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)
```

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)}')
```

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

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]]))
```

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__)

# 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+")


# 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)


# String tensor
T1 = tf.constant([["a"], ["b"]], dtype=tf.string)
print(T1)
```

```
# Transpose
trans = tf.transpose(con1)
print("con1 Transpose:", trans)

# Type casting
con3 = tf.cast(con1, tf.float32)
con4 = tf.cast(con2, tf.float32)
# Element-wise multiplication
mul_elements = tf.multiply(con3, con4)
print("Mul Elements:", mul_elements)

# Matrix multiplication
mul_mat = tf.matmul(con3, con4)
print("Mul Matrix:", mul_mat)

# Reshaping
reshape_con1 = tf.reshape(tensor=con1, shape=[1, 4])
print("Reshape Con1:", reshape_con1)
# 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)


# Reshape the matrix with Total Element
reshape_con7 = tf.reshape(tensor=con7, shape=[6, 6])
print("Reshape con7:", reshape_con7)
```

```
# Identity matrix
id_int = tf.eye(num_rows=3, num_columns=3, dtype=tf.int32)
print("Identity matrix (int):", id_int)

id_float = tf.eye(num_rows=3, num_columns=3, dtype=tf.float32)
print("Identity matrix (float):", id_float)

# Constant tensor
con_ten = tf.constant([[4, 1], [3, 4]])
print(con_ten)

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

n_var_ten = new_var_ten.assign([[4, 1], [3, 4]])
print(n_var_ten)
# Concatenation
row_con = tf.concat(values=(con_ten, new_var_ten), axis=0)
print(row_con)

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

# Zeros and Ones
zeros = tf.zeros(shape=(3, 4), dtype=tf.int32)
print(zeros)
ones = tf.ones(shape=(3, 4), dtype=tf.int32)
print(ones)
```

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])
```

# 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
```

#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())
```

# Normalization

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

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

Normalizer = Normalization(axis=None)
Normalizer.adapt(data)
Normalized = Normalizer(data)
print("Normalized Mean:", np.mean(Normalized))
print("Normalized Variance:", np.var(Normalized))
```

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))
```

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))
```

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))
```

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)
```

```
    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}")
```

Result:

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