Name: Mohit Bhavsar

RollNo: C43435

Batch: B9

Assignment No: 1

BFS:

```
#include<iostream>
#include<stdlib.h>
#include<queue>
using namespace std;


class node
{
  public:

    node *left, *right;
    int data;

};

class Breadthfs
{

 public:

 node *insert(node *, int);
 void bfs(node *);

};


node *insert(node *root, int data)
// inserts a node in tree
{

    if(!root)
    {

           root=new node;
           root->left=NULL;
           root->right=NULL;
```

```
        root->data=data;
        return root;
}

queue<node *> q;
q.push(root);

while(!q.empty())
{

        node *temp=q.front();
        q.pop();

        if(temp->left==NULL)
        {

                temp->left=new node;
                temp->left->left=NULL;
                temp->left->right=NULL;
                temp->left->data=data;
                return root;
        }
        else
        {

        q.push(temp->left);

        }

        if(temp->right==NULL)
        {

                temp->right=new node;
                temp->right->left=NULL;
                temp->right->right=NULL;
                temp->right->data=data;
                return root;
        }
        else
        {

        q.push(temp->right);

        }
```

```
    }

}


void bfs(node *head)
{

        queue<node*> q;
        q.push(head);

        int qSize;

        while (!q.empty())
        {
                qSize  =  q.size();
                #pragma omp parallel for
               //creates parallel threads
                for (int i = 0; i < qSize; i++)
                {
                        node* currNode;
                        #pragma omp critical
                        {
                          currNode = q.front();
                          q.pop();
                          cout<<"\t"<<currNode->data;

                        }// prints parent node
                        #pragma omp critical
                        {
                        if(currNode->left)// push parent's left node in queue
                                q.push(currNode->left);
                        if(currNode->right)
                                q.push(currNode->right);
                        }// push parent's right node in queue

                }
        }

}

int main(){
```

```
    node *root=NULL;
    int data;
    char ans;

    do
    {
        cout<<"\n enter data=>";
        cin>>data;

        root=insert(root,data);

        cout<<"do you want insert one more node?";
        cin>>ans;

    }while(ans=='y'||ans=='Y');

    bfs(root);

    return 0;
}
```

Run Commands:

1. g++ -fopenmp bfs.cpp -o bfs

2. ./bfs

Output:

This code represents a breadth-first search (BFS) algorithm on a binary tree using OpenMP for parallelization. The program asks for user input to insert nodes into the binary tree and then performs the BFS algorithm using multiple threads. Here's an example output for a binary tree with nodes 5, 3, 2, 1, 7, and 8:

```
Enter data => 5
Do you want to insert one more node? (y/n) y

Enter data => 3
Do you want to insert one more node? (y/n) y

Enter data => 2
Do you want to insert one more node? (y/n) y

Enter data => 1
Do you want to insert one more node? (y/n) y

Enter data => 7
Do you want to insert one more node? (y/n) y

Enter data => 8
Do you want to insert one more node? (y/n) n

        5       3       7       2       1       8
```

The nodes are printed in breadth-first order. The #pragma omp parallel for statement is used to parallelize the for loop that processes each level of the binary tree. The #pragma omp critical statement is used to synchronize access to shared data structures, such as the queue that stores the nodes of the binary tree.

Here is an example of the breadth-first traversal for a binary tree with the values 5, 3, 2, 1, 7, and 8:

Starting with the root node containing value 5:

```
    5
   / \
  3   7
 /     \
2       8
 \
  1
```

The traversal would be:

```
5, 3, 7, 2, 8, 1
```

DFS:

```
#include <iostream>
#include <vector>
#include <stack>
#include <omp.h>

using namespace std;

const int MAX = 100000;
vector<int> graph[MAX];
bool visited[MAX];
```

```
void dfs(int node) {
        stack<int> s;
        s.push(node);

        while (!s.empty()) {
        int curr_node = s.top();
        s.pop();

        if (!visited[curr_node]) {
        visited[curr_node] = true;

        if (visited[curr_node]) {
        cout << curr_node << " ";
        }

        #pragma omp parallel for
        for (int i = 0; i < graph[curr_node].size(); i++) {
                int adj_node = graph[curr_node][i];
                if (!visited[adj_node]) {
                s.push(adj_node);
                }
        }
        }
        }
}

int main() {
        int n, m, start_node;
        cout << "Enter No of Node,Edges,and start node:" ;
        cin >> n >> m >> start_node;
      //n: node,m:edges

cout << "Enter Pair of edges:" ;
        for (int i = 0; i < m; i++) {
        int u, v;

        cin >> u >> v;
//u and v: Pair of edges
        graph[u].push_back(v);
        graph[v].push_back(u);
        }

        #pragma omp parallel for
        for (int i = 0; i < n; i++) {
        visited[i] = false;
        }

        dfs(start_node);

/*      for (int i = 0; i < n; i++) {
        if (visited[i]) {
        cout << i << " ";
        }       }*/return 0;}
```

Name: Mohit Bhavsar

RollNo: C43435

Batch: B9

Assignment No: 2

```
#include <iostream>
#include <chrono>
#include <omp.h>
using namespace std;
using namespace std::chrono;


// Function to perform parallel bubble sort
 void parallelBubbleSort(int arr[], int n) {
   #pragma omp parallel num_threads(4)
    {
        int first = omp_get_thread_num() * (n / omp_get_num_threads());
        int last = (omp_get_thread_num() + 1) * (n / omp_get_num_threads()) - 1;
        for (int i = first; i < last; i++) {
            for (int j = 0; j < n - i - 1; j++) {
                if (arr[j] > arr[j + 1]) {
                    swap(arr[j], arr[j + 1]);
                }
            }
        }
    }
}
```

```
// Function to perform parallel merge sort
void parallelMergeSort(int arr[], int l, int r) {
  if (l < r) {
    int m = l + (r - l) / 2;
    #pragma omp parallel sections num_threads(2)
    {
      #pragma omp section
      {
        parallelMergeSort(arr, l, m);
      }
      #pragma omp section
      {
        parallelMergeSort(arr, m + 1, r);
      }
    }
    int i, j, k;
    int n1 = m - l + 1;
    int n2 = r - m;
    int L[n1], R[n2];
    for (i = 0; i < n1; i++) {
      L[i] = arr[l + i];
    }
    for (j = 0; j < n2; j++) {
      R[j] = arr[m + 1 + j];
```

```
    }
    i = 0;
    j = 0;
    k = l;
    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) {
            arr[k] = L[i];
            i++;
        } else {
            arr[k] = R[j];
            j++;
        }
        k++;
    }
    while (i < n1) {
        arr[k] = L[i];
        i++;
        k++;
    }
    while (j < n2) {
        arr[k] = R[j];
        j++;
        k++;
    }
}
```

```
}

int main() {
    int n = 10000;
    int arr[n];
    for (int i = 0; i < n; i++) {
        arr[i] = rand() % 1000;
    }

    // Sequential Bubble Sort
    auto start = high_resolution_clock::now();
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (arr[j] > arr[j + 1]) {
                swap(arr[j], arr[j + 1]);
            }
        }
    }
    auto stop = high_resolution_clock::now();
    auto duration = duration_cast<microseconds>(stop - start);
    cout << "Sequential Bubble Sort Time: " << duration.count() << " microseconds" << endl;

    // Parallel Bubble Sort
    start = high_resolution_clock::now();
    parallelBubbleSort(arr, n);
```

```
stop = high_resolution_clock::now();

duration = duration_cast<microseconds>(stop - start);

cout << "Parallel Bubble Sort Time: " << duration.count() << " microseconds
```

```
Sequential Bubble Sort Time: 107387 microseconds
Parallel Bubble Sort Time: 31406 microseconds
```

Name: Mohit Bhavsar

RollNo: C43435

Batch: B9

Assignment No: 3

```
#include <iostream>
#include <omp.h>
using namespace std;

// Function to perform parallel reduction for min operation
int parallelMin(int arr[], int n) {
    int result = arr[0];
    #pragma omp parallel for reduction(min:result)
    for (int i = 0; i < n; i++) {
        if (arr[i] < result) {
            result = arr[i];
        }
    }
    return result;
}

// Function to perform parallel reduction for max operation
int parallelMax(int arr[], int n) {
    int result = arr[0];
    #pragma omp parallel for reduction(max:result)
    for (int i = 0; i < n; i++) {
```

```
        if (arr[i] > result) {
            result = arr[i];
        }
    }
    return result;
}

// Function to perform parallel reduction for sum operation
int parallelSum(int arr[], int n) {
    int result = 0;
    #pragma omp parallel for reduction(+:result)
    for (int i = 0; i < n; i++) {
        result += arr[i];
    }
    return result;
}

// Function to perform parallel reduction for average operation
double parallelAverage(int arr[], int n) {
    int sum =  parallelSum(arr,  n);
    return static_cast<double>(sum) / n;
}

int main() {
    int n = 10000;
```

```
    int arr[n];
    for (int i = 0; i < n; i++) {
        arr[i] = rand() % 1000;
    }

    // Min Operation
    int min = parallelMin(arr, n);
    cout << "Min: " << min << endl;

    // Max Operation
    int max = parallelMax(arr, n);
    cout << "Max: " << max << endl;

    // Sum Operation
    int sum = parallelSum(arr, n);
    cout << "Sum: " << sum << endl;

    // Average Operation
    double avg = parallelAverage(arr, n);
    cout << "Average: " << avg << endl;

    return 0;
}
```

```
Min: 0
Max: 998
Sum: 4977764
Average: 497.776
```

Name: Mohit Bhavsar

RollNo: C43435

Batch: B9

Assignment No: 6

```
# Load libraries

import numpy as np

import pylab as pl

from sklearn import datasets

from sklearn.tree import DecisionTreeRegressor


##################################

### ADD EXTRA LIBRARIES HERE ###

##################################

from sklearn.metrics import
mean_squared_error,median_absolute_error,r2_score,mean_absolute_error

from sklearn import grid_search

from sklearn.cross_validation import train_test_split


def load_data():

    """Load the Boston dataset."""


    boston = datasets.load_boston()

    return boston
```

```
def explore_city_data(city_data):
    """Calculate the Boston housing statistics."""

    # Get the labels and features from the housing data
    housing_prices  =  city_data.target
    housing_features = city_data.data

    # Please calculate the following values using the Numpy library
    # Size of data (number of houses)?
    # Number of features?
    # Minimum price?
    # Maximum price?
    # Calculate mean price?
    # Calculate median price?
    # Calculate standard deviation?
    number_of_houses = housing_features.shape[0]
    number_of_features = housing_features.shape[1]
    max_price = np.max(housing_prices)
    min_price = np.min(housing_prices)
    mean_price = np.mean(housing_prices)
    median_price = np.median(housing_prices)
    standard_deviation = np.std(housing_prices)

    print "number of houses:",number_of_houses
```

```
    print "number of features:",number_of_features
    print "max price of house:",max_price
    print "min price of house:",min_price
    print "mean price of house:",mean_price
    print "median price of house:",median_price
    print "standard deviation for prices of house:",standard_deviation


def performance_metric(label, prediction):
    """Calculate and return the appropriate error performance metric."""


    # http://scikit-learn.org/stable/modules/classes.html#sklearn-metrics-metrics
    #return median_absolute_error(label, prediction)
    #return r2_score(label, prediction)
    #return mean_absolute_error(label, prediction)
    return mean_squared_error(label,prediction)
    pass



def split_data(city_data):
    """Randomly shuffle the sample set. Divide it into 70 percent training and 30
percent testing data."""


    # Get the features and labels from the Boston housing data
    X, y = city_data.data, city_data.target
    X_train, X_test, y_train, y_test = train_test_split(
```

```
        X, y, test_size=0.30, train_size=0.70, random_state=42)
    return X_train, y_train, X_test, y_test


def learning_curve(depth, X_train, y_train, X_test, y_test):
    """Calculate the performance of the model after a set of training data."""

    # We will vary the training set size so that we have 50 different sizes
    sizes = np.linspace(1, len(X_train), 50)
    train_err = np.zeros(len(sizes))
    test_err = np.zeros(len(sizes))

    print "Decision Tree with Max Depth: "
    print depth



    for i, s in enumerate(sizes):

        # Create and fit the decision tree regressor model
        regressor = DecisionTreeRegressor(max_depth=depth)
        regressor.fit(X_train[:s], y_train[:s])

        # Find the performance on the training and testing set
```

```
        train_err[i] = performance_metric(y_train[:s],
regressor.predict(X_train[:s]))


        test_err[i] = performance_metric(y_test, regressor.predict(X_test))




    pl.figure()
    pl.plot(y_train - regressor.predict(X_train))
    pl.savefig("residual_plot.png")
    # Plot learning curve graph
    learning_curve_graph(sizes, train_err, test_err, depth)



def learning_curve_graph(sizes, train_err, test_err, depth):
    """Plot training and test error as a function of the training size."""

    pl.figure()
    pl.title('Decision Trees: Performance vs Training Size')
    pl.plot(sizes, test_err, lw=2, label = 'test error')
    pl.plot(sizes, train_err, lw=2, label = 'training error')
    pl.legend()
    pl.xlabel('Training Size')
    pl.ylabel('Error')
    #pl.show()
```

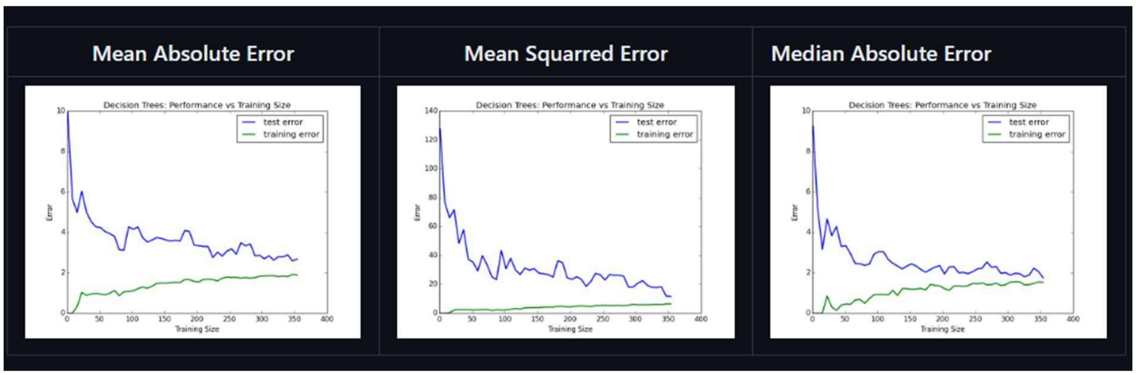
```
    pl.savefig("learning_curve"+"_"+str(depth)+".png")


def model_complexity(X_train, y_train, X_test, y_test):
    """Calculate the performance of the model as model complexity increases."""

    print "Model Complexity: "

    # We will vary the depth of decision trees from 2 to 25
    max_depth = np.arange(1, 25)
    train_err = np.zeros(len(max_depth))
    test_err = np.zeros(len(max_depth))

    for i, d in enumerate(max_depth):
        # Setup a Decision Tree Regressor so that it learns a tree with depth d
        regressor = DecisionTreeRegressor(max_depth=d)

        # Fit the learner to the training data
        regressor.fit(X_train, y_train)

        # Find the performance on the training set
        train_err[i] = performance_metric(y_train, regressor.predict(X_train))

        # Find the performance on the testing set
        test_err[i] = performance_metric(y_test, regressor.predict(X_test))
```

```
    # Plot the model complexity graph
    model_complexity_graph(max_depth, train_err, test_err)


def model_complexity_graph(max_depth, train_err, test_err):
    """Plot training and test error as a function of the depth of the decision tree
learn."""

    pl.figure()
    pl.title('Decision Trees: Performance vs Max Depth')
    pl.plot(max_depth, test_err, lw=2, label = 'test error')
    pl.plot(max_depth, train_err, lw=2, label = 'training error')
    pl.legend()
    pl.xlabel('Max Depth')
    pl.ylabel('Error')
    #pl.show()
    pl.savefig("model_complexity.png")


def fit_predict_model(city_data):
    """Find and tune the optimal model. Make a prediction on housing data."""

    # Get the features and labels from the Boston housing data
    X, y = city_data.data, city_data.target
```

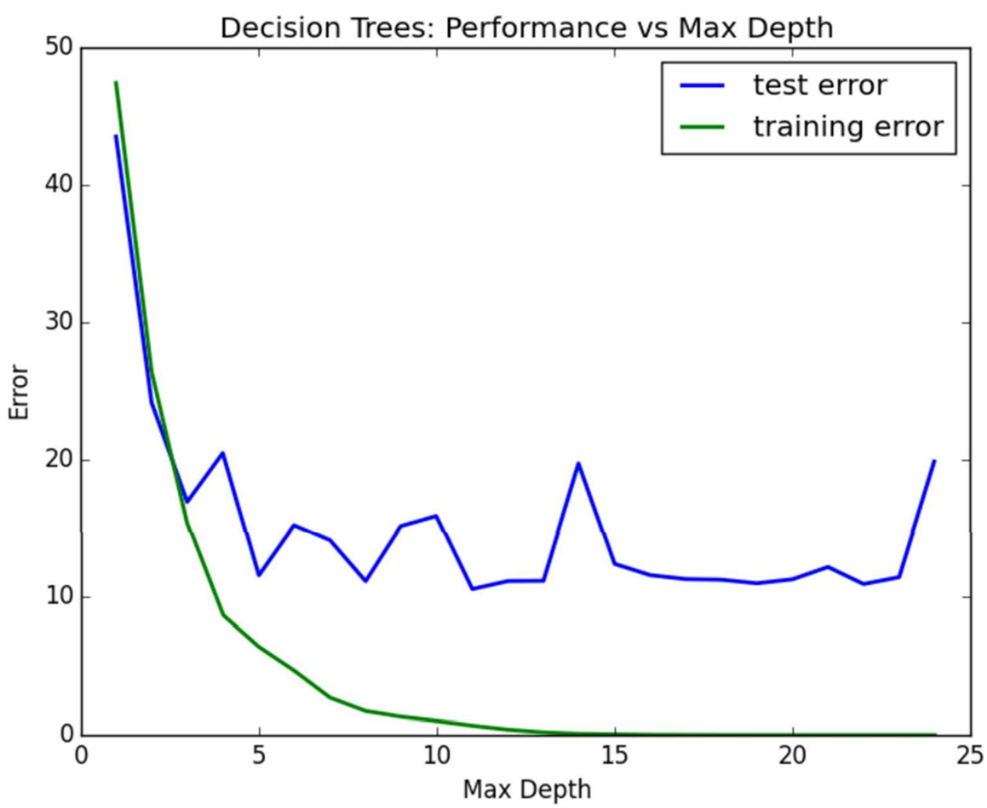
```
# Setup a Decision Tree Regressor
regressor = DecisionTreeRegressor()

parameters = {'max_depth':(1,2,3,4,5,6,7,8,9,10),
    'min_samples_split': (1, 2, 3),
    'min_samples_leaf': (1, 2, 3)
}

regressors = grid_search.GridSearchCV(regressor, parameters,
scoring='mean_squared_error')

regressors.fit(X,y)

# pick the best
reg = regressors.best_estimator_

# Fit the learner to the training data
print "Final Model: "
print reg.fit(X, y)

# Use the model to predict the output of a particular sample
x = [11.95, 0.00, 18.100, 0, 0.6590, 5.6090, 90.00, 1.385, 24, 680.0, 20.20,
332.09, 12.13]
y = reg.predict(x)
print "House: " + str(x)
```

```
    print "Prediction: " + str(y)



def main():
    """Analyze the Boston housing data. Evaluate and validate the
    performanance of a Decision Tree regressor on the housing data.
    Fine tune the model to make prediction on unseen data."""

    # Load data
    city_data = load_data()

    # Explore the data
    explore_city_data(city_data)

    # Training/Test dataset split
    X_train, y_train, X_test, y_test = split_data(city_data)

    # Learning Curve Graphs
    max_depths = [1,2,3,4,5,6,7,8,9,10]
    for max_depth in max_depths:
        learning_curve(max_depth, X_train, y_train, X_test, y_test)

    # Model Complexity Graph
    model_complexity(X_train, y_train, X_test, y_test)
```

```
    # Tune and predict Model
    fit_predict_model(city_data)


if __name__== "__main__":
    main()
```




Decision Trees: Performance vs Training Size
test error
training error
Error
Training Size
0
20
40
60
80
100
120
140
0
50
100
150
200
250
300
350
400


Decision Trees: Performance vs Max Depth
test error
training error
Error
Max Depth
0
10
20
30
40
50
0
5
10
15
20
25

Name: Mohit Bhavsar

RollNo: C43435

Batch: B9

Assignment No: 7

```
import numpy as np

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, Dropout

from tensorflow.keras.optimizers import RMSprop

from tensorflow.keras.datasets import mnist

import matplotlib.pyplot as plt

from sklearn import metrics


# Load the OCR dataset


# The MNIST dataset is a built-in dataset provided by Keras.

# It consists of 70,000 28x28 grayscale images, each of which displays a single handwritten digit from 0 to 9.

# The training set consists of 60,000 images, while the test set has 10,000 images.


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


# X_train and X_test are our array of images while y_train and y_test are our array of labels for each image.

# The first tuple contains the training set features (X_train) and the training set labels (y_train).
```

The second tuple contains the testing set features (X_test) and the testing set labels (y_test).

For example, if the image shows a handwritten 7, then the label will be the intger 7.

plt.imshow(x_train[0], cmap='gray') # imshow() function which simply displays an image.

plt.show() # cmap is responsible for mapping a specific colormap to the values found in the array that you passed as the first argument.

image appears black and white and that each axis of the plot ranges from 0 to 28.

This is because of the format that all the images in the dataset have:

1. All the images are grayscale, meaning they only contain black, white and grey.

2. The images are 28 pixels by 25 pixels in size (28x28).

print(x_train[0])

image data is just an array of digits. You can almost make out a 5 from the pattern of the digits in the array.

Array of 28 values

a grayscale pixel is stored as a digit between 0 and 255 where 0 is black, 255 is white and values in between are different shades of gray.

Therefore, each value in the [28][28] array tells the computer which color to put in that position when we display the actual image.

reformat our X_train array and our X_test array because they do not have the correct shape.

Reshape the data to fit the model

print("X_train shape", x_train.shape)

print("y_train shape", y_train.shape)

print("X_test shape", x_test.shape)

print("y_test shape", y_test.shape)

Here you can see that for the training sets we have 60,000 elements and the testing sets have 10,000 elements.

y_train and y_test only have 1 dimensional shapes because they are just the labels of each element.

x_train and x_test have 3 dimensional shapes because they have a width and height (28x28 pixels) for each element.

(60000, 28, 28) 1st parameter in the tuple shows us how much image we have 2nd and 3rd parameters are the pixel values from x to y (28x28)

The pixel value varies between 0 to 255.

(60000,) Training labels with integers from 0-9 with dtype of uint8. It has the shape (60000,).

(10000, 28, 28) Testing data that consists of grayscale images. It has the shape (10000, 28, 28) and the dtype of uint8. The pixel value varies between 0 to 255.

(10000,) Testing labels that consist of integers from 0-9 with dtype uint8. It has the shape (10000,).

X: Training data of shape (n_samples, n_features)

y: Training label values of shape (n_samples, n_labels)

2D array of height and width, 28 pixels by 28 pixels will just become 784 pixels (28 squared).

Remember that X_train has 60,000 elemenets, each with 784 total pixels so will become shape (60000, 784).

Whereas X_test has 10,000 elements, each with each with 784 total pixels so will become shape (10000, 784).

x_train = x_train.reshape(60000, 784)

x_test = x_test.reshape(10000, 784)

x_train = x_train.astype('float32') # use 32-bit precision when training a neural network, so at one point the training data will have to be converted to 32 bit floats. Since the dataset fits easily in RAM, we might as well convert to float immediately.

x_test = x_test.astype('float32')

x_train /= 255 # Each image has Intensity from 0 to 255

x_test /= 255

Regarding the division by 255, this is the maximum value of a byte (the input feature's type before the conversion to float32),

so this will ensure that the input features are scaled between 0.0 and 1.0.

USING svm-https://mgta.gmu.edu/courses/ml-with-python/handwrittenDigitRecognition.php#:~:text=Remember%20that%20X_train%20has%2060%2C000,keras.

Convert class vectors to binary class matrices

num_classes = 10

y_train = np.eye(num_classes)[y_train] # Return a 2-D array with ones on the diagonal and zeros elsewhere.

y_test = np.eye(num_classes)[y_test] # f your particular categories is present then it mark as 1 else 0 in remain row

Define the model architecture

model = Sequential()

model.add(Dense(512, activation='relu', input_shape=(784,))) # The input_shape argument is passed to the foremost layer. It comprises of a tuple shape,

model.add(Dropout(0.2)) # DROP OUT RATIO 20%

model.add(Dense(512, activation='relu')) #returns a sequence of vectors of dimension 512

model.add(Dropout(0.2))

model.add(Dense(num_classes, activation='softmax'))

Compile the model

model.compile(loss='categorical_crossentropy', # for a multi-class classification problem

optimizer=RMSprop(),

metrics=['accuracy'])

Train the model

batch_size = 128 # batch_size argument is passed to the layer to define a batch size for the inputs.

epochs = 20

history = model.fit(x_train, y_train,

batch_size=batch_size,

epochs=epochs,

verbose=1, # verbose=1 will show you an animated progress bar eg. [==========]

validation_data=(x_test, y_test)) # Using validation_data means you are providing the training set and validation set yourself,

```
                                        # validation_split means you only provide a
training set and keras splits it into a training set and a validation set


# Evaluate the model

score = model.evaluate(x_test, y_test, verbose=0)

print('Test loss:', score[0])

print('Test accuracy:', score[1])
```

Name: Mohit Bhavsar

RollNo: C43435

Batch: B9

Assignment No: 8

```
import tensorflow as tf

import matplotlib.pyplot as plt

from tensorflow import keras

import numpy as np


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


# There are 10 image classes in this dataset and each class has a mapping corresponding to the following labels:


#0 T-shirt/top

#1 Trouser

#2 pullover

#3 Dress

#4 Coat

#5 sandals

#6 shirt

#7 sneaker

#8 bag

#9 ankle boot
```

https://ml-course.github.io/master/09%20-%20Convolutional%20Neural%20Networks.pdf

plt.imshow(x_train[1])

plt.imshow(x_train[0])

Next, we will preprocess the data by scaling the pixel values to be between 0 and 1, and then reshaping the images to be 28x28 pixels.

x_train = x_train.astype('float32') / 255.0

x_test = x_test.astype('float32') / 255.0

x_train = x_train.reshape(-1, 28, 28, 1)

x_test = x_test.reshape(-1, 28, 28, 1)

28, 28 comes from width, height, 1 comes from the number of channels

-1 means that the length in that dimension is inferred.

This is done based on the constraint that the number of elements in an ndarray or Tensor when reshaped must remain the same.

each image is a row vector (784 elements) and there are lots of such rows (let it be n, so there are 784n elements). So TensorFlow can infer that -1 is n.

converting the training_images array to 4 dimensional array with sizes 60000, 28, 28, 1 for 0th to 3rd dimension.

x_train.shape

x_test.shape

y_train.shape

y_test.shape

We will use a convolutional neural network (CNN) to classify the fashion items.

The CNN will consist of multiple convolutional layers followed by max pooling,

dropout, and dense layers. Here is the code for the model:

```
model = keras.Sequential([

    keras.layers.Conv2D(32, (3,3), activation='relu', input_shape=(28,28,1)),

    # 32 filters (default), randomly initialized

    # 3*3 is Size of Filter

    # 28,28,1 size of Input Image

    # No zero-padding: every output 2 pixels less in every dimension

    # in Paramter shwon 320 is value of weights: (3x3 filter weights + 32 bias) *
32 filters

    # 32*3*3=288(Total)+32(bias)= 320




    keras.layers.MaxPooling2D((2,2)),

    # It shown 13 * 13 size image with 32 channel or filter or depth.


    keras.layers.Dropout(0.25),

    # Reduce Overfitting of Training sample drop out 25% Neuron


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

    # Deeper layers use 64 filters
```

3*3 is Size of Filter

Observe how the input image on 28x28x1 is transformed to a 3x3x64 feature map

13(Size)-3(Filter Size)+1(bias)=11 Size for Width and Height with 64 Depth or filtter or channel

in Paramter shwon 18496 is value of weights: (3x3 filter weights + 64 bias) * 64 filters

64*3*3=576+1=577*32 + 32(bias)=18496

keras.layers.MaxPooling2D((2,2)),

It shown 5 * 5 size image with 64 channel or filter or depth.

keras.layers.Dropout(0.25),

keras.layers.Conv2D(128, (3,3), activation='relu'),

Deeper layers use 128 filters

3*3 is Size of Filter

Observe how the input image on 28x28x1 is transformed to a 3x3x128 feature map

It show 5(Size)-3(Filter Size)+1(bias)=3 Size for Width and Height with 64 Depth or filtter or channel

128*3*3=1152+1=1153*64 + 64(bias)= 73856

To classify the images, we still need a Dense and Softmax layer.

We need to flatten the 3x3x128 feature map to a vector of size 1152

https://medium.com/@iamvarman/how-to-calculate-the-number-of-parameters-in-the-cnn-5bd55364d7ca

```
    keras.layers.Flatten(),

    keras.layers.Dense(128, activation='relu'),

    # 128 Size of Node in Dense Layer

    # 1152*128 = 147584

    keras.layers.Dropout(0.25),

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

    # 10 Size of Node another Dense Layer

    # 128*10+10 bias= 1290

])

model.summary()

# Compile and Train the Model

# After defining the model, we will compile it and train it on the training data.

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

history = model.fit(x_train, y_train, epochs=10, validation_data=(x_test,
y_test))

# 1875 is a number of batches. By default batches contain 32 samles.60000 / 32
= 1875

# Finally, we will evaluate the performance of the model on the test data.

test_loss, test_acc = model.evaluate(x_test, y_test)

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