# Assignment No. 1

## Depth First Search

Code -

#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();

if (!visited[curr\_node]) { visited[curr\_node] = true;

s.pop(); 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,no. of Edges and Starting Node of graph:\n"; cin >> n >> m >> start\_node;

//n: node,m:edges

cout<<"Enter pair of node and edges:\n";

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;

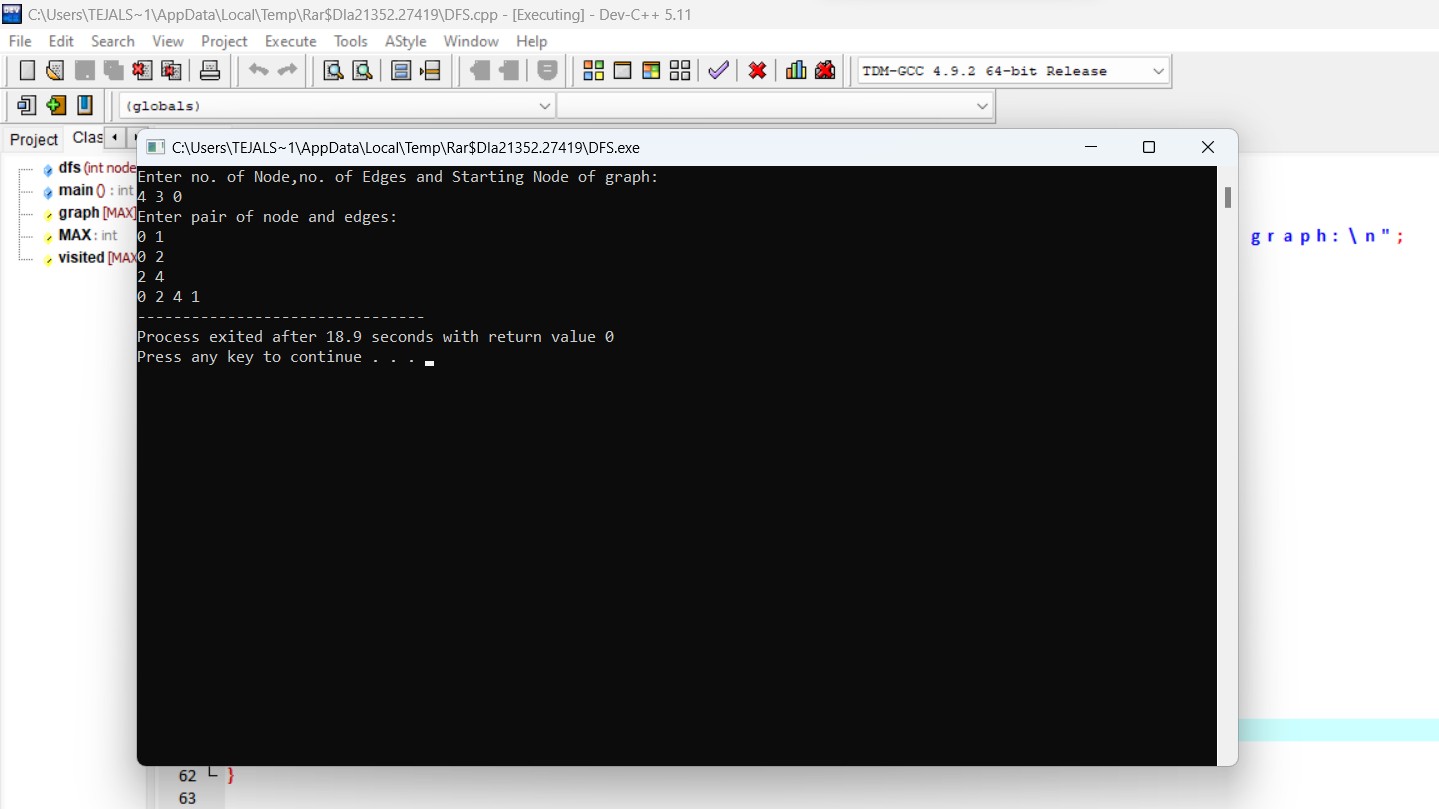
}

dfs(start\_node);

return 0;

}

Output -



## Breadth First Search

Code -

#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)

{

}

else

{

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

q.push(temp->left);

}

if(temp->right==NULL)

{

}

else

{

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

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);

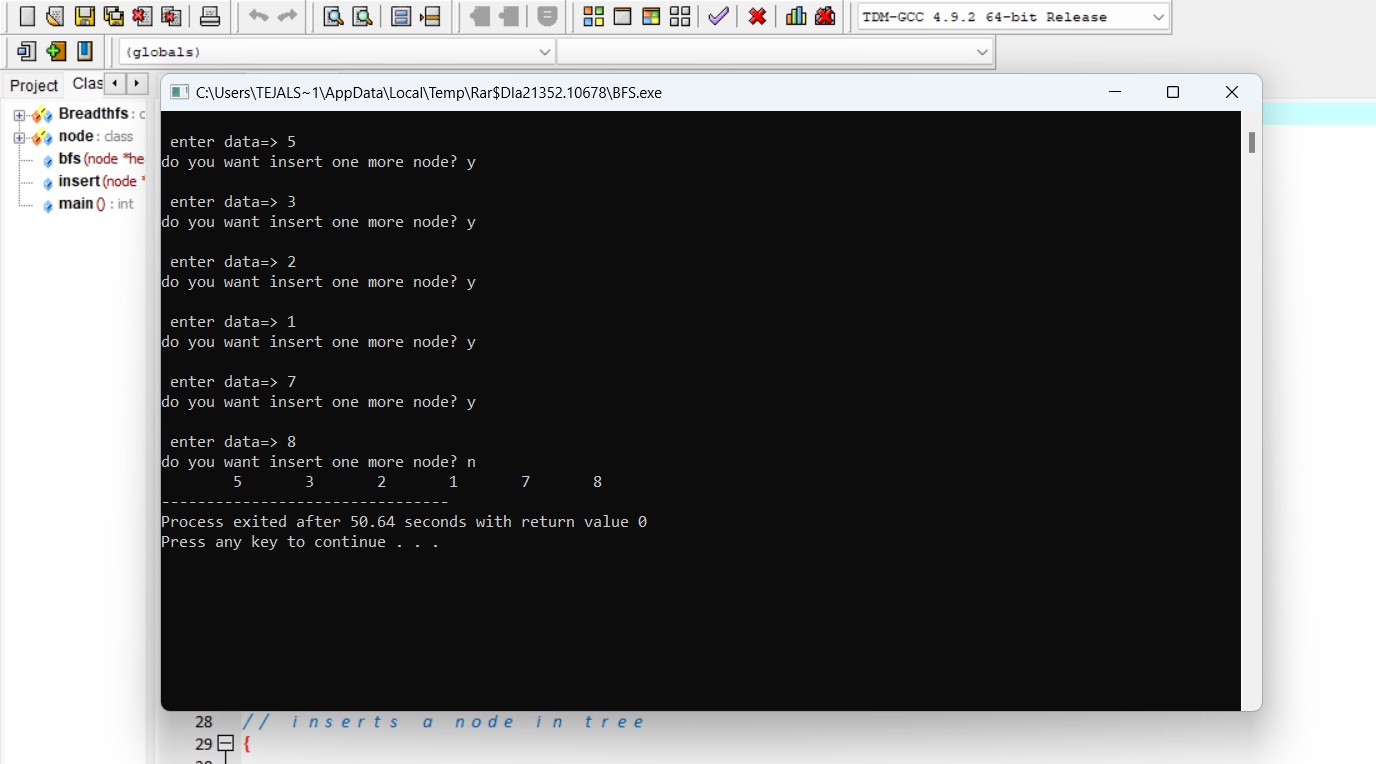
cout<<"do you want insert one more node?"; cin>>ans;

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

return 0;

}

Output -



# Assignment No. 2

## Bubble Sort

Code -

import time import random

#timer to keep track of performance start = time.perf\_counter()

# a function to implement bubble sort in parallel def Parallel\_bubble\_sort(lst):

# variable to keep track of swaps to end the while loop Sorted = 0

# variable to get length of list n = len(lst)

#loop to traverse all list elements in phases while Sorted == 0:

# set to 1 initially to assume list is sorted # and no swaps occurred

Sorted = 1

# traverse all list elements in pair # start at index 0 for odd phase # start at index 1 for even phase

for i in range(0, n-1, 2):

# check if current element greater than next element if lst[i] > lst[i+1]:

# if so, swap the elements

lst[i], lst[i+1] = lst[i+1], lst[i]

# set to 0 to imply a swap occurred

Sorted = 0 for i in range(1, n-1, 2):

if lst[i] > lst[i+1]:

lst[i], lst[i+1] = lst[i+1], lst[i] Sorted = 0

# print final sorted list print(lst)

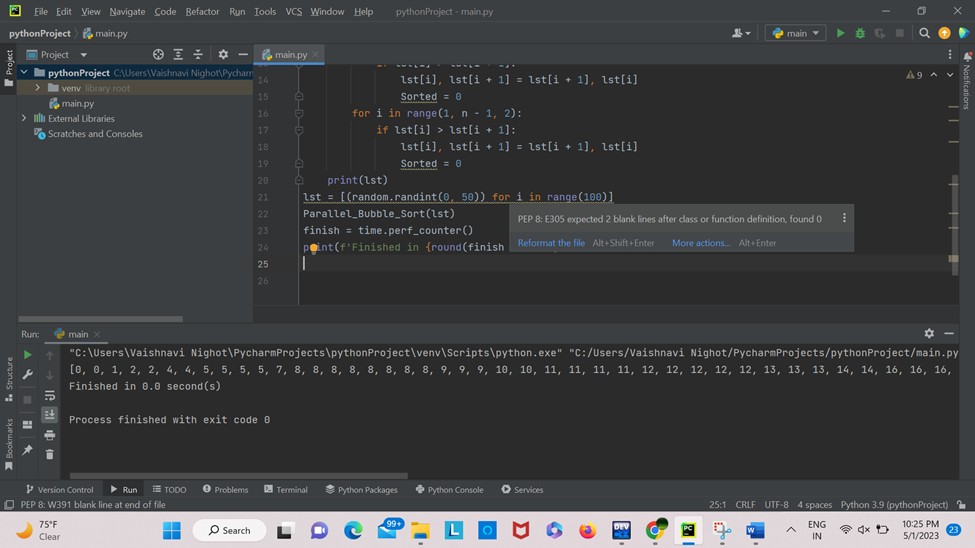
# an example list to test above program

lst = [(random.randint(0,100)) for i in range(100)] Parallel\_Bubble\_Sort(lst)

finish = time.perf\_counter()

print(f'Finished in {round(finish-start,2)} second(s)')

Output -



## Merge Sort

Code -

def merge(arr, l, m, r):

n1 = m - l + 1 n2 = r - m

# create temp arrays L = [0] \* (n1)

R = [0] \* (n2)

# Copy data to temp arrays L[] and R[] for i in range(0, n1):

L[i] = arr[l + i]

for j in range(0, n2):

R[j] = arr[m + 1 + j]

# Merge the temp arrays back into arr[l..r] i = 0 # Initial index of first subarray

j = 0 # Initial index of second subarray k = l # Initial index of merged subarray

while i < n1 and j < n2:

if L[i] <= R[j]:

arr[k] = L[i] i += 1

else:

k += 1

arr[k] = R[j] j += 1

# Copy the remaining elements of L[], if there # are any

while i < n1:

arr[k] = L[i] i += 1

k += 1

# Copy the remaining elements of R[], if there # are any

while j < n2:

arr[k] = R[j]

j += 1

k += 1

# l is for left index and r is right index of the # sub-array of arr to be sorted

def mergeSort(arr, l, r): if l < r:

# Same as (l+r)//2, but avoids overflow for # large l and h

m = l+(r-l)//2

# Sort first and second halves mergeSort(arr, l, m) mergeSort(arr, m+1, r) merge(arr, l, m, r)

# Driver code to test above arr = [12, 11, 13, 5, 6, 7]

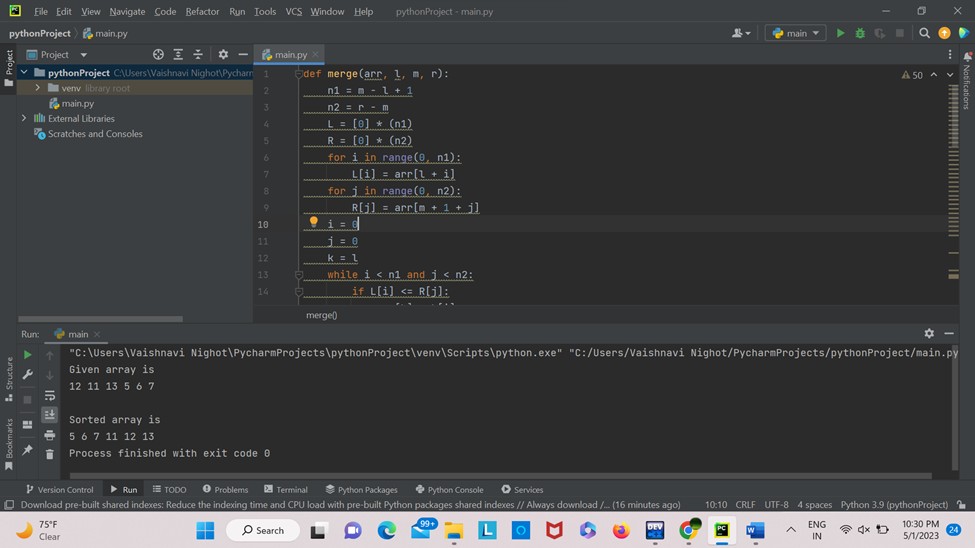
n = len(arr) print("Given array is") for i in range(n):

print("%d" % arr[i],end=" ")

mergeSort(arr, 0, n-1) print("\n\nSorted array is") for i in range(n):

print("%d" % arr[i],end=" ")

Output -



# Assignment No. 3

## Min, Max, Sum and Average operations using Parallel Reduction

Code -

#include <iostream> #include <vector> #include <omp.h> #include <climits>

using namespace std;

void min\_reduction(vector<int>& arr) { int min\_value = INT\_MAX;

#pragma omp parallel for reduction(min: min\_value) for (int i = 0; i < arr.size(); i++) {

if (arr[i] < min\_value) { min\_value = arr[i];

}

}

cout << "Minimum value: " << min\_value << endl;

}

void max\_reduction(vector<int>& arr) { int max\_value = INT\_MIN;

#pragma omp parallel for reduction(max: max\_value) for (int i = 0; i < arr.size(); i++) {

if (arr[i] > max\_value) { max\_value = arr[i];

}

}

cout << "Maximum value: " << max\_value << endl;

}

void sum\_reduction(vector<int>& arr) { int sum = 0;

#pragma omp parallel for reduction(+: sum) for (int i = 0; i < arr.size(); i++) {

sum += arr[i];

}

cout << "Sum: " << sum << endl;

}

void average\_reduction(vector<int>& arr) { int sum = 0;

#pragma omp parallel for reduction(+: sum) for (int i = 0; i < arr.size(); i++) {

sum += arr[i];

}

cout << "Average: " << (double)sum / arr.size() << endl;

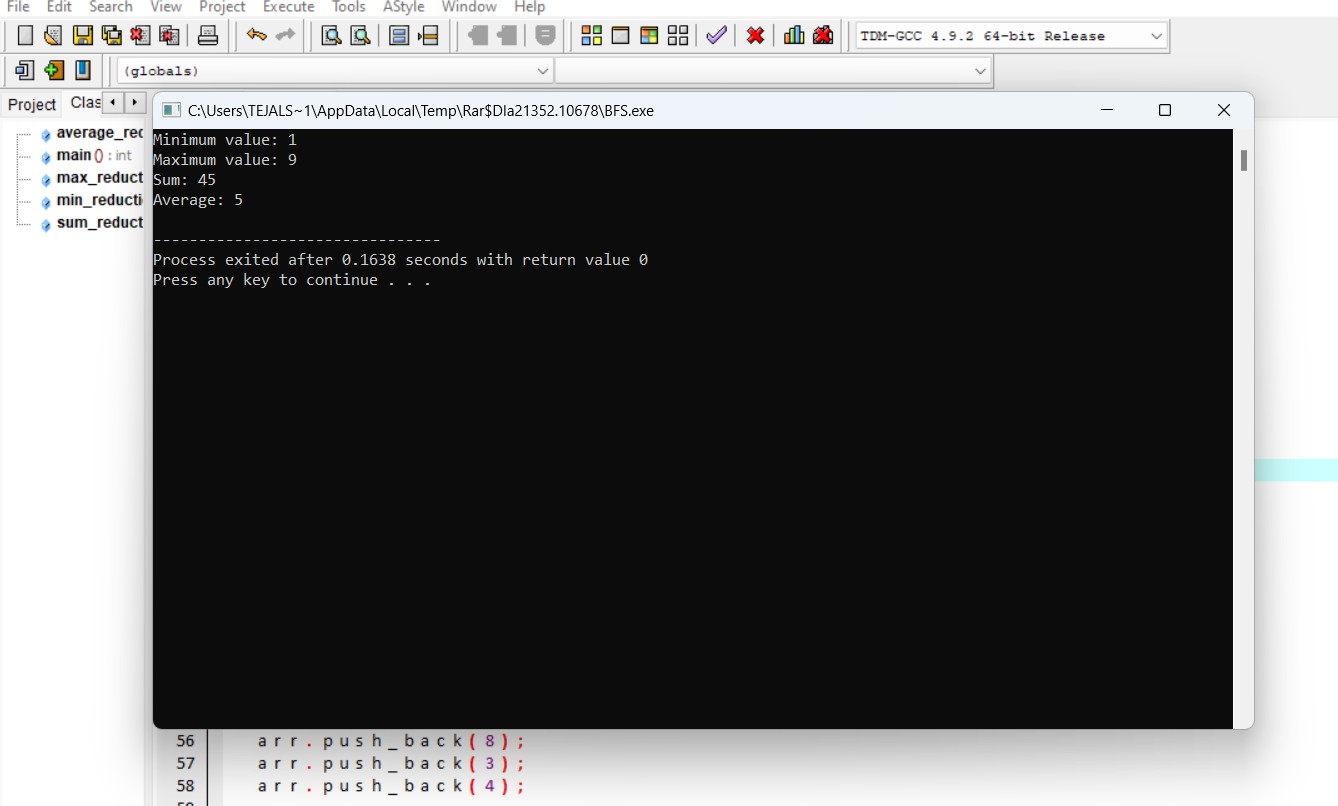
}

int main() { vector<int> arr; arr.push\_back(5); arr.push\_back(2); arr.push\_back(9); arr.push\_back(1); arr.push\_back(7); arr.push\_back(6); arr.push\_back(8); arr.push\_back(3); arr.push\_back(4);

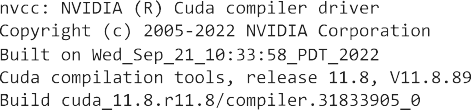
min\_reduction(arr); max\_reduction(arr); sum\_reduction(arr); average\_reduction(arr);

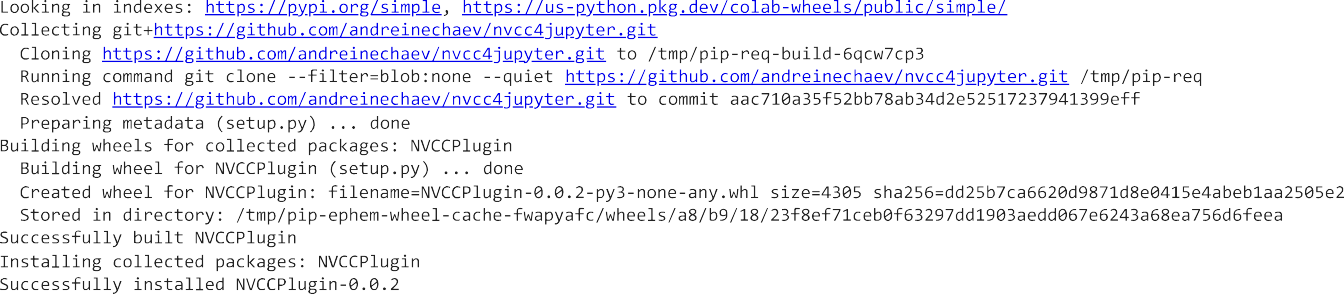
}

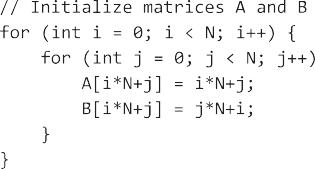
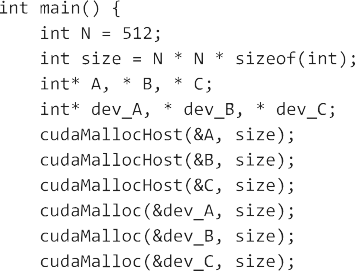
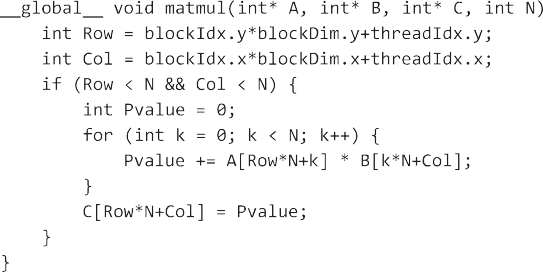
Output -

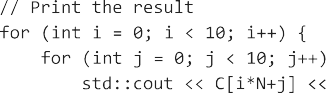






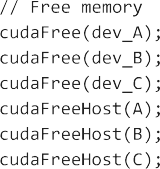


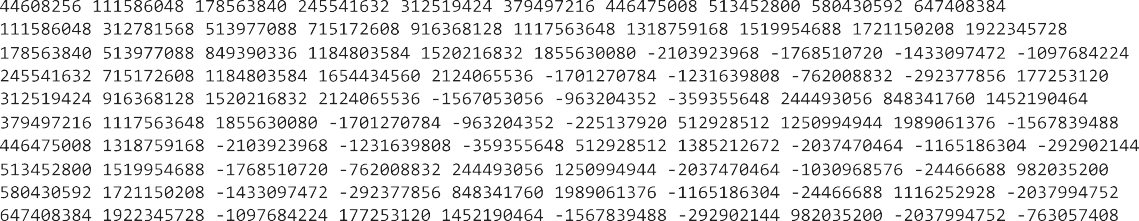


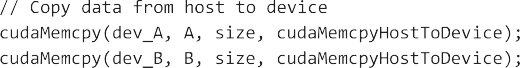
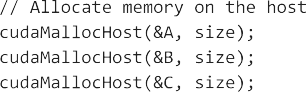
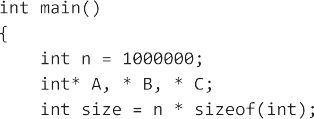


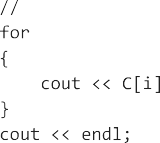
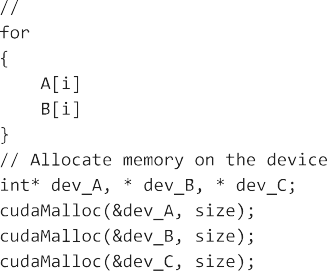
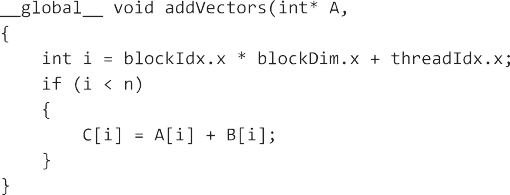




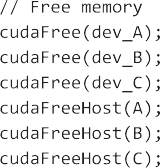


















**Linear Regression on Boston Housing Dataset**

This data was originally a part of UCI Machine Learning Repository and has been removed now. This data also ships with the scikit-learn library. There are 506 samples and 13 feature variables in this data-set. The objective is to predict the value of prices of the house using the given features.

The description of all the features is given below:

**CRIM**: Per capita crime rate by town

**ZN**: Proportion of residential land zoned for lots over 25,000 sq. ft

**INDUS**: Proportion of non-retail business acres per town

**CHAS**: Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)

**NOX**: Nitric oxide concentration (parts per 10 million)

**RM**: Average number of rooms per dwelling

**AGE**: Proportion of owner-occupied units built prior to 1940 **DIS**: Weighted distances to five Boston employment centers **RAD**: Index of accessibility to radial highways

**TAX**: Full-value property tax rate per $10,000

**B**: 1000(Bk - 0.63)², where Bk is the proportion of [people of African American descent] by town

**LSTAT**: Percentage of lower status of the population

**MEDV**: Median value of owner-occupied homes in $1000s I**mport the required Libraries**

In [ ]:

**import** numpy **as** np

**import** matplotlib.pyplot **as** plt

**import** pandas **as** pd

**import** seaborn **as** sns

**%**matplotlib inline

### Load the Boston Housing DataSet from scikit-learn

**from** sklearn.datasets **import** load\_boston boston\_dataset **=** load\_boston()

*# boston\_dataset is a dictionary # let's check what it contains*

boston\_dataset.keys()

Out[2]:

dict\_keys(['data', 'target', 'feature\_names', 'DESCR'])

### Load the data into pandas dataframe

In [3]:

boston **=** pd.DataFrame(boston\_dataset.data, columns**=**boston\_dataset.feature\_names) boston.head()

Out[3]:

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **CRIM** | **ZN** | **INDUS** | **CHAS** | **NOX** | **RM** | **AGE** | **DIS** | **RAD** | **TAX** | **PTRATIO** | **B** |
| **0** 0.00632 | 18.0 | 2.31 | 0.0 | 0.538 | 6.575 | 65.2 | 4.0900 | 1.0 | 296.0 | 15.3 | 396.90 |
| **1** 0.02731 | 0.0 | 7.07 | 0.0 | 0.469 | 6.421 | 78.9 | 4.9671 | 2.0 | 242.0 | 17.8 | 396.90 |
| **2** 0.02729 | 0.0 | 7.07 | 0.0 | 0.469 | 7.185 | 61.1 | 4.9671 | 2.0 | 242.0 | 17.8 | 392.83 |
| **3** 0.03237 | 0.0 | 2.18 | 0.0 | 0.458 | 6.998 | 45.8 | 6.0622 | 3.0 | 222.0 | 18.7 | 394.63 |
| **4** 0.06905 | 0.0 | 2.18 | 0.0 | 0.458 | 7.147 | 54.2 | 6.0622 | 3.0 | 222.0 | 18.7 | 396.90 |

### The target values is missing from the data. Create a new column of target values and add it to dataframe

In [ ]:

boston['MEDV'] **=** boston\_dataset.target

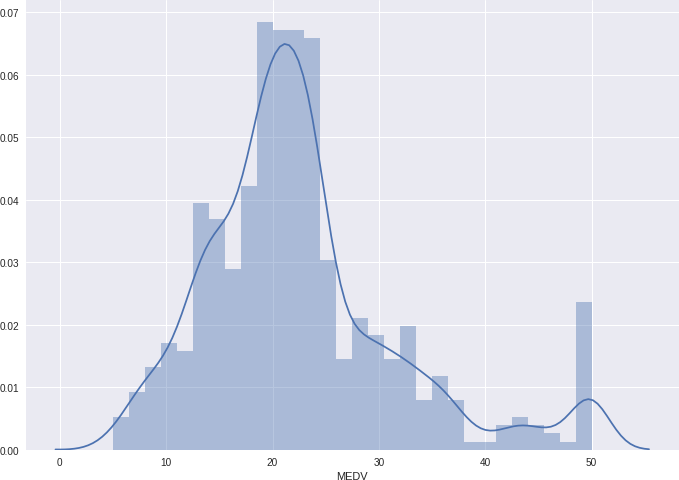
### Data preprocessing

In [5]:

boston.isnull().sum()

|  |  |
| --- | --- |
| Out[5]: |  |
| CRIM | 0 |
| ZN | 0 |
| INDUS | 0 |
| CHAS | 0 |
| NOX | 0 |
| RM | 0 |
| AGE | 0 |
| DIS | 0 |
| RAD | 0 |
| TAX | 0 |
| PTRATIO | 0 |
| B | 0 |
| LSTAT | 0 |
| MEDV | 0 |
| dtype: | int64 |

### Data Visualization

In [6]:

sns.set(rc**=**{'figure.figsize':(11.7,8.27)}) sns.distplot(boston['MEDV'], bins**=**30)

plt.show()

### Correlation matrix

In [ ]:

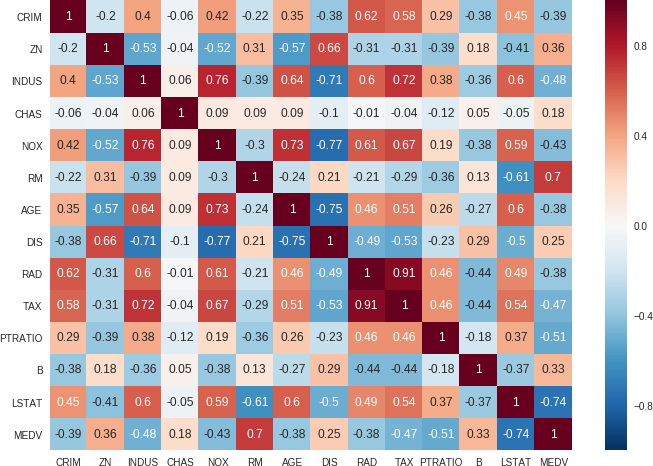
correlation\_matrix **=** boston.corr().round(2)

In [8]:

sns.heatmap(data**=**correlation\_matrix, annot**=True**)

Out[8]:

<matplotlib.axes.\_subplots.AxesSubplot at 0x7f17840122b0>



### Observations

From the above coorelation plot we can see that **MEDV** is strongly correlated to **LSTAT**, **RM**

**RAD** and **TAX** are stronly correlated, so we don't include this in our features together to avoid multi- colinearity

plt.figure(figsize**=**(20, 5))

features **=** ['LSTAT', 'RM'] target **=** boston['MEDV']

**for** i, col **in** enumerate(features):

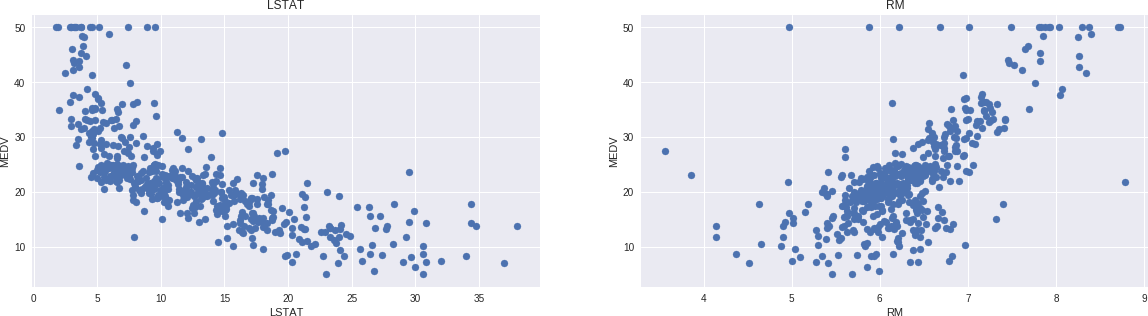
plt.subplot(1, len(features) , i**+**1) x **=** boston[col]

y **=** target

plt.scatter(x, y, marker**=**'o') plt.title(col)

plt.xlabel(col)

plt.ylabel('MEDV')



### Prepare the data for training

In [ ]:

X **=** pd.DataFrame(np.c\_[boston['LSTAT'], boston['RM']], columns **=** ['LSTAT','RM']) Y **=** boston['MEDV']

### Split the data into training and testing sets

In [11]:

**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**=** print(X\_train.shape)

print(X\_test.shape) print(Y\_train.shape) print(Y\_test.shape)

(404, 2)

(102, 2)

(404,)

(102,)

### Train the model using sklearn LinearRegression

**from** sklearn.linear\_model **import** LinearRegression

**from** sklearn.metrics **import** mean\_squared\_error, r2\_score

lin\_model **=** LinearRegression() lin\_model.fit(X\_train, Y\_train)

Out[16]:

LinearRegression(copy\_X=True, fit\_intercept=True, n\_jobs=1, normalize=Fals e)

In [21]:

y\_train\_predict **=** lin\_model.predict(X\_train)

rmse **=** (np.sqrt(mean\_squared\_error(Y\_train, y\_train\_predict))) r2 **=** r2\_score(Y\_train, y\_train\_predict)

print("The model performance for training set") print(" ")

print('RMSE is {}'.format(rmse)) print('R2 score is {}'.format(r2)) print("\n")

y\_test\_predict **=** lin\_model.predict(X\_test)

rmse **=** (np.sqrt(mean\_squared\_error(Y\_test, y\_test\_predict))) r2 **=** r2\_score(Y\_test, y\_test\_predict)

print("The model performance for testing set") print(" ")

print('RMSE is {}'.format(rmse)) print('R2 score is {}'.format(r2))

The model performance for training set

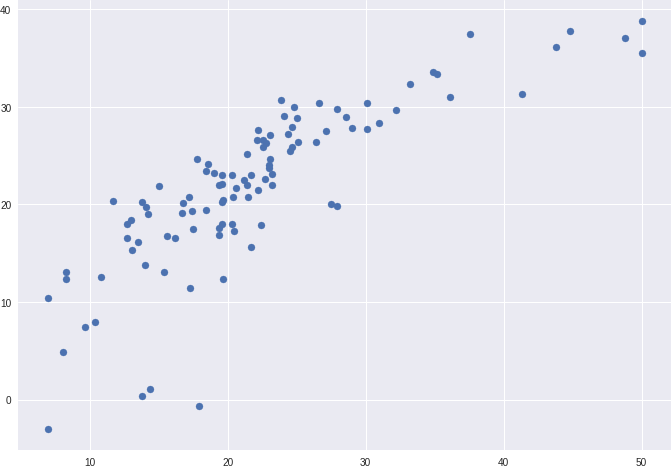
RMSE is 5.6371293350711955

R2 score is 0.6300745149331701

The model performance for testing set

RMSE is 5.137400784702911

R2 score is 0.6628996975186952



*# plotting the y\_test vs y\_pred*

*# ideally should have been a straight line*

plt.scatter(Y\_test, y\_test\_predict) plt.show()

In [1]:

**import** numpy **as** np **import** pandas **as** pd **import** os

In [ ]:

In [3]:

**import** easyocr

**import** cv2

**from** matplotlib **import** pyplot **as** plt

**import** numpy **as** np

In [4]:

image\_path **=** '/kaggle/input/standard-ocr-dataset/data/testing\_data/A/28320.png'

reader **=** easyocr.Reader(['en'], gpu **= False**) result **=** reader.readtext(image\_path)

result

Out[4]:

[([[4, 0], [34, 0], [34, 35], [4, 35]], 'A', 0.9998576691731529)]

In [5]:

top\_left **=** tuple(result[0][0][0])

bottom\_right **=** tuple(result[0][0][2]) text **=** result[0][1]

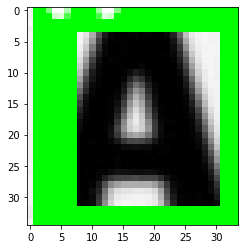
font **=** cv2.FONT\_HERSHEY\_SIMPLEX

img **=** cv2.imread(image\_path)

img **=** cv2.rectangle(img, top\_left, bottom\_right, (0, 255, 0), 5)

img **=** cv2.putText(img, text, top\_left, font, 0.5, (255,255,255), 2, cv2.LINE\_AA) plt.imshow(img)

plt.show()



In [ ]:

**Deep Learning Based Approach**

In [7]:

**import** os

**import** numpy **as** np

**import** matplotlib.pyplot **as** plt

**from** pathlib **import** Path

**from** collections **import** Counter

**import** tensorflow **as** tf

**from** tensorflow **import** keras

**from** tensorflow.keras **import** layers

In [8]:

**!**curl **-**LO https:**//**github.com**/**AakashKumarNain**/**CaptchaCracker**/**raw**/**master**/**captcha\_images\_v2

**!**unzip **-**qq captcha\_images\_v2.zip

% Total % Received % Xferd Average Speed Time Time Time Cu rrent

Dload Upload Total Spent Left Sp

eed

0 0

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 0 | 0 | 0 --:--:-- | --:--:-- | --:--:-- |
| 100 | 8863k | 0 | 0 | 13.5M | 0 --:--:-- | --:--:-- | --:--:-- 1 |

0

100 8863k

3.5M

data\_dir **=** Path("./captcha\_images\_v2/")

In [10]:

images **=** sorted(list(map(str, list(data\_dir.glob("\*.png")))))

labels **=** [img.split(os.path.sep)[**-**1].split(".png")[0] **for** img **in** images] characters **=** set(char **for** label **in** labels **for** char **in** label)

characters **=** sorted(list(characters))

In [11]:

batch\_size **=** 16

img\_width **=** 200

img\_height **=** 50

In [12]:

downsample\_factor **=** 4

max\_length **=** max([len(label) **for** label **in** labels])

In [13]:

char\_to\_num **=** layers.StringLookup(

vocabulary **=** list(characters), mask\_token **= None**

)

num\_to\_char **=** layers.StringLookup(

vocabulary **=** char\_to\_num.get\_vocabulary(), mask\_token **= None**, invert **= True**

)

**def** split\_data(images, labels, train\_size **=** 0.9, shuffle **= True**): size **=** len(images)

indices **=** np.arange(size)

**if** shuffle:

np.random.shuffle(indices)

train\_samples **=** int(size **\*** train\_size)

x\_train, y\_train **=** images[indices[:train\_samples]], labels[indices[:train\_samples]] x\_valid, y\_valid **=** images[indices[train\_samples:]], labels[indices[train\_samples:]]

**return** x\_train, x\_valid, y\_train, y\_valid

x\_train, x\_valid, y\_train, y\_valid **=** split\_data(np.array(images), np.array(labels))

**def** encode\_single\_sample(img\_path, label): img **=** tf.io.read\_file(img\_path)

img **=** tf.io.decode\_png(img, channels **=** 1)

img **=** tf.image.convert\_image\_dtype(img, tf.float32) img **=** tf.image.resize(img, [img\_height, img\_width]) img **=** tf.transpose(img, perm **=** [1,0,2])

label **=** char\_to\_num(tf.strings.unicode\_split(label, input\_encoding **=** "UTF-8"))

**return** {"image" : img, "label" : label}

2023-01-17 05:07:31.665008: I tensorflow/stream\_executor/cuda/cuda\_gpu\_exe cutor.cc:937] successful NUMA node read from SysFS had negative value (-

1), but there must be at least one NUMA node, so returning NUMA node zero 2023-01-17 05:07:31.666296: I tensorflow/stream\_executor/cuda/cuda\_gpu\_exe cutor.cc:937] successful NUMA node read from SysFS had negative value (-

1), but there must be at least one NUMA node, so returning NUMA node zero 2023-01-17 05:07:31.667063: I tensorflow/stream\_executor/cuda/cuda\_gpu\_exe cutor.cc:937] successful NUMA node read from SysFS had negative value (-

1), but there must be at least one NUMA node, so returning NUMA node zero 2023-01-17 05:07:31.667958: I tensorflow/core/platform/cpu\_feature\_guard.c c:142] This TensorFlow binary is optimized with oneAPI Deep Neural Network Library (oneDNN) to use the following CPU instructions in performance-crit ical operations: AVX2 AVX512F FMA

To enable them in other operations, rebuild TensorFlow with the appropriat e compiler flags.

2023-01-17 05:07:31.668991: I tensorflow/stream\_executor/cuda/cuda\_gpu\_exe cutor.cc:937] successful NUMA node read from SysFS had negative value (-

1), but there must be at least one NUMA node, so returning NUMA node zero 2023-01-17 05:07:31.669635: I tensorflow/stream\_executor/cuda/cuda\_gpu\_exe cutor.cc:937] successful NUMA node read from SysFS had negative value (-

1), but there must be at least one NUMA node, so returning NUMA node zero 2023-01-17 05:07:31.670514: I tensorflow/stream\_executor/cuda/cuda\_gpu\_exe cutor.cc:937] successful NUMA node read from SysFS had negative value (-

1), but there must be at least one NUMA node, so returning NUMA node zero 2023-01-17 05:07:33.628909: I tensorflow/stream\_executor/cuda/cuda\_gpu\_exe cutor.cc:937] successful NUMA node read from SysFS had negative value (-

1), but there must be at least one NUMA node, so returning NUMA node zero 2023-01-17 05:07:33.629796: I tensorflow/stream\_executor/cuda/cuda\_gpu\_exe cutor.cc:937] successful NUMA node read from SysFS had negative value (-

1), but there must be at least one NUMA node, so returning NUMA node zero 2023-01-17 05:07:33.630620: I tensorflow/stream\_executor/cuda/cuda\_gpu\_exe cutor.cc:937] successful NUMA node read from SysFS had negative value (-

1), but there must be at least one NUMA node, so returning NUMA node zero 2023-01-17 05:07:33.631308: I tensorflow/core/common\_runtime/gpu/gpu\_devic e.cc:1510] Created device /job:localhost/replica:0/task:0/device:GPU:0 wit h 14969 MB memory: -> device: 0, name: Tesla P100-PCIE-16GB, pci bus id: 0000:00:04.0, compute capability: 6.0

In [14]:

train\_dataset **=** tf.data.Dataset.from\_tensor\_slices((x\_train, y\_train))

train\_dataset **=** (

train\_dataset.map(

encode\_single\_sample, num\_parallel\_calls **=** tf.data.AUTOTUNE

).batch(batch\_size).prefetch(buffer\_size **=** tf.data.AUTOTUNE)

)

validation\_dataset**=** tf.data.Dataset.from\_tensor\_slices((x\_valid, y\_valid)) validation\_dataset **=** (

validation\_dataset.map(

encode\_single\_sample, num\_parallel\_calls **=** tf.data.AUTOTUNE

).batch(batch\_size).prefetch(buffer\_size **=** tf.data.AUTOTUNE)

)

### Model Construction

In [15]:

|  |  |  |
| --- | --- | --- |
| **class** CTCLayer(layers.Layer):  **def** init (self, name **= None**): super(). init (name **=** name)  self.loss\_fn **=** keras.backend.ctc\_batch\_cost  **def** call(self, y\_true, y\_pred):  batch\_len **=** tf.cast(tf.shape(y\_true)[0], dtype **=** 'int64')  input\_length **=** tf.cast(tf.shape(y\_pred)[1], dtype **=** 'int64') label\_length **=** tf.cast(tf.shape(y\_true)[1], dtype **=** 'int64')  input\_length **=** input\_length **\*** tf.ones(shape **=** (batch\_len, 1), dtype **=** 'int64') label\_length **=** label\_length **\*** tf.ones(shape **=** (batch\_len, 1), dtype **=** 'int64')  loss **=** self.loss\_fn(y\_true, y\_pred, input\_length, label\_length) self.add\_loss(loss)  **return** y\_pred  **def** build\_model():  input\_img **=** layers.Input(  shape **=** (img\_width, img\_height, 1), name **=** "image", dtype **=** "float32"  )  labels **=** layers.Input(name **=** "label", shape **=** (**None**,), dtype **=** "float32")  x **=** layers.Conv2D(32, (3,3), activation **=** 'relu', kernel\_initializer **=** "he\_normal", x **=** layers.MaxPooling2D((2,2), name **=** 'pool1')(x)  x **=** layers.Conv2D(64, (3,3), activation **=** 'relu', kernel\_initializer **=** "he\_normal", x **=** layers.MaxPooling2D((2,2), name **=** "pool2")(x)  new\_shape **=** ((img\_width **//** 4), (img\_height **//** 4) **\*** 64)  x **=** layers.Reshape(target\_shape **=** new\_shape, name **=** "reshape")(x) x **=** layers.Dense(64, activation **=** 'relu', name **=** 'dense1')(x)  x **=** layers.Dropout(0.2)(x)  x **=** layers.Bidirectional(layers.LSTM(128, return\_sequences **= True**, dropout **=** 0.25))( x **=** layers.Bidirectional(layers.LSTM(64, return\_sequences **= True**, dropout **=** 0.25))(x  x **=** layers.Dense(len(char\_to\_num.get\_vocabulary()) **+** 1, activation **=** "softmax", name output **=** CTCLayer(name **=** "ctc\_loss")(labels, x)  model **=** keras.models.Model(inputs **=** [input\_img, labels], outputs **=** output, name **=** 'o opt **=** keras.optimizers.Adam()  model.compile(optimizer **=** opt)  **return** model  model **=** build\_model() model.summary() | | |
|  |  |  |

Model: "ocr\_model\_v1"

Layer (type) Output Shape Param # Connected to

==========================================================================

========================

image (InputLayer) [(None, 200, 50, 1)] 0

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Conv1 [0] | (Conv2D) | (None, | 200, | 50, | 32) | 320 | image[0] |
|  |  |  |  |  |  |  |  |
| pool1 [0] | (MaxPooling2D) | (None, | 100, | 25, | 32) | 0 | Conv1[0] |
|  |  |  |  |  |  |  |  |
| Conv2 [0] | (Conv2D) | (None, | 100, | 25, | 64) | 18496 | pool1[0] |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| pool2 [0] | (MaxPooling2D) | (None, | | 50, | 12, 64) | 0 | Conv2[0] |
| reshape (Reshape) [0] | | | (None, | 50, | 768) | 0 | pool2[0] |
|  | | |  |  |  |  |  |
| dense1 (Dense) [0][0] | | | (None, | 50, | 64) | 49216 | reshape |
|  | | |  |  |  |  |  |
| dropout (Dropout) [0] | | | (None, | 50, | 64) | 0 | dense1[0] |
|  | | |  |  |  |  |  |
| bidirectional (Bidirectional) [0][0] | | | (None, | 50, | 256) | 197632 | dropout |
|  | | |  |  |  |  |  |
| bidirectional\_1 (Bidirectional) onal[0][0] | | | (None, | 50, | 128) | 164352 | bidirecti |

label (InputLayer) [(None, None)] 0

dense2 (Dense) (None, 50, 21) 2709 bidirecti onal\_1[0][0]

ctc\_loss (CTCLayer) (None, 50, 21) 0 label[0]

[0]

dense2[0]

[0]

==========================================================================

========================

Total params: 432,725

Trainable params: 432,725

Non-trainable params: 0

In [16]:



Epoch 1/100

2023-01-17 05:07:35.857141: I tensorflow/compiler/mlir/mlir\_graph\_optim ization\_pass.cc:185] None of the MLIR Optimization Passes are enabled

(registered 2)

2023-01-17 05:07:42.771610: I tensorflow/stream\_executor/cuda/cuda\_dnn. cc:369] Loaded cuDNN version 8005

59/59 [==============================] - 17s 57ms/step - loss: 22.6683

- val\_loss: 16.4451 Epoch 2/100

59/59 [==============================] - 2s 32ms/step - loss: 16.3679 -

val\_loss: 16.4390 Epoch 3/100

59/59 [==============================] - 2s 33ms/step - loss: 16.3517 -

val\_loss: 16.4366 Epoch 4/100

59/59 [==============================] - 3s 44ms/step - loss: 16.3495 -

val\_loss: 16.4229 Epoch 5/100

epochs **=** 100

early\_stopping\_patience **=** 10

early\_stopping **=** keras.callbacks.EarlyStopping(

monitor **=** 'val\_loss', patience **=** early\_stopping\_patience, restore\_best\_weights **= Tru**

)

history **=** model.fit( train\_dataset,

validation\_data **=** validation\_dataset, epochs **=** epochs,

callbacks **=** [early\_stopping]

)

In [ ]:

### Inferencing

prediction\_model **=** keras.models.Model(

model.get\_layer(name **=** "image").input, model.get\_layer(name **=** "dense2").outp

)

prediction\_model.summary()

Model: "model"

Layer (type) Output Shape Param #

=================================================================

image (InputLayer) [(None, 200, 50, 1)] 0

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Conv1 | (Conv2D) | (None, | 200, | 50, | 32) | 320 |
| pool1 | (MaxPooling2D) | (None, | 100, | 25, | 32) | 0 |
| Conv2  pool2 | (Conv2D)  (MaxPooling2D) | (None,  (None, | 100,  50, | 25, 64)  12, 64) | | 18496  0 |
| reshape (Reshape) | | (None, | 50, | 768) | 0 | |
| dense1 (Dense) | | (None, | 50, | 64) | 49216 | |
| dropout (Dropout) | | (None, | 50, | 64) | 0 | |
| bidirectional (Bidirectional | | (None, | 50, | 256) | 197632 | |
| bidirectional\_1 (Bidirection | | (None, | 50, | 128) | 164352 | |
| dense2 (Dense) | | (None, | 50, | 21) | 2709 | |

=================================================================

Total params: 432,725

Trainable params: 432,725

Non-trainable params: 0

In [18]:

**def** decode\_batch\_predictions(pred):

input\_len **=** np.ones(pred.shape[0]) **\*** (pred.shape[1])

results **=** keras.backend.ctc\_decode(pred, input\_length **=** input\_len, greedy **= True**)[0] output\_text **=** []

**for** res **in** results:

res **=** tf.strings.reduce\_join(num\_to\_char(res)).numpy().decode("utf-8") output\_text.append(res)

**return** output\_text

**for** batch **in** validation\_dataset.take(1): batch\_images **=** batch['image']

batch\_labels **=** batch['label']

preds **=** prediction\_model.predict(batch\_images) pred\_texts **=** decode\_batch\_predictions(preds)

orig\_texts **=** []

**for** label **in** batch\_labels:

label **=** tf.strings.reduce\_join(num\_to\_char(label)).numpy().decode("utf-8") orig\_texts.append(label)

\_, ax **=** plt.subplots(4, 4, figsize **=** (15,5))

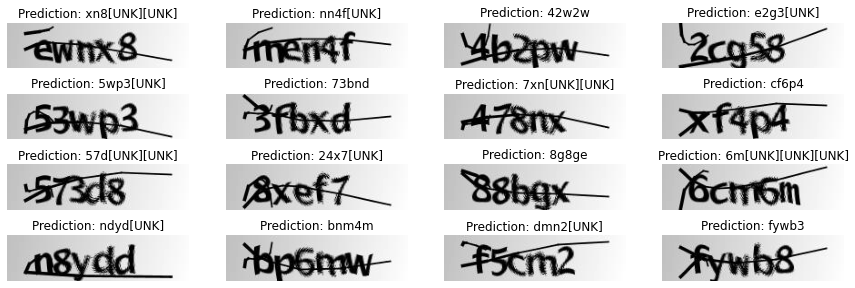
**for** i **in** range(len(pred\_texts)):

img **=** (batch\_images[i, : , : , 0] **\*** 255).numpy().astype(np.uint8)

img **=** img.T

title **=** f"Prediction: {pred\_texts[i]}"

ax[i**//**4, i**%**4].imshow(img,cmap **=** 'gray') ax[i**//**4, i**%**4].set\_title(title)

ax[i**//**4, i**%**4].axis("off") plt.show()

In [ ]:

### OCR Handwriting Recognition

In [19]:

**!**wget **-**q https:**//**git.io**/**J0fjL **-**O IAM\_Words.zip

**!**unzip **-**qq IAM\_Words.zip

**!**mkdir data

**!**mkdir data**/**words

**!**tar **-**xf IAM\_Words**/**words.tgz **-**C data**/**words

**!**mv IAM\_Words**/**words.txt data

In [20]:

**!**head **-**20 data**/**words.txt

# words.txt

----# #

# iam database word information #

# format: a01-000u-00-00 ok 154 1 408 768 27 51 AT A #

# a01-000u-00-00 -> word id for line 00 in form a01-000u # ok -> result of word segmentation

# ok: word was correctly

# er: segmentation of word can be bad #

# 154 -> graylevel to binarize the line containing this wo rd

# 1 -> number of components for this word

# 408 768 27 51 -> bounding box around this word in x,y,w,h format # AT -> the grammatical tag for this word, see the

# file tagset.txt for an explanation # A -> the transcription for this word

#

a01-000u-00-00 ok 154 408 768 27 51 AT A

a01-000u-00-01 ok 154 507 766 213 48 NN MOVE

In [21]:

**from** tensorflow.keras.layers.experimental.preprocessing **import** StringLookup

**from** tensorflow **import** keras

**import** matplotlib.pyplot **as** plt

**import** tensorflow **as** tf

**import** numpy **as** np

**import** os

np.random.seed(42)

tf.random.set\_seed(42)

In [22]:

base\_path **=** "/kaggle/working/data/" words\_list **=** []

words **=** open(f"{base\_path}/words.txt", "r").readlines()

**for** line **in** words:

**if** line[0] **==** "#":

**continue**

**if** line.split(" ")[1] **!=** "err": words\_list.append(line)

len(words\_list)

np.random.shuffle(words\_list)

In [23]:

split\_idx **=** int(0.9 **\*** len(words\_list)) train\_samples **=** words\_list[:split\_idx] test\_samples **=** words\_list[split\_idx:]

val\_split\_idx **=** int(0.5 **\*** len(test\_samples))

validation\_samples **=** test\_samples[:val\_split\_idx] test\_samples **=** test\_samples[val\_split\_idx:]

**assert** len(words\_list) **==** len(train\_samples) **+** len(validation\_samples) **+** len(test\_sample print(f"Total training samples : {len(train\_samples)}")

print(f"Total validation samples: {len(validation\_samples)}")

Total training samples : 86810 Total validation samples: 4823

In [ ]:

In [ ]:

In [ ]:

