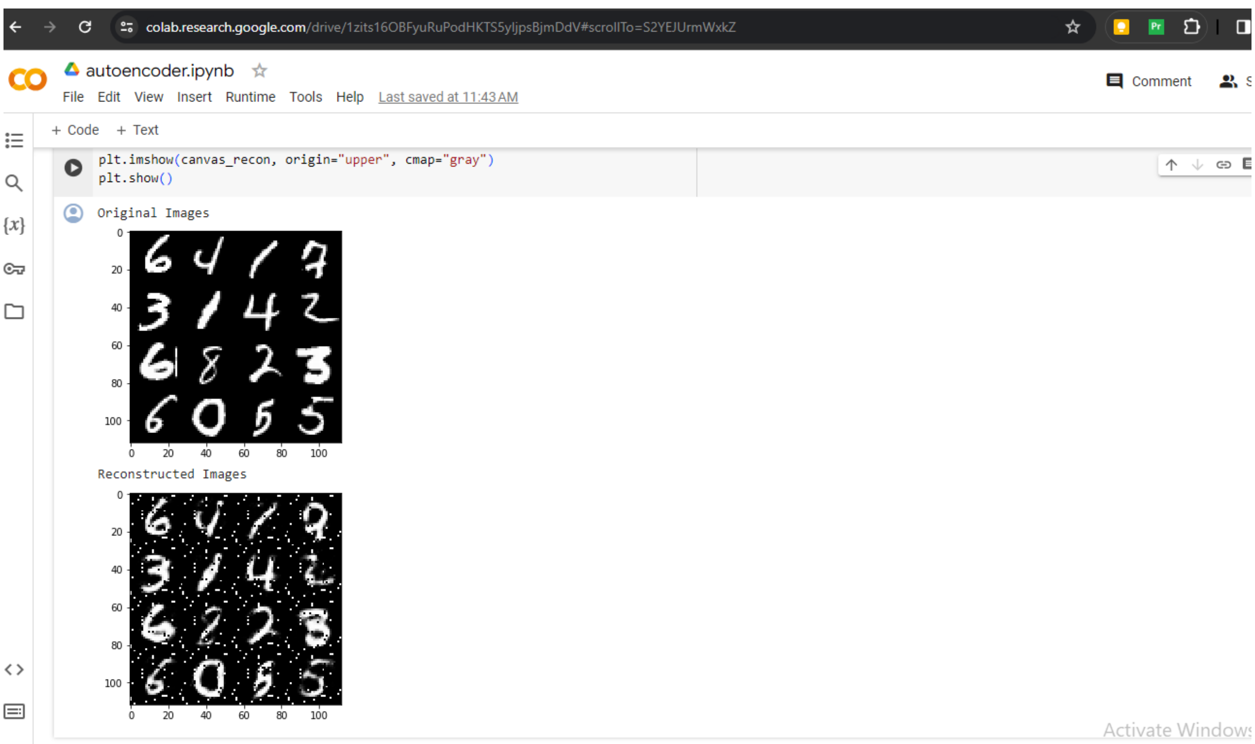
```
# Define loss and optimizer, minimize the squared error
loss = tf.reduce_mean(tf.pow(y_true - y_pred, 2))
optimizer = tf.train.RMSPropOptimizer(learning_rate).minimize(loss)

# Initialize the variables (i.e. assign their default value)
init = tf.global_variables_initializer()
```

```
# Start Training
# Start a new TF session
sess = tf.Session()

# Run the initializer
sess.run(init)

# Training
for i in range(1, num_steps+1):
    # Prepare Data
    # Get the next batch of MNIST data (only images are needed, not labels)
    batch_x, _ = mnist.train.next_batch(batch_size)

    # Run optimization op (backprop) and cost op (to get loss value)
    _, l = sess.run([optimizer, loss], feed_dict={X: batch_x})
    # Display logs per step
    if i % display_step == 0 or i == 1:
        print('Step %i: Minibatch Loss: %f' % (i, l))
```

```
# Testing
# Encode and decode images from test set and visualize their reconstruction.
n = 4
canvas_orig = np.empty((28 * n, 28 * n))
canvas_recon = np.empty((28 * n, 28 * n))
for i in range(n):
    # MNIST test set
    batch_x, _ = mnist.test.next_batch(n)
    # Encode and decode the digit image
    g = sess.run(decoder_op, feed_dict={X: batch_x})

    # Display original images
    for j in range(n):
        # Draw the generated digits
        canvas_orig[i * 28:(i + 1) * 28, j * 28:(j + 1) * 28] = batch_x[j].reshape([28, 28])
    # Display reconstructed images
    for j in range(n):
```

```
    # Draw the generated digits
    canvas_recon[i * 28:(i + 1) * 28, j * 28:(j + 1) * 28] = g[j].reshape([28, 28])

print("Original Images")
plt.figure(figsize=(n, n))
plt.imshow(canvas_orig, origin="upper", cmap="gray")
plt.show()

print("Reconstructed Images")
plt.figure(figsize=(n, n))
plt.imshow(canvas_recon, origin="upper", cmap="gray")
plt.show()
```

OUTPUT:



RESULT:

The above program has been executed successfully and its output has been verified.

Exp: 8

LSTM USING TENSORFLOW

AIM:

To implement LSTM (Long Short Term Memory) using Tensorflow and Keras.

ALGORITHM:

Step 1 : Start the process.

Step 2 : Open the Google colab

Step 3 : Import tensorflow library, Import Keras from Tensorflow

Step 4 : Generate a sample dataset of sequence of numbers.

Step 5 : Reshape the data for LSTM input

Step 6 : Create an LSTM model with 100 epochs.

Step 7 : Predict the next value in the sequence.

Step 8 : Close the application and stop the process.

PROGRAM:

```
# Import necessary libraries
import numpy as np
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense

# Generate some sample data (you can replace this with your dataset)
data = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
target = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])

# Reshape data for LSTM input
data = data.reshape(len(data), 1, 1)

# Create an LSTM model
model = Sequential()
model.add(LSTM(50, activation='relu', input_shape=(1, 1)))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mean_squared_error')

# Train the model
model.fit(data, target, epochs=100, batch_size=1)
```

OUTPUT:

```
1/1 [==============================] - 0s 217ms/step
[[11.893926]]
```

RESULT:

The above program has been executed successfully and its output has been verified.

Exp: 9

OPINION MINING IN RECURRENT NEURAL NETWORK

AIM:

To implement an Opinion Mining in Recurrent Neural network.

ALGORITHM:

Step 1 : Start the process.

Step 2 : Open the Google colab

Step 3 : Import required libraries.

Step 4 : Load the IMDB dataset

Step 5 : Build the RNN model

Step 6 : Compile the model

Step 7 : Train the model

Step 8 : Evaluate the model on test set

Step 9 : Plot training accuracy.

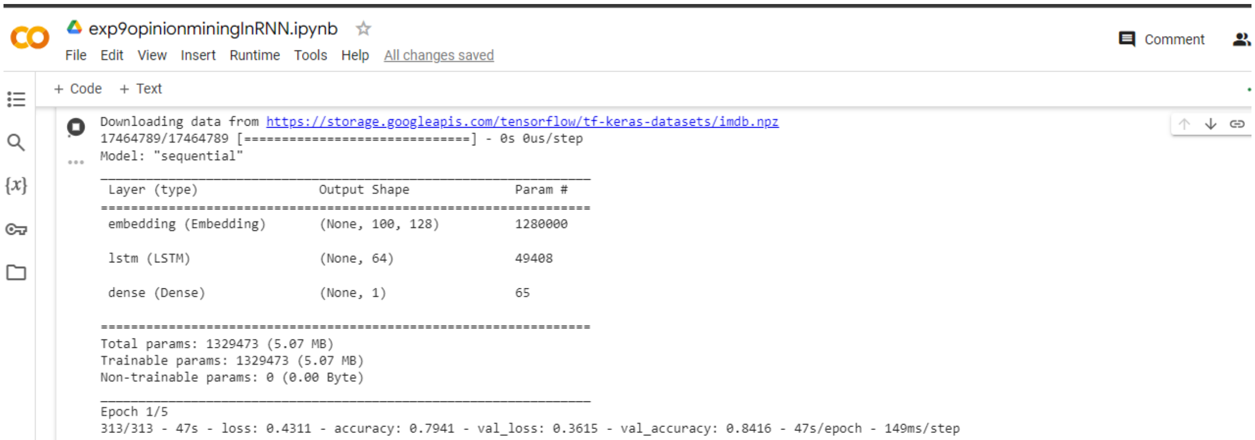
Step 10: Close the application and stop the process.

PROGRAM:

```
# Import necessary libraries
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

# Load the IMDB dataset
num_words = 10000  # Top 10,000 most frequent words
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=num_words)

# Pad sequences to a fixed length
max_len = 100
x_train = pad_sequences(x_train, maxlen=max_len)
x_test = pad_sequences(x_test, maxlen=max_len)
```

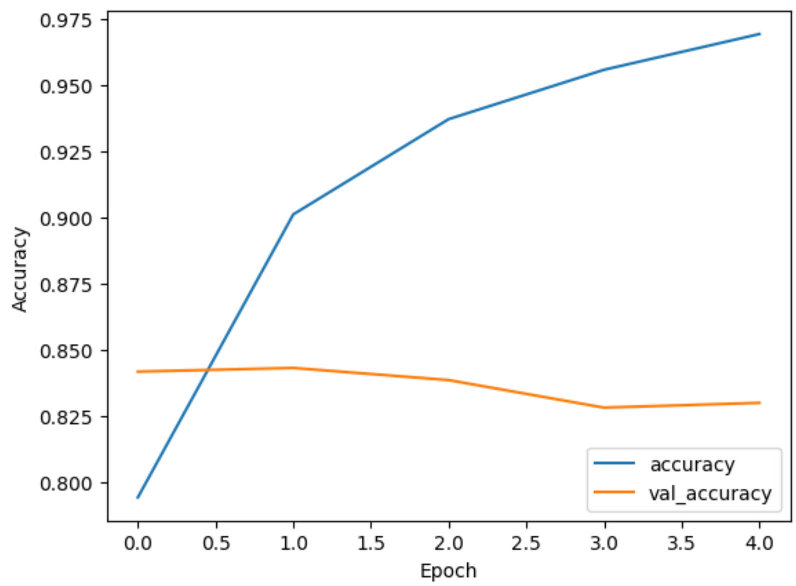
```python
# Build the RNN model
model = Sequential()
model.add(Embedding(input_dim=num_words, output_dim=128, input_length=max_len))
model.add(LSTM(units=64))
model.add(Dense(units=1, activation='sigmoid'))

# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Display model summary
model.summary()

# Train the model
batch_size = 64
epochs = 5
history = model.fit(x_train, y_train, batch_size=batch_size, epochs=epochs,
                    validation_split=0.2, verbose=2)

# Evaluate the model on the test set
test_loss, test_acc = model.evaluate(x_test, y_test, verbose=2)
print(f"\nTest accuracy: {test_acc}")

# Plot training history
import matplotlib.pyplot as plt

plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label='val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(loc='lower right')
plt.show()
```

OUTPUT:




```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/imdb.npz
17464789/17464789 [==============================] - 0s 0us/step
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 100, 128)          1280000

 lstm (LSTM)                 (None, 64)                49408

 dense (Dense)               (None, 1)                 65

=================================================================
Total params: 1329473 (5.07 MB)
Trainable params: 1329473 (5.07 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Epoch 1/5
313/313 - 47s - loss: 0.4311 - accuracy: 0.7941 - val_loss: 0.3615 - val_accuracy: 0.8416 - 47s/epoch - 149ms/step
```

```
Epoch 5/5
313/313 - 41s - loss: 0.0922 - accuracy: 0.9692 - val_loss: 0.5232 - val_accuracy: 0.8298 - 41s/epoch - 132ms/step
782/782 - 14s - loss: 0.5240 - accuracy: 0.8290 - 14s/epoch - 18ms/step

Test accuracy: 0.8289999961853027
```



RESULT:

The above program has been executed successfully and its output has been verified.

Exp: 10

OBJECT DETECTION USING CNN

AIM:

To implement object detection using CNN.

ALGORITHM:

Step 1 : Start the process.

Step 2 : Open the Google colab

Step 3 : Import required libraries.

Step 4 : Load CIFAR-10 dataset.

Step 5 : Split the dataset into training and validation sets

Step 6 : Define CNN model.

Step 7 : Train the model.

Step 8 : Evaluate the model performance on test dataset.

Step 9 : Plot the training and validation accuracy.

Step 10: Close the application and stop the process.

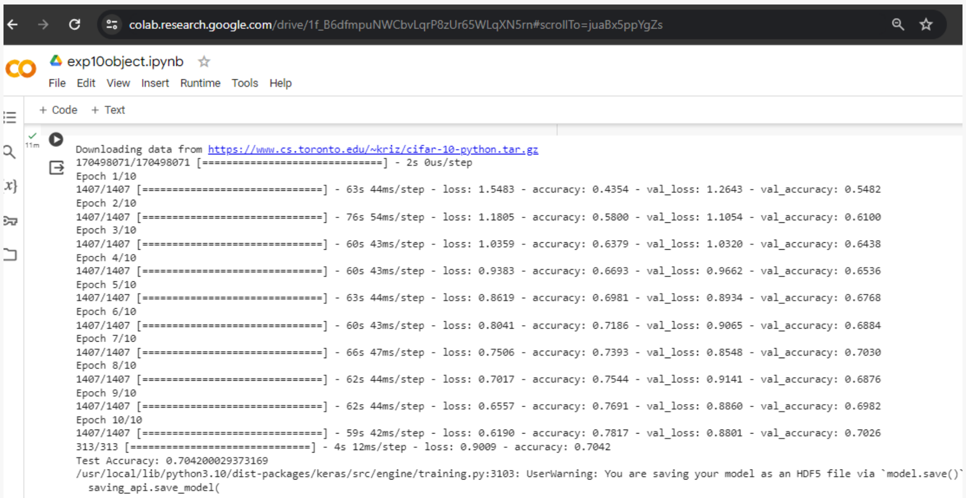
PROGRAM:

```
# Install TensorFlow 2.x (if not already installed)
!pip install -q tensorflow

import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.utils import to_categorical
from sklearn.model_selection import train_test_split

# 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_test = x_train / 255.0, x_test / 255.0
```

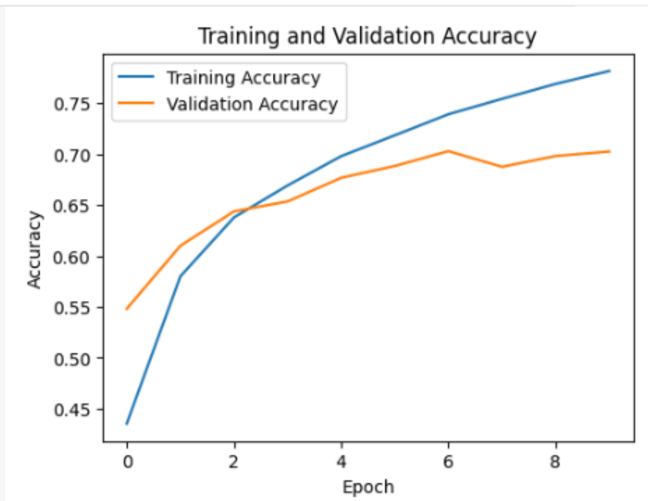
```
# One-hot encode the labels
y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)

# Split the dataset into training and validation sets
x_train, x_val, y_train, y_val = train_test_split(x_train, y_train, test_size=0.1, random_state=42)

# Define the CNN model
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))

# Compile the model
model.compile(optimizer='adam',
          loss='categorical_crossentropy',
          metrics=['accuracy'])

# Train the model
history = model.fit(x_train, y_train, epochs=10, validation_data=(x_val, y_val))

# Evaluate the model on the test set
test_loss, test_acc = model.evaluate(x_test, y_test)
print(f'Test Accuracy: {test_acc}')

# Plot training and validation accuracy
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
```

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

OUTPUT:



```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 [==============================] - 2s 0us/step
Epoch 1/10
1407/1407 [==============================] - 63s 44ms/step - loss: 1.5483 - accuracy: 0.4354 - val_loss: 1.2643 - val_accuracy: 0.5482
Epoch 2/10
1407/1407 [==============================] - 76s 54ms/step - loss: 1.1805 - accuracy: 0.5800 - val_loss: 1.1054 - val_accuracy: 0.6100
Epoch 3/10
1407/1407 [==============================] - 60s 43ms/step - loss: 1.0359 - accuracy: 0.6379 - val_loss: 1.0320 - val_accuracy: 0.6438
Epoch 4/10
1407/1407 [==============================] - 60s 43ms/step - loss: 0.9383 - accuracy: 0.6693 - val_loss: 0.9662 - val_accuracy: 0.6536
Epoch 5/10
1407/1407 [==============================] - 63s 44ms/step - loss: 0.8619 - accuracy: 0.6981 - val_loss: 0.8934 - val_accuracy: 0.6768
Epoch 6/10
1407/1407 [==============================] - 60s 43ms/step - loss: 0.8041 - accuracy: 0.7186 - val_loss: 0.9065 - val_accuracy: 0.6884
Epoch 7/10
1407/1407 [==============================] - 66s 47ms/step - loss: 0.7506 - accuracy: 0.7393 - val_loss: 0.8548 - val_accuracy: 0.7030
Epoch 8/10
1407/1407 [==============================] - 62s 44ms/step - loss: 0.7017 - accuracy: 0.7544 - val_loss: 0.9141 - val_accuracy: 0.6876
Epoch 9/10
1407/1407 [==============================] - 62s 44ms/step - loss: 0.6557 - accuracy: 0.7691 - val_loss: 0.8860 - val_accuracy: 0.6982
Epoch 10/10
1407/1407 [==============================] - 59s 42ms/step - loss: 0.6190 - accuracy: 0.7817 - val_loss: 0.8801 - val_accuracy: 0.7026
313/313 [==============================] - 4s 12ms/step - loss: 0.9009 - accuracy: 0.7042
Test Accuracy: 0.704200029373169
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103: UserWarning: You are saving your model as an HDF5 file via `model.save()`
  saving_api.save_model(
```



RESULT:

The above program has been executed successfully and its output has been verified.