Experiment: 1 Simple Artificial neural network

Aim:

To build a simple Artificial Neural Network (ANN) classification using customer churn dataset.

Software Requirements:

➢ Google Colab

Program:

#Import Libraries

```
import numpy as np
import pandas as pd
import tensorflow as tf


from google.colab import drive
drive.mount('/content/drive')
```

 # Load dataset

```
data = pd.read_csv('Churn_Modelling.csv')
data
```

 # Extract features and label

```
X = data.iloc[:, 3:-1].values
print(X)
Y = data.iloc[:, -1].values
print(Y)
```

 # Encode categorical data

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
X[:, 2] = np.array(le.fit_transform(X[:, 2]))

from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
ct = ColumnTransformer(transformers=[('encoder', OneHotEncoder(), [1])],
remainder='passthrough')
X = np.array(ct.fit_transform(X))
```

```
# Split dataset into training and testing sets
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=0)

# Feature scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Build the ANN model
ann = tf.keras.models.Sequential()
ann.add(tf.keras.layers.Dense(units=6, activation='relu'))
ann.add(tf.keras.layers.Dense(units=1, activation='sigmoid'))

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

# Train the model
ann.fit(X_train, Y_train, batch_size=32, epochs=100)

# Predict a new result
print(ann.predict(sc.transform([[0,1,1,619,0,42,2,60000,1,1,1,101348]])) > 0.5)
```

➔ **Output**

```
[[ True]]
```

Result:
The experiment successfully resulted in a Simple artificial neural network capable of predicting customer churn, achieving accurate classification .

Experiment: 2 Single Layer Perceptron

Aim:
To implement a Single Layer Perceptron using the Iris dataset for binary classification and evaluate its performance.

Software Requirements:

- ➢ Google Colab

Program:

#Import Libraries

```
from sklearn.datasets import load_iris
from sklearn.linear_model import Perceptron
from sklearn.metrics import accuracy_score

# Load the Iris dataset
iris = load_iris()

# Select features and target
x = iris.data[:, (2, 3)]  # Petal length and petal width
y = (iris.target == 0).astype(int)  # Binary classification: Setosa or not

# Initialize the Perceptron
ptron = Perceptron(random_state=42)

# Train the model
ptron.fit(x, y)

# Make predictions
y_pred = ptron.predict(x)
print(y_pred)

# Evaluate the model
print(f'Accuracy Score: {accuracy_score(y, y_pred)}')
```

➔ **Output**

```
Accuracy Score: 1.0
```

Result :
The experiment successfully demonstrated the working of a Single Layer Perceptron on the Iris dataset, achieving accurate binary classification

Experiment: 3 Gradient Descent

Aim:
To implement the Gradient Descent optimization algorithm for minimizing Errors

Software Requirements:

- ➢ Google Colab

Program:

#Import Libraries

```
import numpy as np
```

 # Gradient Descent Function

```
def gradient_descent(gradient, start, learning_rate, iteration=50, tol=1e-06):
    vector = start
    for _ in range(iteration):
        diff = -learning_rate * gradient(vector)
        if np.all(np.abs(diff) <= tol):
            break
        vector += diff
    return vector
```

 # Testing The Function

```
from typing_extensions import LiteralString
print(gradient_descent(gradient = lambda v: 4* v**3 - 10* v-3,start=0,learning_rate=0.2 ))
```

Result:
The experiment successfully demonstrated the use of the Gradient Descent algorithm

➔ **Output**

```
-1.4207567437458342
```

Experiment: 4 Stochastic Gradient Descent-Classifier

Aim:
To implement and evaluate a linear classification model using Stochastic Gradient Descent (SGD) with elastic net regularization.

Software Requirements:

- Google Colab

Program:

```
import numpy as np

from sklearn import linear_model

# Sample dataset

x = np.array([[-1, -1], [-2, -1], [1, 1], [2, 1]])

y = np.array([1, 1, 2, 2])

# Creating SGDClassifier model with elasticnet regularization

sdg_class = linear_model.SGDClassifier(max_iter=1000, tol=1e-3, penalty="elasticnet")

# Fitting model on the original data

sdg_class.fit(x, y)

# Making prediction

print('Prediction is:', sdg_class.predict([[2, 4]]))

# Model parameters

print('Weight vector(s):', sdg_class.coef_)

print('Intercept:', sdg_class.intercept_)

print('Distance to HyperPlane:', sdg_class.decision_function([[2, 4]]))
```

➔ **Output**

```
Prediction is : [2]

Weight vector(s): [[9.77200712 9.77200712]]

Intercpet:  [-10.]

Distance to HyperPlane :  [48.63204273]
```

Result:
The Stochastic Gradient Descent classifier is able to make predictions and display the model’s weight and intercept values and successfully verified.

Experiment: 5 Fundamentals Of TensorFLow

Aim:
To understand and perform basic operations in TensorFlow, including constants, variables, reshaping, matrix multiplication, and concatenation.

Software Requirements:

- Google Colab

Program:

```
import tensorflow as tf

print("TensorFlow version:", tf.__version__)
```

- **Output** : `TensorFlow version: 2.18.0`

 # Check if Eager Execution is enabled

```
if(tf.executing_eagerly()):
  print("Eager Execution Enabled")
else:
  print("Eager Execution Not Available. Upgrade TensorFlow 2.0.0+")
```

- **Output**: `Eager Execution Enabled`

 # Constants

```
con1 = tf.constant([[1.4, 2.1], [3, 4.7]])

con2 = tf.constant([[5], [2]])

con3 = tf.constant([[5, 4], [2, 1]])

con4 = tf.constant([[5, 4, 2], [2, 1, 2]])

print("T1:", con1)

print("T2:", con2)

print("T3:", con3)
```

➔ **Output:**
```
T1: tf.Tensor(
[[1.4 2.1]
 [3.  4.7]], shape=(2, 2), dtype=float32)
T2: tf.Tensor(
[[5]
 [2]], shape=(2, 1), dtype=int32)
T3: tf.Tensor(
[[5 4]
 [2 1]], shape=(2, 2), dtype=int32)
```

 # String tensor

T1 = tf.constant([["a"], ["b"]], dtype=tf.string)

print(T1)

➔ **Output:**
```
<tf.Tensor: shape=(2, 1), dtype=string, numpy=
array([[b'a'],
       [b'b']], dtype=object)>
```

 # Transpose

trans = tf.transpose(con1)

print("con1 Transpose:", trans)

➔ **Output :**
```
con1 Transpose: tf.Tensor(
[[1.4 3. ]
 [2.1 4.7]], shape=(2, 2), dtype=float32)
```

 # Type casting

con3 = tf.cast(con1, tf.float32)

con4 = tf.cast(con2, tf.float32)

➔ **Output :**
```
<tf.Tensor: shape=(2, 1), dtype=float32, numpy=
array([[5.],
       [2.]], dtype=float32)>
```

 # Element-wise multiplication

mul_elements = tf.multiply(con3, con4)

print("Mul Elements:", mul_elements)

➔ **Output :**
```
Mul Elements: tf.Tensor(
[[ 7.  10.5]
 [ 6.   9.4]], shape=(2, 2), dtype=float32)
```

 # Matrix multiplication

mul_mat = tf.matmul(con3, con4)

print("Mul Matrix:", mul_mat)

➔ **Output** :
```
Mul Matrix: tf.Tensor(
[[11.2]
 [24.4]], shape=(2, 1), dtype=float32)
```

 # Reshaping

reshape_con1 = tf.reshape(tensor=con1, shape=[1, 4])

print("Reshape Con1:", reshape_con1)

➔ **Output** :
```
Reshape Con1: tf.Tensor([[1.4 2.1 3.  4.7]], shape=(1, 4),
dtype=float32)
```

 # Create a 6x6 tensor matrix

con7 = tf.constant([[1.4, 2.1, 2.7, 2.8, 2.7, 8.7],[3.1, 4.7, 5.5, 1.4, 2.1, 2.7],

[4.5, 2.5, 3.1, 4.7, 5.5, 5.6], [4.2, 2.2, 4.5, 2.5, 3.1, 4.7],

[6.5, 7.5, 2.5, 3.1, 4.7, 5.5], [8.2, 9.2, 4.5, 2.5, 3.1, 4.7]])

print(con7)

➔ **Output** :
```
tf.Tensor([[1.4 2.1 2.7 2.8 2.7 8.7]
 [3.1 4.7 5.5 1.4 2.1 2.7]
 [4.5 2.5 3.1 4.7 5.5 5.6]
 [4.2 2.2 4.5 2.5 3.1 4.7]
 [6.5 7.5 2.5 3.1 4.7 5.5]
 [8.2 9.2 4.5 2.5 3.1 4.7]], shape=(6, 6), dtype=float32)
```

 # Reshape the matrix with Total Element

reshape_con7 = tf.reshape(tensor=con7, shape=[6, 6])

print("Reshape con7:", reshape_con7)

➔ **Output** :
```
Reshape Con1: tf.Tensor(
[[1.4 2.1 2.7 2.8 2.7 8.7]
 [3.1 4.7 5.5 1.4 2.1 2.7]
 [4.5 2.5 3.1 4.7 5.5 5.6]
 [4.2 2.2 4.5 2.5 3.1 4.7]
 [6.5 7.5 2.5 3.1 4.7 5.5]
 [8.2 9.2 4.5 2.5 3.1 4.7]], shape=(6, 6), dtype=float32)
```

 # Identity matrix

id_int = tf.eye(num_rows=3, num_columns=3, dtype=tf.int32)

print("Identity matrix (int):", id_int)

➔ **Output :**
```
identity matrix: tf.Tensor(
[[1 0 0]
 [0 1 0]
 [0 0 1]], shape=(3, 3), dtype=int32)
```

id_float = tf.eye(num_rows=3, num_columns=3, dtype=tf.float32)

print("Identity matrix (float):", id_float)

➔ **Output :**
```
identity matrix: tf.Tensor(
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]], shape=(3, 3), dtype=float32)
```

 # Constant tensor

con_ten = tf.constant([[4, 1], [3, 4]])

print(con_ten)

➔ **Output :**
```
tf.Tensor(
[[4 1]
 [3 4]], shape=(2, 2), dtype=int32)
```

 # Variable tensor

new_var_ten = tf.Variable([[4, 1], [3, 4]])

print(new_var_ten)

➔ **Output :**
```
tf.Tensor(
[[4 1]
 [3 4]], shape=(2, 2), dtype=int32)
```

n_var_ten = new_var_ten.assign([[4, 1], [3, 4]])

print(n_var_ten)

➔ **Output :**
```
<tf.Variable 'UnreadVariable' shape=(2, 2) dtype=int32, numpy=
array([[4, 1],
       [3, 4]], dtype=int32)>
```

 # Concatenation

row_con = tf.concat(values=(con_ten, new_var_ten), axis=0)

print(row_con)

➔ **Output** :
```
tf.Tensor(
[[4 1]
 [3 4]
 [4 1]
 [3 4]], shape=(4, 2), dtype=int32)
```

col_con = tf.concat(values=(n_var_ten, new_var_ten), axis=1)

print(col_con)

➔ **Output** :
```
tf.Tensor(
[[4 1 4 1]
 [3 4 3 4]], shape=(2, 4), dtype=int32)
```

 # Zeros and Ones

zeros = tf.zeros(shape=(3, 4), dtype=tf.int32)

print(zeros)

➔ **Output** :
```
tf.Tensor(
[[0 0 0 0]
 [0 0 0 0]
 [0 0 0 0]], shape=(3, 4), dtype=int32)
```

ones = tf.ones(shape=(3, 4), dtype=tf.int32)

print(ones)

➔ **Output** :
```
tf.Tensor(
[[1 1 1 1]
 [1 1 1 1]
 [1 1 1 1]], shape=(3, 4), dtype=int32)
```

Result:
The experiment successfully demonstrated various fundamental operations in TensorFlow including tensor creation, reshaping, matrix operations, concatenation, and use of constants and variables.

Experiment: 6 Working with Keras

Aim:
To understand and explore the basic functionalities of the Keras library, including loading datasets, visualizing images, text vectorization, and normalization.

Software Requirements:

- ➢ Google Colab

Program:

 # Importing libraries

```
from tensorflow import keras

import numpy as np

import tensorflow as tf

from matplotlib import pyplot as plt

from keras.datasets import cifar10, mnist, fashion_mnist

from tensorflow.keras.layers import TextVectorization, Normalization
```

 # Load and visualize CIFAR-10

```
tf.keras.datasets

data = cifar10.load_data()
```

#Data Split

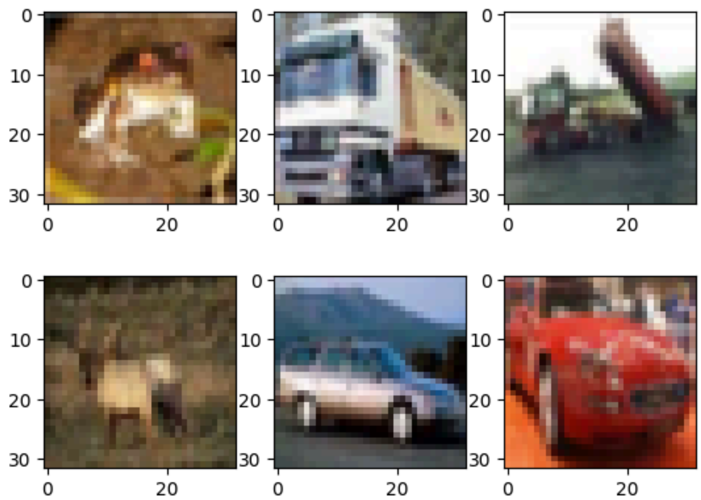
```
(trainX, trainY), (testX, testY) = cifar10.load_data()

print('Train: X=%s, y=%s' % (trainX.shape, trainY.shape))

print('Test: X=%s, y=%s' % (testX.shape, trainY.shape))
```

#Visualizing Data

```
for i in range(6):

  plt.subplot(230+1+i)

  plt.imshow(trainX[i])

plt.show()

plt.imshow(trainX[88])
```

print("Label:", trainY[88])

➔ **Output :**



 # Load and visualize MNIST

from keras.datasets import mnist

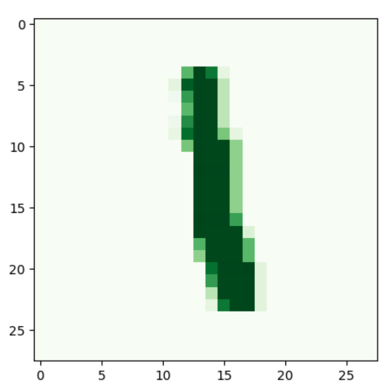
data = mnist.load_data()

(X_train, Y_train), (X_test, Y_test) = mnist.load_data()

plt.imshow(X_train[6], cmap='Greens')

plt.show()

print(Y_train[6])



 # Load and visualize Fashion MNIST

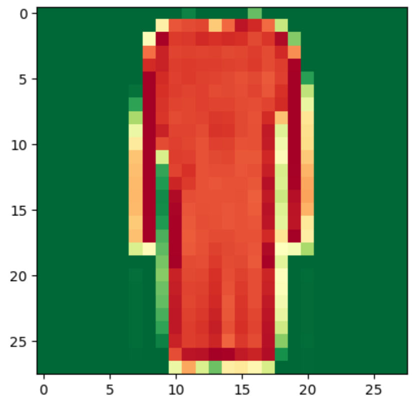
from keras.datasets import fashion_mnist

(x_train, y_train), (x_test, y_test) = fashion_mnist.load_data()

plt.imshow(x_train[25], cmap='RdYlGn_r')

plt.show()

print("Label:", y_train[25])



 # Text Vectorization

Text_data = np.array([["I am Vikki"], ["Visca Barca"], ["I am a fan of Sports"]])

Text_data

➔ **Output :**

```
array([['Iam Vikki'],
    ['Visca Barca'],
    ['I am a fan of Sports']], dtype='<U20')
```

#Applying Text Vectorization

vectorizer = TextVectorization(output_mode='int')

vectorizer.adapt(Text_data)

integer_data = vectorizer(Text_data)

print("Vectorized Data:", integer_data.numpy())

print("Vocabulary:", vectorizer.get_vocabulary())

➔ **Output :**

```
['', '[UNK]',
 np.str_('vikki'),
 np.str_('sports'),
 np.str_('of'),
 np.str_('iam'),
 np.str_('i'),
 np.str_('fan'),
 np.str_('Barca'),
 np.str_('am'),
 np.str_('Visca'),
 np.str_('a')]
```

 # Normalization

data = np.random.randint(0, 256, size=(64, 200, 200, 3)).astype('float32')

print("Original Mean:", np.mean(data))

➔ **Output** : `Mean: 127.53194`

print("Original Variance:", np.var(data))

➔ **Output** : `Variance: 5459.2676`

Normalizer = Normalization(axis=None)

Normalizer.adapt(data)

Normalized = Normalizer(data)

print("Normalized Mean:", np.mean(Normalized))

➔ **Output** : `Mean: 2.853652e-07`

print("Normalized Variance:", np.var(Normalized))

➔ **Output** : `Variance: 0.9999998`

Result:
The experiment successfully demonstrated key features of Keras such as dataset loading, image visualization, text vectorization, and data normalization using TensorFlow layers.

Experiment: 7 Logistic Regression

Aim:
To implement Logistic Regression using TensorFlow for multi-class classification on the Iris dataset.

Software Requirements:

- ➢ Google Colab

Program:

```
# Importing libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import tensorflow as tf
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split

# Load dataset
data = load_iris()
x = data["data"]
y = data["target"]

# Create a DataFrame for reference
dataset = pd.DataFrame(data=np.concatenate((x, y.reshape(-1, 1)), axis=1),
                       columns=["Sepal length", "Sepal width", "Petal length", "Petal width", "target"])
print(dataset.head())

# Split dataset
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.1, shuffle=True)
```

```
# Build logistic regression model using Keras API
input = tf.keras.Input(shape=(4,))
X = tf.keras.layers.Dense(3, activation='sigmoid')(input)
model = tf.keras.models.Model(input, X)

# Compile the model
model.compile(optimizer=tf.keras.optimizers.SGD(learning_rate=0.01),
        loss=tf.keras.losses.sparse_categorical_crossentropy,
        metrics=["accuracy"])

# Train the model
train = model.fit(x_train, y_train, validation_data=(x_test, y_test), epochs=200)

# Prediction for a new data point
New_data = [5.7, 2.8, 4.1, 1.3]
y_pred = model.predict(np.array(New_data).reshape(1, -1))
print("Prediction (Raw):", y_pred)
print("Predicted Class:", np.argmax(y_pred))
```

Output:

```
predicted: [[0.0595241  0.31964937 0.28121248]]
Predicted Class: 1
```

Result:

The logistic regression model was successfully implemented using TensorFlow. It accurately classified the Iris dataset into one of the three species.

Experiment: 8 Multi-Layer Perceptron

Aim:

To build and evaluate a Multi-Layer Perceptron (MLP) model using TensorFlow/Keras for classification of the Iris dataset.

Software Requirements:

- Google Colab

Program:

```
# Importing libraries
from numpy import argmax
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
import numpy as np
import pandas as pd

# Keras Model
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense

# Load Iris dataset
iris = load_iris()
x = iris.data.astype('float32')
y = iris.target

# Create DataFrame
iris_df = pd.DataFrame(x, columns=iris.feature_names)
print("Iris Dataset Preview:")
print(iris_df.head())

# Train-Test Split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.25)
```

```
# Model input feature size
n_features = x_train.shape[1]
print("Number of Features:", n_features)

# Build MLP model
model = Sequential()
model.add(Dense(10, activation='relu', kernel_initializer='he_normal',
input_shape=(n_features,)))
model.add(Dense(4, activation='relu', kernel_initializer='he_normal'))
model.add(Dense(3, activation='softmax'))  # 3 output classes

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

# Train the model
model.fit(x_train, y_train, epochs=150, batch_size=32)

# Evaluate the model
loss, accuracy = model.evaluate(x_test, y_test)
print(f"Test Accuracy: {accuracy * 100:.2f}%")
print(f"Test Loss: {loss:.4f}")

# Prediction on new data
new_data = np.array([  [5.1, 3.5, 1.4, 0.2], [6.4, 3.1, 5.5, 1.8], [6.7, 3.3, 5.7, 2.5]
                      [4.6, 3.4, 1.4, 0.3],[4.8, 3.4, 1.9, 0.2],[5.6, 2.5, 3.9, 1.1]])

y_pred = model.predict(new_data)
print("Predictions (probabilities):")
print(y_pred)
print("Predicted Classes:")
```

print(np.argmax(y_pred, axis=1))

Output :

```
Test Accuracy: 100.00%
```

```
Test Loss: 0.0219
```

Output :

```
Model Evaluation Score: 1.0
```

Output :

```
Prediction: [[9.9630344e-01 3.6965355e-03 3.2302991e-13]
 [2.3607190e-09 3.3192713e-02 9.6680731e-01]
 [1.3617095e-10 7.7143717e-03 9.9228561e-01]
 [9.9582160e-01 4.1784509e-03 4.3446578e-13]
 [9.9535805e-01 4.6419362e-03 5.6031872e-13]
 [6.1574858e-04 9.9901319e-01 3.7104887e-04]]
Prediction Class: 16
```

Result:
The Multi-Layer Perceptron (MLP) model trained using the Iris dataset and successfully Verified.

Experiment: 9 Image Recognition CNN

Aim:

To develop an image classification model using Convolutional Neural Networks (CNN) on the Fashion MNIST dataset.

Software Requirements:

- ➢ Google Colab

Program:

```
# Importing libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow import keras
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report

# Load the Fashion MNIST dataset
fashion = keras.datasets.fashion_mnist
(x_train_sp, y_train_sp), (x_test_sp, y_test_sp) = fashion.load_data()

# Fashion item labels
item_names = ["T-Shirt/Top", "Trouser", "Pullover", "Dress", "Coat",
              "Sandal", "Shirt", "Sneaker", "Bag", "Ankle Boot"]

# Reshape the dataset to add the channel dimension (1 for grayscale)
x_train_sp = x_train_sp.reshape((60000, 28, 28, 1))
x_test_sp = x_test_sp.reshape((10000, 28, 28, 1))

# Normalize the pixel values to range 0–1
x_train_norm = x_train_sp / 255.0
x_test_norm = x_test_sp / 255.0
```

```
# Split validation data from training data
x_validate, x_train = x_train_norm[:5000], x_train_norm[5000:]

y_validate, y_train = y_train_sp[:5000], y_train_sp[5000:]

x_test = x_test_norm

# Set random seed for reproducibility
tf.random.set_seed(42)

# Build CNN model
model = keras.models.Sequential([
  keras.layers.Conv2D(filters=32, kernel_size=(3,3), activation="relu",
input_shape=[28,28,1]),
  keras.layers.MaxPooling2D(pool_size=(2,2)),
  keras.layers.Flatten(),
  keras.layers.Dense(300, activation="relu"),
  keras.layers.Dense(100, activation="relu"),
  keras.layers.Dense(10, activation="softmax")
])

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

# Train the model
history = model.fit(x_train, y_train, epochs=10, batch_size=32,
           validation_data=(x_validate, y_validate))
```

 # Evaluate model on test data

test_loss, test_accuracy = model.evaluate(x_test, y_test_sp)

print(f"Test Accuracy: {test_accuracy * 100:.2f}%")

 # Predicting on test data

y_pred = model.predict(x_test)

y_pred_classes = np.argmax(y_pred, axis=1)

 # Confusion matrix and classification report

print("\nClassification Report:\n")

print(classification_report(y_test_sp, y_pred_classes, target_names=item_names))

➔ **Output** :

```
accuracy: 0.91527 - loss: 0.2513
Test Accuracy: 89.56%
```

```
Classification Report:
             precision    recall  f1-score   support
 T-Shirt/Top       0.40      0.87      0.55      1000
     Trouser       0.00      0.00      0.00      1000
    Pullover       0.31      0.75      0.44      1000
       Dress       0.16      0.81      0.27      1000
        Coat       0.00      0.00      0.00      1000
      Sandal       0.00      0.00      0.00      1000
       Shirt       0.00      0.00      0.00      1000
    Sneakers       0.70      0.24      0.36      1000
         Bag       0.00      0.00      0.00      1000
  Ankle Boot       1.00      0.00      0.00      1000
    accuracy                           0.27     10000
   macro avg       0.26      0.27      0.16     10000
weighted avg       0.26      0.27      0.16     10000
```

Result:

The CNN model was successfully built and trained using the Fashion MNIST dataset. It learned to recognize different clothing items and achieved good accuracy on the test data.

Experiment: 10 Transfer Learning for Audio Classification

Aim:
To perform audio classification using Transfer Learning with the YAMNet model, by extracting audio embeddings and training a custom classifier to classify animal sounds (dog and cat) from the ESC-50 dataset.

Software Requirements:

- ➢ Google Colab

Program:

 # Importing libraries

```
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import tensorflow as tf
import tensorflow_hub as hub
import tensorflow_io as tfio
from IPython import display
```

 # Load YAMNet model from TensorFlow Hub

```
yamnet_model_handle = "https://tfhub.dev/google/yamnet/1"
yamnet_model = hub.load(yamnet_model_handle)
```

 # Load test audio

```
testing_wav_file_name = tf.keras.utils.get_file('miaow_16k.wav',
   'https://storage.googleapis.com/audioset/miaow_16k.wav',
   cache_dir='./', cache_subdir='test_data')
```

 # Function to load and resample audio

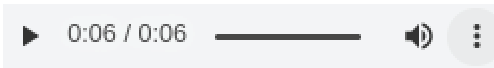
```
def load_wav_16k_mono(filename):
   file_contents = tf.io.read_file(filename)
```

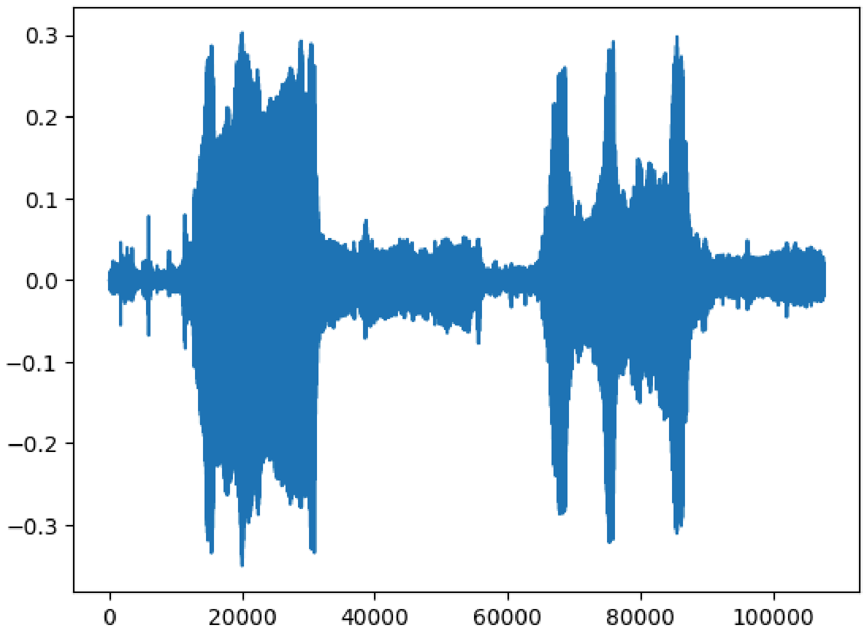
```python
    wav, sample_rate = tf.audio.decode_wav(file_contents, desired_channels=1)
    wav = tf.squeeze(wav, axis=-1)
    sample_rate = tf.cast(sample_rate, dtype=tf.int64)
    wav = tfio.audio.resample(wav, rate_in=sample_rate, rate_out=16000)
    return wav
testing_wav_data = load_wav_16k_mono(testing_wav_file_name)

# Play audio
display.Audio(testing_wav_data, rate=16000)

# Load class names
class_map_path = yamnet_model.class_map_path().numpy().decode('utf-8')
class_names = list(pd.read_csv(class_map_path)['display_name'])

# Run YAMNet prediction
scores, embeddings, spectrogram = yamnet_model(testing_wav_data)
class_scores = tf.reduce_mean(scores, axis=0)
top_class = tf.math.argmax(class_scores)
print(f"The main sound is: {class_names[top_class]}")
print(f"The embeddings shape: {embeddings.shape}")


# Load ESC-50 dataset
_ = tf.keras.utils.get_file('ESC-50.zip',
    'https://github.com/karoldvl/ESC-50/archive/master.zip',
    cache_dir='./', cache_subdir='datasets', extract=True)

# Read metadata and filter dog and cat
esc50_csv = './datasets/ESC-50-master/meta/esc50.csv'
base_data_path = './datasets/ESC-50-master/audio/'
pd_data = pd.read_csv(esc50_csv)
```

```
my_classes = ['dog', 'cat']
map_class_to_id = {'dog': 0, 'cat': 1}
filtered_pd = pd_data[pd_data.category.isin(my_classes)]
filtered_pd = filtered_pd.assign(target=filtered_pd['category'].map(map_class_to_id))
filtered_pd['filename'] = filtered_pd['filename'].apply(lambda name:
os.path.join(base_data_path, name))

# Prepare dataset
filesnames = filtered_pd['filename']
targets = filtered_pd['target']
folds = filtered_pd['fold']
main_ds = tf.data.Dataset.from_tensor_slices((filesnames, targets, folds))

def load_wav_for_map(filename, label, fold):
  return load_wav_16k_mono(filename), label, fold
main_ds = main_ds.map(load_wav_for_map)

# Extract embeddings from YAMNet
def extract_embedding(wav_data, label, fold):
  scores, embeddings, spectrogram = yamnet_model(wav_data)
  num_embeddings = tf.shape(embeddings)[0]
  return embeddings, tf.repeat(label, num_embeddings), tf.repeat(fold, num_embeddings)

main_ds = main_ds.map(extract_embedding).unbatch()
cached_ds = main_ds.cache()

# Split dataset
train_ds = cached_ds.filter(lambda emb, label, fold: fold < 4).map(lambda emb, label, fold:
(emb, label))
val_ds = cached_ds.filter(lambda emb, label, fold: fold == 4).map(lambda emb, label, fold:
(emb, label))
test_ds = cached_ds.filter(lambda emb, label, fold: fold == 5).map(lambda emb, label, fold:
```

```
(emb, label))

# Batch and prefetch
train_ds = train_ds.shuffle(1000).batch(32).prefetch(tf.data.AUTOTUNE)
val_ds = val_ds.batch(32).prefetch(tf.data.AUTOTUNE)
test_ds = test_ds.batch(32).prefetch(tf.data.AUTOTUNE)

# Build and train model
my_model = tf.keras.Sequential([
    tf.keras.layers.Input(shape=(1024,), name='input_embedding'),
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dense(len(my_classes))
], name='my_model')

my_model.compile(loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
                 optimizer='adam',
                 metrics=['accuracy'])

callback = tf.keras.callbacks.EarlyStopping(monitor='loss', patience=5,
restore_best_weights=True)

history = my_model.fit(train_ds, epochs=20, validation_data=val_ds, callbacks=[callback])

# Evaluate on test set
loss, accuracy = my_model.evaluate(test_ds)
print("Loss:", loss)
print("Accuracy:", accuracy)

# Predict new sound class
scores, embeddings, spectrogram = yamnet_model(testing_wav_data)
result = my_model(embeddings).numpy()===
inferred_class = my_classes[result.mean(axis=0).argmax()]
print(f"The main sound is: {inferred_class}")
```

➔ **Output :**






```
Loss:  0.30093947052955 63
```

```
Accuracy:  0.893750011920929
```

```
The main sound is: cat
```

Result:

The audio classification model was successfully built using transfer learning with YAMNet.

Experiment: 11 Stock Prediction Using LSTM

Aim:

To develop a deep learning model using LSTM (Long Short-Term Memory) for predicting Tesla stock prices based on historical data.

Software Requirements:

- ➢ Google Colab

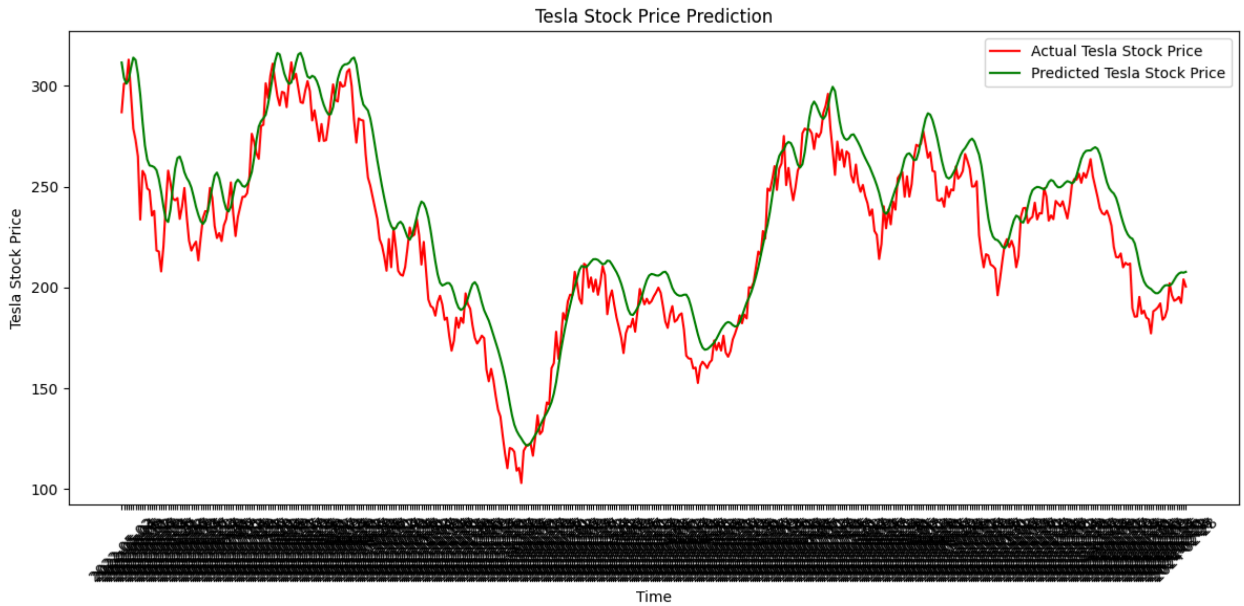
Program:

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt


from sklearn.preprocessing import MinMaxScaler

from keras.models import Sequential

from keras.layers import LSTM, Dropout, Dense


# Load the dataset

tesla = pd.read_csv("/content/drive/MyDrive/Colab Notebooks/Deep Learning/LSTM(RNN)/Tesla DataSet 5 years.csv")

print(tesla.head())


# Data splitting

training = tesla.iloc[:800, 1:2].values  # "Open" prices

testing = tesla.iloc[800:, 1:2].values   # Remaining for testing


# Data normalization

nm_scale = MinMaxScaler(feature_range=(0,1))

training_scaled = nm_scale.fit_transform(training)


# Create data structure with 60 time steps
```

```
x_train = []
y_train = []
for i in range(60, 800):
    x_train.append(training_scaled[i-60:i, 0])
    y_train.append(training_scaled[i, 0])

x_train, y_train = np.array(x_train), np.array(y_train)
x_train = np.reshape(x_train, (x_train.shape[0], x_train.shape[1], 1))  # 3D input for LSTM

# Model architecture
model = Sequential()
model.add(LSTM(units=50, return_sequences=True, input_shape=(x_train.shape[1], 1)))
model.add(Dropout(0.2))
model.add(LSTM(units=50, return_sequences=True))
model.add(Dropout(0.2))
model.add(LSTM(units=50, return_sequences=True))
model.add(Dropout(0.2))
model.add(LSTM(units=50))
model.add(Dropout(0.2))
model.add(Dense(units=1))

# Compile and train the model
model.compile(optimizer='adam', loss='mean_squared_error')
model.fit(x_train, y_train, epochs=100, batch_size=32)

# Preparing test inputs
total_data = pd.concat((tesla.iloc[:800, 1:2], tesla.iloc[800:, 1:2]), axis=0)
inputs = total_data[len(total_data) - len(testing) - 60:].values
inputs = inputs.reshape(-1, 1)
```

```
inputs = nm_scale.transform(inputs)

x_test = []
for i in range(60, 60 + len(testing)):
    x_test.append(inputs[i-60:i, 0])

x_test = np.array(x_test)
x_test = np.reshape(x_test, (x_test.shape[0], x_test.shape[1], 1))

# Predicting the stock prices
predicted_stock_price = model.predict(x_test)
predicted_stock_price = nm_scale.inverse_transform(predicted_stock_price)

# Actual stock prices
actual_stock_price = testing

# Plotting results
plt.figure(figsize=(12,6))
plt.plot(tesla.loc[800:, "Date"], actual_stock_price, color="red", label="Actual Tesla Stock Price")
plt.plot(tesla.loc[800:, "Date"], predicted_stock_price, color="green", label="Predicted Tesla Stock Price")
plt.title("Tesla Stock Price Prediction")
plt.xlabel("Time")
plt.ylabel("Tesla Stock Price")
plt.xticks(rotation=45)
plt.legend()
plt.tight_layout()
plt.show()
```

➔ **Output :**



Result:

The LSTM-based Recurrent Neural Network was trained using Tesla stock data and successfully predicted future stock prices with reasonable accuracy.

Experiment: 12 Image Denoising using Autoencoder

Aim:

To develop and train an Autoencoder model for denoising images using the Fashion MNIST dataset.

Software Requirements:

- ➢ Google Colab

Program:

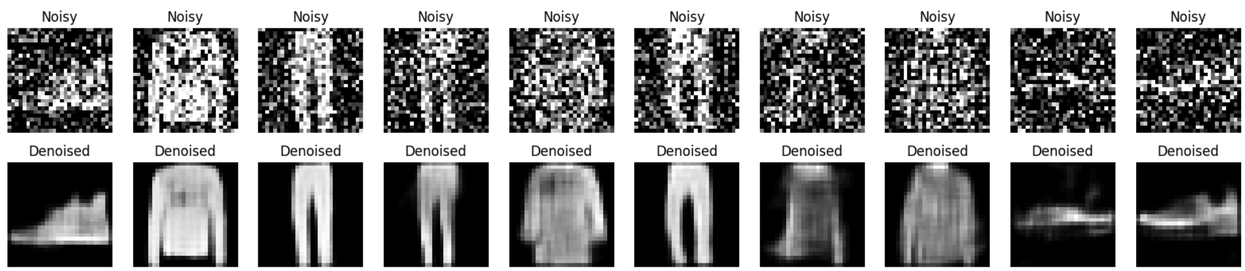
```
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.datasets import fashion_mnist
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Dense, Conv2D, MaxPooling2D, UpSampling2D
from tensorflow.keras.optimizers import Adam

# Load Fashion MNIST dataset
(x_train, _), (x_test, _) = fashion_mnist.load_data()

# Normalize & Reshape images
x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0
x_train = np.expand_dims(x_train, -1)
x_test = np.expand_dims(x_test, -1)

# Add Noise
noise_factor = 0.5
x_train_noisy = x_train + noise_factor * np.random.normal(loc=0.0, scale=1.0,
size=x_train.shape)
x_test_noisy = x_test + noise_factor * np.random.normal(loc=0.0, scale=1.0,
size=x_test.shape)
x_train_noisy = np.clip(x_train_noisy, 0., 1.)
x_test_noisy = np.clip(x_test_noisy, 0., 1.)

# Build Autoencoder
input_img = Input(shape=(28, 28, 1))
x = Conv2D(32, (3, 3), activation='relu', padding='same')(input_img)
x = MaxPooling2D((2, 2), padding='same')(x)
x = Conv2D(32, (3, 3), activation='relu', padding='same')(x)
encoded = MaxPooling2D((2, 2), padding='same')(x)
x = Conv2D(32, (3, 3), activation='relu', padding='same')(encoded)
x = UpSampling2D((2, 2))(x)
```

```
x = Conv2D(32, (3, 3), activation='relu', padding='same')(x)
x = UpSampling2D((2, 2))(x)
decoded = Conv2D(1, (3, 3), activation='sigmoid', padding='same')(x)

# Compile and train
autoencoder = Model(input_img, decoded)
autoencoder.compile(optimizer=Adam(), loss='binary_crossentropy')
autoencoder.fit(x_train_noisy, x_train, epochs=10, batch_size=128, shuffle=True,
validation_data=(x_test_noisy, x_test))

# Evaluate
decoded_imgs = autoencoder.predict(x_test_noisy)

# Visualization
n = 10
plt.figure(figsize=(20, 4))
for i in range(n):
    ax = plt.subplot(2, n, i + 1)
    plt.imshow(x_test_noisy[i].reshape(28, 28), cmap='gray')
    plt.title("Noisy")
    plt.axis('off')

    ax = plt.subplot(2, n, i + 1 + n)
    plt.imshow(decoded_imgs[i].reshape(28, 28), cmap='gray')
    plt.title("Denoised")
    plt.axis('off')
plt.show()

# Accuracy estimation
loss = autoencoder.evaluate(x_test_noisy, x_test, verbose=0)
print(f"Test Loss: {loss}")

decoded_imgs = autoencoder.predict(x_test_noisy)
diff = np.abs(decoded_imgs - x_test)
threshold = 0.1
correct_predictions = np.sum(diff < threshold)
accuracy = correct_predictions / (x_test.shape[0] * x_test.shape[1] * x_test.shape[2])
print(f"Accuracy based on threshold: {accuracy}")
```

➔ **Output** :

```
Test Loss: 0.29475435614585876
```

```
Accuracy based on threshold: 0.7584261479591837
```



Result:
The Autoencoder model was successfully trained on the Fashion MNIST dataset and effectively denoised the noisy images.

Experiment: 13 Neural Style Transfer GAN

Aim:

To apply neural style transfer using a pre-trained GAN model from TensorFlow Hub to blend the style of one image with the content of another.

Software Requirements:

- ➢ Google Colab

Program:

```
from matplotlib import gridspec

import matplotlib.pylab as plt

import numpy as np

import tensorflow as tf

import tensorflow_hub as tensorflow_hub

import PIL

from google.colab import files
```

 # Image Processing Function

```
def load_image(image, image_size=(512,512)):

   img = tf.io.read_file(image)

   img = tf.image.decode_image(img, channels=3)

   img = tf.image.convert_image_dtype(img, tf.float32)

   img = tf.image.resize(img, image_size, preserve_aspect_ratio=True)

   img = img[tf.newaxis, :]

   return img
```

 # Upload content image

```
upload = files.upload()

orginal_image = load_image("Lion.jpg")
```

 # Upload style image

```
upload = files.upload()
style_image = load_image("Tiger.jpeg")
```

 # Display function

```
def imshow(image, title=None):
    if len(image.shape) > 3:
        image = tf.squeeze(image, axis=0)
    plt.imshow(image)
    if title:
        plt.title(title)
```

 # Display content and style images

```
imshow(orginal_image, "Original Image")
```

imshow(style_image, "Style Image")

 # Load model from TensorFlow Hub

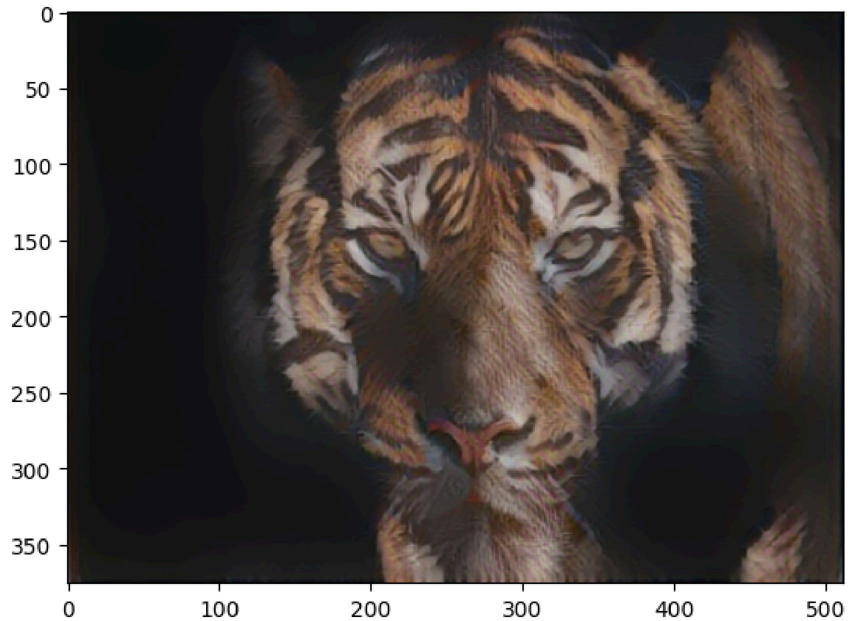
hub_handle = "https://tfhub.dev/google/magenta/arbitrary-image-stylization-v1-256/2"

hub_module = tensorflow_hub.load(hub_handle)

 # Apply style transfer

outputs = hub_module(orginal_image, style_image)

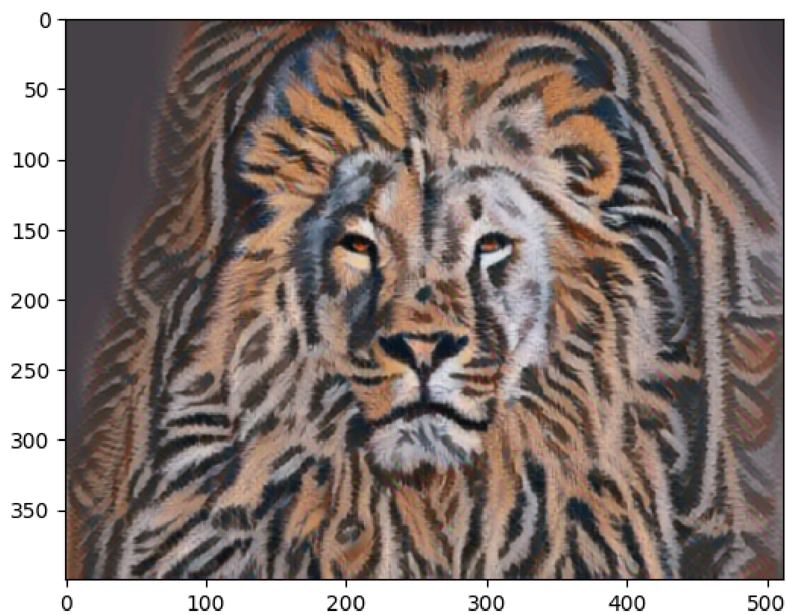
imshow(outputs[0], "Stylized Image")



 # Optionally reverse style and content

outputs = hub_module(style_image, orginal_image)

imshow(outputs[0], "Reversed Style Image")



Result:

The neural style transfer model was successfully implemented. The original image was stylized using the features of the style image, and the resulting images were visualized effectively.