#1 Write a python program to perform the basic mathematical operation and matrix operations using tensor flow.

```
import tensorflow as tf

# Basic Operations

a = tf.constant(5)

b = tf.constant(3)

# Addition

c = tf.add(a, b)

print("Addition of a and b: ", c.numpy())

# Subtraction

d = tf.subtract(a, b)

print("Subtraction of a and b: ", d.numpy())

# Multiplication

e = tf.multiply(a, b)

print("Multiplication of a and b: ", e.numpy())

# Division

f = tf.divide(a, b)

print("Division of a and b: ", f.numpy())

# Matrix Operations

# Matrix 1

matrix1 = tf.constant([[1, 2], [3, 4]])

print("Matrix 1: \n", matrix1.numpy())


# Matrix 2

matrix2 = tf.constant([[5, 6], [7, 8]])

print("Matrix 2: \n", matrix2.numpy())


# Matrix Addition

matrix_add = tf.add(matrix1, matrix2)

print("Matrix Addition: \n", matrix_add.numpy())
```

```
# Matrix Subtraction
matrix_sub = tf.subtract(matrix1, matrix2)
print("Matrix Subtraction: \n", matrix_sub.numpy())

# Matrix Multiplication
matrix_mul = tf.matmul(matrix1, matrix2)
print("Matrix Multiplication: \n", matrix_mul.numpy())



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

#2 Write a python program to perform the basic logic gates AND , OR using Mcculloch pitts model.

class McCullochPitts:
    def __init__(self, weights, bias):
        self.weights = weights
        self.bias = bias

    def activate(self, inputs):
        weighted_sum = sum([w * x for w, x in zip(self.weights, inputs)])
        return 1 if weighted_sum + self.bias > 0 else 0

class ANDGate(McCullochPitts):
    def __init__(self):
        super().__init__([1, 1], -1)

class ORGate(McCullochPitts):
    def __init__(self):
        super().__init__([1, 1], 0)
```

```
# Example usage:
and_gate = ANDGate()
or_gate = ORGate()

inputs = [[0, 0], [0, 1], [1, 0], [1, 1]]

print("AND gate:")
for i in inputs:
    output = and_gate.activate(i)
    print(f"{i[0]} AND {i[1]} = {output}")

print("\nOR gate:")
for i in inputs:
    output = or_gate.activate(i)
    print(f"{i[0]} OR {i[1]} = {output}")
```

#3 Write a python program to implement NAND and NOR gate using Mcculloch pitts model.

```
class McCullochPitts:
    def __init__(self, weights, bias):
        self.weights = weights
        self.bias = bias

    def activate(self, inputs):
        weighted_sum = sum([w * x for w, x in zip(self.weights, inputs)])
        return 1 if weighted_sum + self.bias > 0 else 0

class NANDGate(McCullochPitts):
    def __init__(self):
        super().__init__([-1, -1], 2)
```

```
class NORGate(McCullochPitts):
    def __init__(self):
        super().__init__([-1, -1], 1)

# Example usage:
nand_gate = NANDGate()
nor_gate = NORGate()

inputs = [[0, 0], [0, 1], [1, 0], [1, 1]]

print("NAND gate:")
for i in inputs:
    output = nand_gate.activate(i)
    print(f"{i[0]} NAND {i[1]} = {output}")

print("\nNOR gate:")
for i in inputs:
    output = nor_gate.activate(i)
    print(f"{i[0]} NOR {i[1]} = {output}")



#4 Write a python program to implement logistic regression using house price data set
import numpy as np
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt

# Load the Boston house price dataset
```

```
df = pd.read_csv('/content/housing.csv', header=0)
X = df.iloc[:, :-1].values
y = df.iloc[:, -1].values

# Convert the target variable to binary
y = y.astype(float)
y = np.where(y >= np.median(y), 1, 0)



# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train the logistic regression model
model = LogisticRegression()
model.fit(X_train, y_train)

# Make predictions on the testing set
y_pred = model.predict(X_test)
import matplotlib.pyplot as plt

plt.scatter(y_test, y_pred)
plt.xlabel("True Values")
plt.ylabel("Predictions")
plt.show()

# Evaluate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy*100)



# Create a new dataframe with predicted probabilities and true target values
```

```
output_df = pd.DataFrame({'Predicted Probability': y_pred, 'True Target Values': y_test})

# Print the output dataframe
print(output_df.head())
```

#5 Write a python program to implement MLP classifier using digit classification data set.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score

# Load the digits dataset
digits = load_digits()

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(digits.data, digits.target, test_size=0.2,
random_state=42)

# Create the MLP classifier with 2 hidden layers of 16 neurons each
mlp = MLPClassifier(hidden_layer_sizes=(16, 16), max_iter=1000)

# Train the classifier on the training set
mlp.fit(X_train, y_train)

# Make predictions on the testing set
y_pred = mlp.predict(X_test)

# Calculate the accuracy of the classifier
```

```
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy %:", accuracy*100)

# Plot some example images and their predicted labels
fig, ax = plt.subplots(4, 4, figsize=(8, 8))
for i, axi in enumerate(ax.flat):
    axi.imshow(X_test[i].reshape(8, 8), cmap='gray')
    axi.set(xticks=[], yticks=[])
    axi.set_ylabel(f"Predicted: {y_pred[i]}")
plt.show()
```

#6 Write a python program to implement linear binary classifier using multilayer perceptron.

```
import numpy as np
import pandas as pd
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Load the dataset
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/breast-cancer-wisconsin/wdbc.data"
df = pd.read_csv(url, header=None)

# Add column names
columns = ['id', 'diagnosis', 'mean_radius', 'mean_texture', 'mean_perimeter', 'mean_area',
        'mean_smoothness', 'mean_compactness', 'mean_concavity', 'mean_concave_points',
        'mean_symmetry', 'mean_fractal_dimension', 'radius_se', 'texture_se', 'perimeter_se',
        'area_se', 'smoothness_se', 'compactness_se', 'concavity_se', 'concave_points_se',
        'symmetry_se', 'fractal_dimension_se', 'worst_radius', 'worst_texture', 'worst_perimeter',
        'worst_area', 'worst_smoothness', 'worst_compactness', 'worst_concavity',
'worst_concave_points',
```

```
        'worst_symmetry', 'worst_fractal_dimension']
df.columns = columns

# Drop the id column as it is not useful for classification
df.drop('id', axis=1, inplace=True)

# Encode the diagnosis column using one-hot encoding
diagnosis_encoded = pd.get_dummies(df['diagnosis'])
df.drop('diagnosis', axis=1, inplace=True)
encoded_df = pd.concat([df, diagnosis_encoded], axis=1)

X = encoded_df.iloc[:, :-2].values
y = encoded_df.iloc[:, -2:].values

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train the MLP classifier
model = MLPClassifier(hidden_layer_sizes=(1,), activation='identity', solver='lbfgs')
model.fit(X_train, y_train)

# Make predictions on the testing set
y_pred = model.predict(X_test)

# Concatenate the predicted values with the test set
predicted_df = pd.DataFrame(y_pred, columns=['predicted_B', 'predicted_M'])
test_df = pd.DataFrame(y_test, columns=['actual_B', 'actual_M'])
result_df = pd.concat([test_df, predicted_df], axis=1)

# Print the resulting dataframe
print(result_df.head())
```

```
# Evaluate the accuracy of the model

accuracy = accuracy_score(y_test, y_pred)

print("Accuracy %:", accuracy*100)
```

7 Write a python program in which the database contains 76 attributes but all published experiments refer to using a subset of 14 of them the target field in it is integer value 0 is equal to no, and 1 is equal to more chance of heart attack for performing classification (data set is available on kaggle).

```
import pandas as pd

import numpy as np

from sklearn.model_selection import train_test_split

from sklearn.neural_network import MLPClassifier

from sklearn.metrics import accuracy_score


# Load the dataset

df = pd.read_csv("/content/heart.csv")


# Select a subset of 14 features

feature_cols = ['age', 'sex', 'cp', 'trestbps', 'chol', 'fbs', 'restecg',

            'thalach', 'exang', 'oldpeak', 'slope', 'ca', 'thal']

X = df[feature_cols].values


# Perform binary classification on the target field

y = df['target'].apply(lambda x: 1 if x > 0 else 0).values


# Split the data into training and testing sets

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)


# Train the MLP classifier

model = MLPClassifier(hidden_layer_sizes=(5,), activation='relu', solver='adam')

model.fit(X_train, y_train)
```

```
# Make predictions on the testing set
y_pred = model.predict(X_test)

# Evaluate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy %:", accuracy*100)

# Print the output dataset
output_df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
test_df = pd.DataFrame(X_test, columns=feature_cols)
output_df = pd.concat([test_df, output_df], axis=1)
print(output_df.head())

#8 Implement ann using keras lib for binary classification
import numpy as np
from keras.models import Sequential
from keras.layers import Dense
from keras import regularizers

# Define the input data and labels
X_train = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
y_train = np.array([0, 1, 1, 0])

# Define the model architecture
model = Sequential()
model.add(Dense(8, input_dim=2, activation='tanh', kernel_regularizer=regularizers.l2(0.01)))
model.add(Dense(4, activation='tanh', kernel_regularizer=regularizers.l2(0.01)))
model.add(Dense(1, activation='sigmoid'))

# Compile the model
```

```
model.compile(loss='binary_crossentropy', optimizer='rmsprop', metrics=['accuracy'])


# Train the model
model.fit(X_train, y_train, epochs=5000, verbose=0)


# Evaluate the model
X_test = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
y_test = np.array([0, 1, 1, 0])
loss, accuracy = model.evaluate(X_test, y_test, verbose=0)
print('Accuracy: %.2f' % (accuracy*100))



# 9
# In the given dataset there are various factors which are involved when the patient is hospitalised on the basis of these factors predicting whether the patient will survive or not. The dataset has 85 columns so perform reduction using PCA and normalise the data and perform classification using ANN (patient data set is available on kaggle)


# dataset link  https://www.kaggle.com/andrewmvd/heart-failure-clinical-data


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


# Load the data
data = pd.read_csv('heart_failure_clinical_records_dataset.csv')


# Split the data into features and labels
X = data.drop('DEATH_EVENT', axis=1)
```

```
y = data['DEATH_EVENT']

# Normalize the data
scaler = StandardScaler()
X_norm = scaler.fit_transform(X)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_norm, y, test_size=0.2, random_state=42)

# Define the model architecture
model = Sequential()
model.add(Dense(16, input_dim=12, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

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

# Train the model
model.fit(X_train, y_train, epochs=100, batch_size=32, verbose=0)

# Evaluate the model
loss, accuracy = model.evaluate(X_test, y_test, verbose=0)
print('Accuracy:2f' % (accuracy*100))

# 10
# Write a program to Build a prediction model that will perform the following:
# a) classified if a customer is going to churn or not.
# b) preferably based on the model performance to improve the accuracy.

# # create a csv ile with this data and import it in the model
```

```
#customer_id,age,gender,income,credit_score,num_of_products,has_churned
# 1,25,M,50000,600,2,0
# 2,35,F,75000,700,1,0
# 3,45,M,100000,800,3,0
# 4,30,F,60000,650,2,1
# 5,50,M,120000,750,1,0
# 6,40,F,90000,700,2,1
# ,55,M,150000,800,3,0
# 8,28,F,55000,600,1,0
# 9,32,M,65000,700,2,0
# 10,48,F,110000,750,3,0

# import required libraries
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# load dataset
df = pd.read_csv("customer_churn_dataset.csv")

# create dummy variables for categorical features
# df = pd.get_dummies(df, columns=["gender", "region", "partner", "dependents", "phone_service", "multiple_lines", "internet_service", "online_security", "online_backup", "device_protection", "tech_support", "streaming_tv", "streaming_movies", "contract", "paperless_billing", "payment_method"])

# split dataset into training and testing sets
X = df.drop("churn", axis=1)
y = df["churn"]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

```
# create logistic regression model
lr = LogisticRegression(random_state=42)

# define hyperparameters for grid search
params = {"penalty": ["l1", "l2"], "C": [0.001, 0.01, 0.1, 1, 10, 100]}

# perform grid search to find best hyperparameters
gs = GridSearchCV(lr, params, cv=5)
gs.fit(X_train, y_train)

# evaluate model on test set
y_pred = gs.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))
```

#11

Write a python program to build a weather predicton model for the dates given also Visualize the actual predicted data using matplotlib.

https://www.kaggle.com/muthuj7/weather-dataset

```
# import required libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# load dataset
df = pd.read_csv("/content/sample_data/weatherHistory.csv")
```

```
# extract features and target variable
X = df[["Humidity", "Wind Speed (km/h)"]].values
y = df["Temperature (C)"].values

# split dataset into training and testing sets
split = int(0.8*len(df))
X_train, X_test = X[:split], X[split:]
y_train, y_test = y[:split], y[split:]

# create linear regression model
lr = LinearRegression()

# train model on training set
lr.fit(X_train, y_train)

# predict temperature for test set
y_pred = lr.predict(X_test)

# calculate metrics for evaluation
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print("Mean Squared Error:", mse)
print("R2 Score:", r2)

# plot actual and predicted temperature values
plt.plot(y_test, label="Actual Temperature")
plt.plot(y_pred, label="Predicted Temperature")
plt.xlabel("Time (Days)")
plt.ylabel("Temperature (C)")
plt.legend()
```

```
plt.show()



#12
# Normalise the data in the given dataset in question 11, and perform classification using ANN.
# https://www.kaggle.com/muthuj7/weather-dataset
# import required libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from keras.models import Sequential
from keras.layers import Dense

# load dataset
df = pd.read_csv("/content/sample_data/weatherHistory.csv")

# extract features and target variable
X = df[["Humidity", "Wind Speed (km/h)"]].values
y = df["Temperature (C)"].values

# normalize features
scaler = StandardScaler()
X = scaler.fit_transform(X)

# split dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# create ANN model
model = Sequential()
```

```
model.add(Dense(10, input_dim=2, activation='relu'))
model.add(Dense(1, activation='linear'))

# compile model
model.compile(loss='mean_squared_error', optimizer='adam', metrics=['mse'])

# train model on training set
history = model.fit(X_train, y_train, epochs=10, batch_size=32, validation_split=0.2)

# evaluate model on test set
loss, mse = model.evaluate(X_test, y_test)
print("Mean Squared Error:", mse)

# plot training and validation loss over epochs
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend(['Train', 'Validation'])
plt.show()
```

#13

```
# Using Neural network train the model with train dataset , predict the covid  death cases  and confirmed cases. visualize the actual predict data using matplotlib. use20days time stamp.

# url for data set download https://www.kaggle.com/datasets/imdevskp/corona-virus-report?select=full_grouped.csv

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 Dense, LSTM

# Load data from CSV file
df = pd.read_csv('/content/full_grouped.csv')

# Extract the relevant features
data = df[['Confirmed', 'Deaths']].values.astype(float)

# Normalize the data
scaler = MinMaxScaler(feature_range=(0, 1))
data = scaler.fit_transform(data)

# Split the data into training and testing sets
train_size = int(len(data) * 0.8)
train_data = data[:train_size, :]
test_data = data[train_size:, :]

# Define the time stamp
time_stamp = 20

# Define the input and target variables for training and testing
x_train, y_train = [], []
x_test, y_test = [], []

for i in range(time_stamp, len(train_data)):
    x_train.append(train_data[i-time_stamp:i, :])
    y_train.append(train_data[i, :])

for i in range(time_stamp, len(test_data)):
    x_test.append(test_data[i-time_stamp:i, :])
```

```
    y_test.append(test_data[i, :])

# Convert the input and target variables to numpy arrays
x_train, y_train = np.array(x_train), np.array(y_train)
x_test, y_test = np.array(x_test), np.array(y_test)

# Define the neural network model
model = Sequential()
model.add(LSTM(50, return_sequences=True, input_shape=(time_stamp, 2)))
model.add(LSTM(50))
model.add(Dense(2))
model.compile(loss='mean_squared_error', optimizer='adam')

# Train the model
model.fit(x_train, y_train, epochs=5, batch_size=32)

# Predict the number of confirmed cases and deaths for the next 20 days
x_pred = data[-time_stamp:, :]
x_pred = np.reshape(x_pred, (1, time_stamp, 2))
y_pred = model.predict(x_pred)
y_pred = scaler.inverse_transform(y_pred)
y_pred = y_pred.reshape((2,)) # reshape y_pred from (1, 2) to (2,)
# Visualize the actual and predicted data using matplotlib
fig, ax = plt.subplots(figsize=(10, 5))

ax.plot(df['Confirmed'], label='Actual Confirmed')
ax.plot(df.index[-2:], y_pred[0:], label='Predicted Confirmed') # use y_pred[0] for confirmed cases
ax.plot(df['Deaths'], label='Actual Deaths')
ax.plot(df.index[-1:], y_pred[1:], label='Predicted Deaths') # use y_pred[1] for death cases

ax.set_xlabel('Date')
```

```
ax.set_ylabel('Number of Cases')

ax.set_title('COVID-19 Prediction')

ax.legend()


plt.show()



#14

# a.	Build CNN for identifying gestures of human being.

# b.	Improve the model tuning hyper parameters

# Url for data set download https://www.kaggle.com/datasets/grassknoted/asl-alphabet?select=asl_alphabet_test


import numpy as np

from keras.preprocessing.image import ImageDataGenerator


# define paths to the dataset

train_path = 'asl_alphabet_train'

test_path = 'asl_alphabet_test'


# define the data generators for train and test sets

train_datagen = ImageDataGenerator(rescale=1./255,

                                   rotation_range=20,

                                   width_shift_range=0.2,

                                   height_shift_range=0.2,

                                   shear_range=0.2,

                                   zoom_range=0.2,

                                   horizontal_flip=True)


test_datagen = ImageDataGenerator(rescale=1./255)
```

```
train_generator = train_datagen.flow_from_directory(train_path,
                                                    target_size=(64, 64),
                                                    batch_size=32,
                                                    class_mode='categorical')

test_generator = test_datagen.flow_from_directory(test_path,
                                                  target_size=(64, 64),
                                                  batch_size=32,
                                                  class_mode='categorical')




from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout

# define the CNN model
model = Sequential()

model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(64, 64, 3)))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(29, activation='softmax'))

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

```
# train the model
model.fit(train_generator, epochs=10, validation_data=test_generator)

# evaluate the model
test_loss, test_acc = model.evaluate(test_generator)
print(f'Test loss: {test_loss:.4f}')
print(f'Test accuracy: {test_acc:.4f}')



from sklearn.model_selection import RandomizedSearchCV
from keras.wrappers.scikit_learn import KerasClassifier

# define the function that creates the model
def create_model():
    model = Sequential()
    model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(64, 64, 3)))
    model.add(MaxPooling2D((2, 2)))
    model.add(Conv2D(64, (3, 3), activation='relu'))
    model.add(MaxPooling2D((2, 2)))
    model.add(Conv2D(128, (3, 3), activation='relu'))
    model.add(MaxPooling2D((2, 2)))
    model.add(Flatten())
    model.add(Dense(128, activation='relu'))
    model



#15
```

```
# Build a deep learning model which classifies cats and dogs using CNN.

# URL of the dataset for download
# url = 'https://www.microsoft.com/en-us/download/details.aspx?id=54765'

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.preprocessing.image import ImageDataGenerator



train_datagen = ImageDataGenerator(
   rescale=1./255,shear_range=0.2,
   zoom_range=0.2,horizontal_flip=True,validation_split=0.2)
train_generator = train_datagen.flow_from_directory(
   '/content/cats_and_dogs_filtered/train',target_size=(224, 224),
   batch_size=32,class_mode='binary',subset='training')

validation_generator = train_datagen.flow_from_directory(
   '/content/cats_and_dogs_filtered/train',target_size=(224, 224),
   batch_size=32,class_mode='binary',subset='validation')
model = keras.Sequential([
   layers.Conv2D(32, (3,3), activation='relu', input_shape=(224,224,3)),
   layers.MaxPooling2D((2,2)),layers.Conv2D(64, (3,3), activation='relu'),
   layers.MaxPooling2D((2,2)),layers.Conv2D(128, (3,3), activation='relu'),
   layers.MaxPooling2D((2,2)),layers.Conv2D(128, (3,3), activation='relu'),
   layers.MaxPooling2D((2,2)),layers.Flatten(),layers.Dense(512, activation='relu'),
   layers.Dense(1, activation='sigmoid')])

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

```
history = model.fit(train_generator,epochs=10,validation_data=validation_generator,verbose=1)

test_generator = train_datagen.flow_from_directory('./cats-vs-dogs/test',
                                                   target_size=(224, 224),batch_size=32,class_mode='binary')


test_loss, test_acc = model.evaluate(test_generator, verbose=1)

print('Test accuracy:', test_acc)
```

#16

The stock price of the present day can be predicted by the stock prices obtained for past 50 days. Using Simple RNN train the model with “Google_stock_price_train.csv” dataset, predict the stock prices for the dates given in “Google_stock_price_train.csv” dataset and visualize the actual predicted prices using matplotlib

url for dataset download https://www.kaggle.com/datasets/vaibhavsxn/google-stock-prices-training-and-test-data?select=Google_Stock_Price_Train.csv

```
import pandas as pd

import numpy as np

from sklearn.preprocessing import MinMaxScaler

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, SimpleRNN, Dropout


# load the data

df = pd.read_csv('/content/sample_data/Google_Stock_Price_Train.csv')


# preprocess the data

df['Close'] = df['Close'].str.replace(',', '').astype(float)

training_data = df['Close'].values.reshape(-1, 1)

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

training_data = scaler.fit_transform(training_data)


X_train = []

y_train = []
```

```
# prepare the data for RNN
for i in range(50, len(training_data)):
    X_train.append(training_data[i-50:i, 0])
    y_train.append(training_data[i, 0])

X_train, y_train = np.array(X_train), np.array(y_train)

# build the model
model = Sequential()
model.add(SimpleRNN(units=50, return_sequences=True, input_shape=(X_train.shape[1], 1)))
model.add(Dropout(0.2))
model.add(SimpleRNN(units=50, return_sequences=True))
model.add(Dropout(0.2))
model.add(SimpleRNN(units=50))
model.add(Dropout(0.2))
model.add(Dense(units=1))

model.compile(optimizer='adam', loss='mean_squared_error')

# train the model
model.fit(X_train, y_train, epochs=50, batch_size=32)

# predict on the training data
y_pred = model.predict(X_train)

# inverse transform the predicted and actual data
y_pred = scaler.inverse_transform(y_pred)
y_train = scaler.inverse_transform(y_train.reshape(-1, 1))

# visualize the results
```

```
import matplotlib.pyplot as plt


plt.plot(y_train, label='Actual')
plt.plot(y_pred, label='Predicted')
plt.legend()
plt.show()



#17
# Write a python program to Use a stochastic gradient descent optimizer for the similar classification problem and compare the result of both the model in the first case you can use optimizer Adam.
import numpy as np
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.optimizers import SGD, Adam
from tensorflow.keras.utils import to_categorical

# Load the MNIST dataset
(X_train, y_train), (X_test, y_test) = mnist.load_data()

# Normalize the pixel values
X_train = X_train / 255.0
X_test = X_test / 255.0

# One-hot encode the target variable
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)

# Define the model architecture
model = Sequential()
```

```
model.add(Flatten(input_shape=(28, 28)))
model.add(Dense(units=128, activation='relu'))
model.add(Dense(units=10, activation='softmax'))

# Compile the model using Adam optimizer
adam_optimizer = Adam()
model.compile(optimizer=adam_optimizer, loss='categorical_crossentropy', metrics=['accuracy'])
adam_history = model.fit(X_train, y_train, epochs=5, validation_data=(X_test, y_test))

# Compile the model using SGD optimizer
sgd_optimizer = SGD(learning_rate=0.01, momentum=0.9)
model.compile(optimizer=sgd_optimizer, loss='categorical_crossentropy', metrics=['accuracy'])
sgd_history = model.fit(X_train, y_train, epochs=5, validation_data=(X_test, y_test))

# Compare the results
print("Adam optimizer accuracy: ", np.mean(adam_history.history['val_accuracy']))
print("SGD optimizer accuracy: ", np.mean(sgd_history.history['val_accuracy']))
```

Write a python program to perform Multiclass classification on MNIST data set by Artificial neural network use softmax activation function for the output layer Evaluate the performance by using confusion matrix.

```
import numpy as np
import tensorflow as tf
from tensorflow import keras
from sklearn.metrics import confusion_matrix
from matplotlib import pyplot as plt

# Load the MNIST dataset
(x_train, y_train), (x_test, y_test) = keras.datasets.mnist.load_data()

# Normalize the data
```

```
x_train = x_train.astype("float32") / 255.0
x_test = x_test.astype("float32") / 255.0

# Convert the labels to one-hot encoded vectors
num_classes = 10
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)

# Define the ANN model
model = keras.Sequential([
   keras.layers.Flatten(input_shape=(28, 28)),
   keras.layers.Dense(128, activation='relu'),
   keras.layers.Dense(num_classes, 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, batch_size=32,
                validation_data=(x_test, y_test))

# Evaluate the model
test_loss, test_acc = model.evaluate(x_test, y_test)
print('Test accuracy:', test_acc)

# Make predictions
y_pred = model.predict(x_test)
y_pred = np.argmax(y_pred, axis=1)
```

```
# Create a confusion matrix
cm = confusion_matrix(np.argmax(y_test, axis=1), y_pred)

# Plot the confusion matrix
fig, ax = plt.subplots(figsize=(10,10))
ax.imshow(cm, cmap='Blues')
ax.grid(False)
ax.set_xlabel('Predicted labels', fontsize=12, color='black')
ax.set_ylabel('True labels', fontsize=12, color='black')
ax.set_xticks(range(10))
ax.set_yticks(range(10))
ax.xaxis.set_ticklabels(range(10), fontsize=10)
ax.yaxis.set_ticklabels(range(10), fontsize=10)
plt.show()
```

#19

```
# Perform Multi class classification on MNIST dataset using ANN. (Note: Use softmax activation function for the output layer with 3-4 hidden layers consisting of ReLU activation function also evaluate the performance by using confusion matrix.)
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Flatten
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.utils import to_categorical
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
```

```
# Load MNIST dataset
(X_train, y_train), (X_test, y_test) = mnist.load_data()

# Normalize pixel values to be between 0 and 1
X_train = X_train.astype('float32') / 255.0
X_test = X_test.astype('float32') / 255.0

# Reshape input data to a flat vector of 784 pixels
X_train = X_train.reshape(-1, 784)
X_test = X_test.reshape(-1, 784)

# Convert target variable to categorical format
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)

# Define model architecture
model = Sequential()
model.add(Dense(512, activation='relu', input_shape=(784,)))
model.add(Dropout(0.2))
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(10, activation='softmax'))

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

# Train the model
```

```
history = model.fit(X_train, y_train,
                    batch_size=128,
                    epochs=20,
                    verbose=1,
                    validation_split=0.2)

# Evaluate the model
test_loss, test_acc = model.evaluate(X_test, y_test, verbose=0)
print(f'Test Loss: {test_loss:.4f}')
print(f'Test Accuracy: {test_acc:.4f}')

# Make predictions
y_pred = model.predict(X_test)
y_pred = np.argmax(y_pred, axis=1)
y_test = np.argmax(y_test, axis=1)

# Create confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Visualize confusion matrix
plt.imshow(cm, cmap='binary')
plt.colorbar()
plt.xticks(np.arange(10))
plt.yticks(np.arange(10))
plt.xlabel('Predicted label')
plt.ylabel('True label')
plt.show()
```