```
# Peactical NO 4 Convolutional neural network (CNN) (Any One from the following)
# #### Use any dataset of plant disease and design a plant disease detection system using CNN.
# #### Use MNIST Fashion Dataset and create a classifier to classify fashion clothing into categories.

import pandas as pd
import numpy as np
import tensorflow as tf
from tensorflow.keras import layers, models

train_df = pd.read_csv('fashion-mnist_train.csv')
test_df = pd.read_csv('fashion-mnist_test.csv')

train_df.head(2)
test_df.head(2)


# Split features and labels
x_train = train_df.iloc[:, 1:].values
y_train = train_df.iloc[:, 0].values

x_test = test_df.iloc[:, 1:].values
y_test = test_df.iloc[:, 0].values

# Normalize pixel values
x_train = x_train / 255.0
x_test = x_test / 255.0
# Reshape for CNN: (samples, height, width, channels)
x_train = x_train.reshape(-1, 28, 28, 1)
x_test = x_test.reshape(-1, 28, 28, 1)

# Build CNN model
model = models.Sequential([
layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
```

```
layers.MaxPooling2D(2, 2),

layers.Conv2D(64, (3, 3), activation='relu'),

layers.MaxPooling2D(2, 2),

layers.Flatten(),

layers.Dense(64, activation='relu'),

layers.Dense(10, activation='softmax')

])

 # Compile model

model.compile(optimizer='adam',

loss='sparse_categorical_crossentropy',

metrics=['accuracy'])

 # Train

model.fit(x_train, y_train, epochs=5, validation_split=0.1)


# Evaluate

loss,acc = model.evaluate(x_test,y_test)

print(f"\nTest Accuracy:,{acc}")

import matplotlib.pyplot as plt

class_names=['T-
shirt/top','Trouser','Pullover','Dress','Coat','Sandal','Shirt','Snakers','Bag','Ankleboot']

plt.figure(figsize=(10,10))

for i in  range(25):

  plt.subplot(5,5,i+1)

  plt.xticks([])

  plt.yticks([])

  plt.grid(False)

  plt.imshow(x_test[i],cmap=plt.cm.binary)

  plt.xlabel(class_names[y_test[i]])

plt.show()
```

Linear regression by using Deep Neural network: Implement Boston housing price prediction problem by Linear regression using Deep Neural Network. Use Boston House Price prediction Dataset.

```
import tensorflow as tf

from tensorflow import keras

from tensorflow.keras import layers

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.preprocessing import StandardScaler

from sklearn.model_selection import train_test_split

from tensorflow.keras.optimizers import Adam

df = pd.read_csv("HousingData.csv")

df.head()
df.info()

df.fillna(df.mean(), inplace=True)

 X = df.drop(columns=['MEDV'])

y = df['MEDV']

scaler = StandardScaler()

X = scaler.fit_transform(X)

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

model = keras.Sequential([

keras.Input(shape=(X.shape[1],)), # Explicit Input Layer

layers.Dense(64, activation='relu'), # Hidden Layer 1

layers.Dense(32, activation='relu'), # Hidden Layer 2

layers.Dense(1, activation='linear') # Output layer (Regression)

])
```

```
model.compile(optimizer=Adam(learning_rate=0.01), loss='mse', metrics=['mae'])

history = model.fit(X_train, y_train, epochs=100, validation_data=(X_test, y_test), batch_size=16,
verbose=1)

 loss, mae = model.evaluate(X_test, y_test)
print(f"Test Loss (MSE): {loss}")
print(f"Test Mean Absolute Error (MAE):{mae}")

prediction = model.predict(X_test)

plt.scatter(y_test, prediction)
plt.xlabel("Actual Prices")
plt.ylabel("Predicted Prices")
plt.title("Actual Prices vs Predicted Prices")
plt.show()



# Practical No 2
# Binary classification using Deep Neural Networks Example: Classify movie
# reviews into positive" reviews and "negative" reviews, just based on the
# text content of the reviews. Use IMDB dataset.

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
import matplotlib.pyplot as plt

# Load and preprocess IMDB dataset
vocab_size = 10000
max_length = 200
```

```
(x_train, y_train), (x_test, y_test) = keras.datasets.imdb.load_data(num_words=vocab_size)


x_train = keras.preprocessing.sequence.pad_sequences(x_train, maxlen=max_length,
padding='post')

x_test = keras.preprocessing.sequence.pad_sequences(x_test, maxlen=max_length, padding='post')


# Build the model

model = keras.Sequential([

    layers.Embedding(input_dim=vocab_size, output_dim=64),

    layers.Conv1D(32, 5, activation='relu'),

    layers.GlobalMaxPooling1D(),

    layers.Dense(64, activation='relu'),

    layers.Dense(1, activation='sigmoid')

])


# Compile the model

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


# Train the model

history = model.fit(x_train, y_train, epochs=5, batch_size=64, validation_data=(x_test, y_test),
verbose=1)


# Evaluate the model

loss, accuracy = model.evaluate(x_test, y_test, verbose=1)

print(f"Test Accuracy: {accuracy:.4f}")

print(f"Test Loss: {loss:.4f}")



# Plot training history

plt.plot(history.history['accuracy'], label='Train Accuracy')
```

```
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')

plt.xlabel('Epochs')

plt.ylabel('Accuracy')

plt.legend()

plt.show()


# Implement Min, Max, Sum and Average oprations using Parallel Reduction.


import multiprocessing

import random


def parallel_reduction(operation, arr):

    with multiprocessing.Pool() as pool:

        if operation == "min":

            return min(pool.map(min, arr))

        elif operation == "max":

            return max(pool.map(max, arr))

        elif operation == "sum":

            return sum(pool.map(sum, arr))

        elif operation == "avg":

            return sum(pool.map(sum, arr)) / sum(len(chunk) for chunk in arr)


if __name__ == "__main__":

    arr = [random.randint(0, 10000) for _ in range(10000)]

    cpu_count = multiprocessing.cpu_count()

    chunked_arr = [arr[i::cpu_count] for i in range(cpu_count)]


    print(f"Min: {parallel_reduction('min', chunked_arr)}")

    print(f"Max: {parallel_reduction('max', chunked_arr)}")

    print(f"Sum: {parallel_reduction('sum', chunked_arr)}")

    print(f"Average: {parallel_reduction('avg', chunked_arr)}")
```

//Design and implement Parallel Breadth First Search and Depth First Search based on existing algorithms using OpenMP. Use a Tree or an undirected graph for BFS and DFS .

```
#include <iostream>

#include <vector>

#include <queue>

#include <stack>

#include <omp.h>

using namespace std;


class Graph {

   int V; vector<vector<int>> adj;

public:

   Graph(int v): V(v) { adj.resize(v);

 }

   void add(int u, int v) { adj[u].push_back(v); adj[v].push_back(u);

}


   void bfs(int s) {

      vector<bool> v(V,0); queue<int> q; q.push(s); v[s]=1;

      cout<<"BFS: ";

      while(!q.empty()) {

         int sz = q.size();

         #pragma omp parallel for

         for(int i=0; i<sz; i++) {

            int n;

            #pragma omp critical

            { n=q.front(); q.pop(); cout<<n<<" "; }

            for(int nb:adj[n]) {

               #pragma omp critical

               if(!v[nb]) { v[nb]=1; q.push(nb); }

            }
```

```
        }
      }
      cout<<endl;
    }

    void dfs(int s) {
      vector<bool> v(V,0); stack<int> st; st.push(s); v[s]=1;
      cout<<"DFS: ";
      while(!st.empty()) {
        int n;
        #pragma omp critical
        { n=st.top(); st.pop(); cout<<n<<" "; }
        #pragma omp parallel for
        for(int i=0; i<adj[n].size(); i++) {
          int nb = adj[n][i];
          #pragma omp critical
          if(!v[nb]) { v[nb]=1; st.push(nb); }
        }
      }
      cout<<endl;
    }
};

int main() {
    Graph g(6);
    g.add(0,1); g.add(0,2); g.add(1,3); g.add(1,4); g.add(2,5);
    g.bfs(0); g.dfs(0);
    return 0;
}
```

//Write a program to implement Parallel Bubble Sort and Merge sort using OpenMP. Use existing algorithms and measure the performance of sequential and parallel algorithms.

```
#include <iostream>

#include <vector>

#include <omp.h>

using namespace std;


void bubbleSeq(vector<int>& a) {

   for (int i = 0; i < a.size()-1; i++)

      for (int j = 0; j < a.size()-i-1; j++)

         if (a[j] > a[j+1]) swap(a[j], a[j+1]);

}


void bubblePar(vector<int>& a) {

   for (int i = 0; i < a.size(); i++) {

      #pragma omp parallel for

      for (int j = i%2; j < a.size()-1; j+=2)

         if (a[j] > a[j+1]) swap(a[j], a[j+1]);

   }

}


void merge(vector<int>& a, int l, int m, int r) {

   vector<int> L(a.begin()+l, a.begin()+m+1), R(a.begin()+m+1, a.begin()+r+1);

   int i=0, j=0, k=l;

   while (i<L.size() && j<R.size()) a[k++] = (L[i]<R[j]) ? L[i++] : R[j++];

   while (i<L.size()) a[k++] = L[i++];

   while (j<R.size()) a[k++] = R[j++];

}


void mergeSeq(vector<int>& a,   int l, int r) {

   if (l < r) {
```

```
        int m = (l+r)/2;
        mergeSeq(a, l, m);
        mergeSeq(a, m+1, r);
        merge(a, l, m, r);
    }
}

void mergePar(vector<int>& a, int l, int r) {
    if (l < r) {
        int m = (l+r)/2;
        #pragma omp parallel sections
        {
            #pragma omp section
            mergePar(a, l, m);
            #pragma omp section
            mergePar(a, m+1, r);
        }
        merge(a, l, m, r);
    }
}

int main() {
    vector<int> d = {8,5,2,9,1,4};
    vector<int> a1=d, a2=d, a3=d, a4=d;

    bubbleSeq(a1);
    bubblePar(a2);
    mergeSeq(a3, 0, a3.size()-1);
    mergePar(a4, 0, a4.size()-1);

    cout<<"Seq Bubble: "; for(int x:a1) cout<<x<<" "; cout<<endl;
```

```
    cout<<"Par Bubble: "; for(int x:a2) cout<<x<<" "; cout<<endl;
    cout<<"Seq Merge: "; for(int x:a3) cout<<x<<" "; cout<<endl;
    cout<<"Par Merge: "; for(int x:a4) cout<<x<<" "; cout<<endl;

    return 0;
}
```