Configuration Manual

M.Sc. Research Project
M.Sc. in Data Analytics

Laxman Singh Doliya
Student ID: 20244665

School of Computing
National College of Ireland

Supervisor: Mr. Jorge Basilio

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Laxman Singh Doliya		
Student ID:	20244665		
Programme:	M.Sc. in Data Analytics	**Year:**	2022-23
Module:	M.Sc. Research Project		
Supervisor:	Mr. Jorge Basilio		
Submission Due Date:	15 December 2022		
Project Title:	Application of Graph Theory on Dublin Airport Management		
Word Count:	1123	**Page Count**	11

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Laxman Singh Doliya

Date: 14 December 2022

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Laxman Singh Doliya
ID:20244665
Research Project
MSCDAD JAN22 A

Dec 2022

1 Introduction

The configuration manual provides a comprehensive set of steps to follow in order to replicate the research and verify the findings of the research. The manual is divided into sections that provide specific steps required for each component of the research.

2 System Configuration

Following are the minimum system requirements required for the research.

Operating System	Windows 10 or higher
RAM	16 GB
Minimum Hard Disk Space	128 GB
Processor	Intel 11 Gen, i5 or higher
GPU	Required for faster training of the model.

3 Integrated Development Environment

The following software (Integrated Development Environments) are required:

1. Anaconda Distribution Package: required for running .ipynb notebooks. `https://www.anaconda.com/products/distribution`

2. Python 3.9.7: the programming language. `https://www.python.org/downloads/release/python-397/`

3. Notepad++: for running .py files. `https://notepad-plus-plus.org/downloads/`

4 Python Libraries

The python libraries are the packages required to execute and create prediction models. To install the libraries open the Anaconda prompt and use the following command: **pip install library-name == library-version**
Following is the list of the libraries to install:

- colorama==0.4.4
- cuda-python==11.8.1
- keras==2.9.0
- matplotlib==3.4.3
- numpy==1.20.3
- opencv-python==4.5.5.64
- pandas==1.3.4
- Pillow==8.4.0
- scikit-learn==1.1.2
- scipy==1.7.1
- seaborn==0.11.2
- tensorflow==2.9.1
- torch==1.13.0
- torchvision==0.14.0
- tqdm==4.62.3

5 Dataset

The dataset for the research can be downloaded following the following url: `https://www.kaggle.com/datasets/tthien/shanghaitech`
Create a directory "CrowdDetection". Create a subdirectory "Data" and place the part_A and part_B folders of the Shanghai dataset in this directory.

6 Code File Names

The research contains the following code files:

1. **generateDensityMapsPartA.ipynb** converts part_A image data to NumPy files containing density map information.

2. **generateDensityMapsPartB.ipynb** converts part_B image data to NumPy files containing density map information.

3. **utils.py** contains parameters for normalization of image data.

4. **config.py** contains configuration data for the crowd-detection model like the number of epochs, and batch size.

5. **dataset.py** python file that creates a data loader for the crowd-density detection model

6. **model.py** python file that creates the crowd-detection model.

7. **train.ipynb** the notebook to train the crowd-detection model on the crowd data.

8. **01.Get_Density_Map.ipynb** converts image data to density map image.

9. **02.Convert_Desnity_Map_To_Grid_Graph.ipynb** converts density map image to gird graph representation.

10. **03.A-Star Path Finding Algorithm.ipynb** finds the optimal path using A* path-finding algorithm.

11. **04.Simulation_A_Star.ipynb** verifies the A* path against the path provided by greedy path-finding algorithm.

12. **05.GraphColoring.ipynb** provides different colors for the flight gates

7 Crowd-Detection Model

To create the crowd detection model, we need to run the mentioned files in a sequence. The **root directory** is the ”CrowdDetection” directory we created in the Dataset section. Copy all the code files inside the root directory. The sequence is defined in the below section.

7.1 Data preparation

1. Navigate to the ”Data” folder. Place *generateDensityMapsPartA.ipynb* file into part_A folder and *generateDensityMapsPartB.ipynb* file into the part_B folder inside the data folder. Run these two files in Jupyter and wait until completion as in the figure below.

```
Epoch  27  MAE:  19.42757475527027  Min MAE:  9.402992176104195  Min Epoch:  22
100%|████████████████████████████████████████| 400/400 [1:11:16<00:00, 10.69s/it]
100%|████████████████████████████████████████| 316/316 [19:10<00:00,  3.64s/it]
Epoch  28  MAE:  16.124443114558353  Min MAE:  9.402992176104195  Min Epoch:  22
100%|████████████████████████████████████████| 400/400 [1:11:14<00:00, 10.69s/it]
100%|████████████████████████████████████████| 316/316 [19:04<00:00,  3.62s/it]
Epoch  29  MAE:  30.88859890684297  Min MAE:  9.402992176104195  Min Epoch:  22
```

7.2 dataset.py

This python file creates a data loader used to train and test images. The phase is used to change the folder for training data. By default it is set to *train_data* folder inside *part_B* folder of Data folder.

```python
class CrowdDataset(torch.utils.data.Dataset):
    '''
    CrowdDataset
    '''

    def __init__(self, root, phase, main_transform=None, img_transform=None, dmap_transform=None):
        '''
        root: the root path of dataset.
        phase: train or test.
        main_transform: transforms on both image and density map.
        img_transform: transforms on image.
        dmap_transform: transforms on densitymap.
        '''
        self.img_path = os.path.join(root, phase+'_data','images')
#         print(self.img_path)
        self.dmap_path = os.path.join(root, phase+'_data','densitymaps')
#         print(self.dmap_path)
        self.data_files = [filename for filename in os.listdir(self.img_path)
                           if os.path.isfile(os.path.join(self.img_path, filename))]
        self.main_transform = main_transform
        self.img_transform = img_transform
        self.dmap_transform = dmap_transform
```

The data loader method reads images from the specified folder and creates a tensor of each image. Data augmentation is carried out by calling RandomHorizontalFlip which is controlled by *use_flip* flag, and PairedCrop functions as shown in the figure.

```python
def create_train_dataloader(root, use_flip, batch_size):
    '''
    Create train dataloader.
    root: the dataset root.
    use_flip: True or false.
    batch size: the batch size.
    '''
    main_trans_list = []
    if use_flip:
        main_trans_list.append(RandomHorizontalFlip())
    main_trans_list.append(PairedCrop())
    main_trans = Compose(main_trans_list)
    img_trans = Compose([ToTensor(), Normalize(mean=[0.5,0.5,0.5],std=[0.225,0.225,0.225])])
    dmap_trans = ToTensor()
    dataset = CrowdDataset(root=root, phase='train', main_transform=main_trans,
                    img_transform=img_trans,dmap_transform=dmap_trans)
    dataloader = torch.utils.data.DataLoader(dataset,batch_size=batch_size,shuffle=True)
    return dataloader
```

7.3 model.py

The model.py actually creates the CSRNet crowd-density detection model. The *frontend_feat* in __init__() method of CSRNet class is the list of size of input layers in the frontend layers of the model. Similarly, *backend_feat* is the list of the size of input layers in the backend layers of the model.

```python
class CSRNet(nn.Module):
    def __init__(self, load_weights=False):
        super(CSRNet, self).__init__()
        self.frontend_feat = [64, 64, 'M', 128, 128,
                              'M', 256, 256, 256, 'M', 512, 512, 512]
        self.backend_feat = [512, 512, 512, 256, 128, 64]
        self.frontend = make_layers(self.frontend_feat)
        self.backend = make_layers(
            self.backend_feat, in_channels=512, dilation=True)
        self.output_layer = nn.Conv2d(64, 1, kernel_size=1)
        if not load_weights:
            mod = models.vgg16(pretrained=True)
            self._initialize_weights()
            fsd = collections.OrderedDict()
            # 10 convlution *(weight, bias) = 20 parameters
#             state_dict (dict) - a dict containing parameters and persistent buffers.
            for i in range(len(self.frontend.state_dict().items())):
                temp_key = list(self.frontend.state_dict().items())[i][0]
                fsd[temp_key] = list(mod.state_dict().items())[i][1]
#             load_state_dict: Copies parameters and buffers from state_dict into this module and its descendants.
            self.frontend.load_state_dict(fsd)
```

The make_layer function creates the sequential CSRNet layers. In an 'M' is encountered in the list of layers, a max-pooling layer is added instead of a convolutional layer. If batch_norm is set to true Batch Normalization layer is added to the model. If dilation is set to True, a dilation layer of size 2 is added to the model, else dilation layer of size 1 is added as shown in the figure.

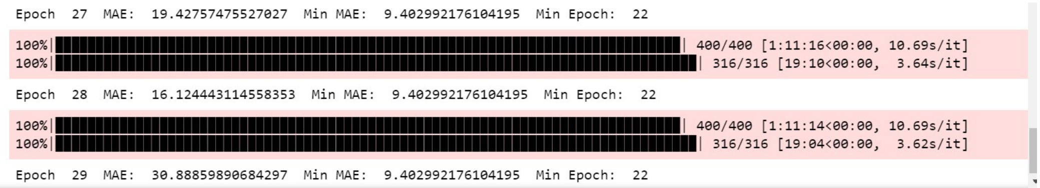
```python
def make_layers(cfg, in_channels=3, batch_norm=False, dilation=False):
    if dilation:
        d_rate = 2
    else:
        d_rate = 1
    layers = []
    for v in cfg:
        if v == 'M':
            layers += [nn.MaxPool2d(kernel_size=2, stride=2)]
        else:
            conv2d = nn.Conv2d(in_channels, v, kernel_size=3,
                               padding=d_rate, dilation=d_rate)
            if batch_norm:
                layers += [conv2d, nn.BatchNorm2d(v), nn.ReLU(inplace=True)]
            else:
                layers += [conv2d, nn.ReLU(inplace=True)]
            in_channels = v
    return nn.Sequential(*layers)
```

7.4 Training Phase

1. Now initialize the parameters in config.py. The parameters inside the __init__ method can be adjusted to control the training phase.

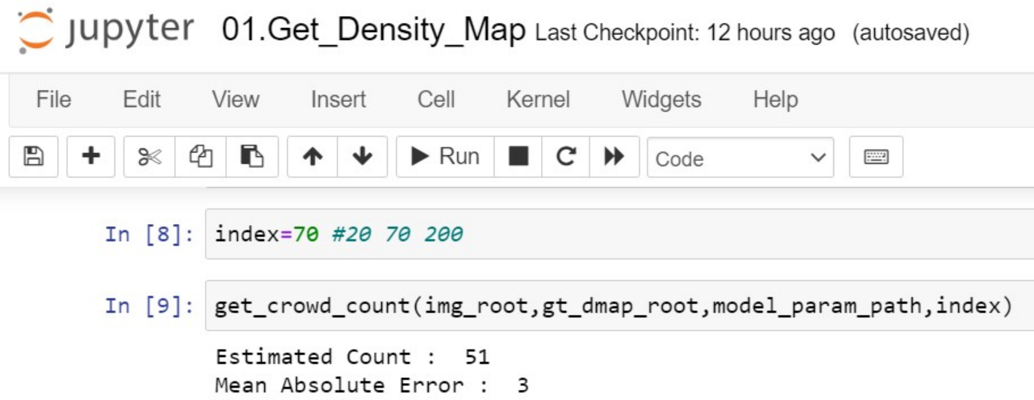
```python
class Config():
    '''
    Config class
    '''
    def __init__(self):
        current_directory = os.getcwd()
        dataset_path = os.path.join(current_directory,"Data","part_B")
        self.dataset_root = dataset_path
        self.device       = torch.device('cuda') if torch.cuda.is_available() else torch.device('cpu')
        self.lr           = 1e-5                # learning rate
        self.batch_size   = 1                   # batch size
        self.epochs       = 30                 # epochs
        self.checkpoints  = './checkpoints'     # checkpoints dir
        self.writer       = SummaryWriter()     # tensorboard writer

        self.__mkdir(self.checkpoints)
```

2. Run train.py In research, we are using 30 epochs(0-29) so the model stops training at epoch 29 as in the figure below.

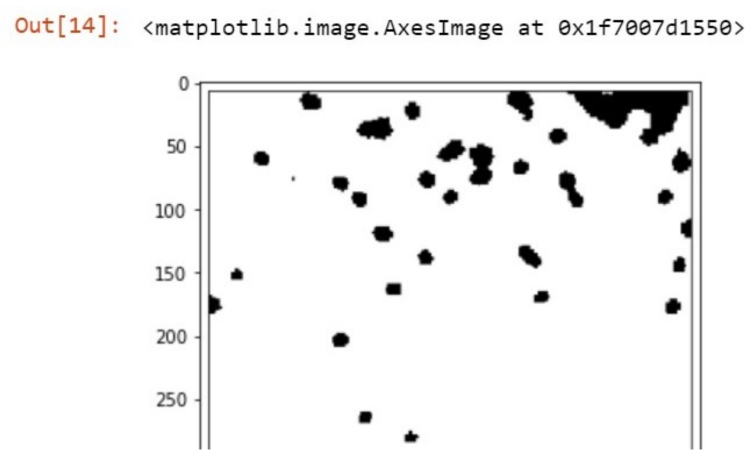


7.5 Predict crowd Density

Run the *01.Get_Density_Map.ipynb* file. To get the density map of an image, set the index to the index of the image inside the text_data folder. In the research, the **test images** are kept inside *CrowdDetection/Data/part_B/test_data/images* folder. The index can be set in cell 8 of the notebook as shown in the figure.



The output is the density map image of the given image as shown in the figure.



8 OpenCV method to convert density map image to grid graph representation

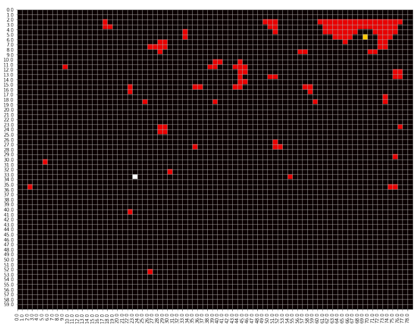
Run *02.Convert_Desnity_Map_To_Grid_Graph.ipynb* file. The image is divided into regions using a loop. A threshold method gets the solid color for the image and an average function calculates the average intensity of the color. If the intensity is zero, the region of the image is an obstacle in the grid. The data is written into a text file. An '$' sign denotes an obstacle and a '-' sign denotes a path in the text file.

jupyter 02.Convert_Desnity_Map_To_Grid_Graph Last Checkpoint: a few seconds ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help

```
In [5]: list_for_astar = []
        f = open("demofile2.txt", "w")
        for y in np.linspace(0,thresh1.shape[0],61):
            if y == 300:
                break
            inner_list = []
            for x in np.linspace(0,thresh1.shape[1],80):
                f = open("demofile2.txt", "a")
                if x==395.0:
                    break
                roi = thresh1[int(y):int(y)+5,int(x):int(x)+5]
                avg_color_per_row = np.average(roi, axis=0)
                avg_color = np.average(avg_color_per_row, axis=0)
        #         print(avg_color)
                if avg_color==0:
        #             print("Black Obstacle")
                    inner_list.append(" $ ")
                    f.write(" $ ")
                else:
        #             print("White Path")
                    inner_list.append(" - ")
                    f.write(" - ")
```

The output of the file is the grid visual in the file.



9 A* path-finding algorithm

Run *03.A-Star Path Finding Algorithm.ipynb* file. The Node class creates an object that contains information about each square in the grid. The children function considers the square in all four direction. It is the next square that can be used to move. A square is a possible path only if it is not an obstacle of '$' in the text file created in the last step.

In [2]:
```
class Node:
    def __init__(self,value,point):
        self.value = value
        self.point = point
        self.parent = None
        self.H = 0
        self.G = 0
    def move_cost(self,other):
        return 0 if self.value == '-' else 1
```

Mention the cost in the report

In [3]:
```
def children(point,grid):
    x,y = point.point
    possible_links=[[x-1, y],[x,y - 1],[x,y + 1],[x+1,y]]
    links = [grid[d[0]][d[1]] for d in possible_links if -1<d[0]<60 if -1<d[1]<79]
    return [link for link in links if link.value != '$']
```

The aStar function finds the path using two sets that contain visited and non-visited nodes for the path. Two cost functions namely, G-cost and H-cost are used. These are o.G and o.H in the code.

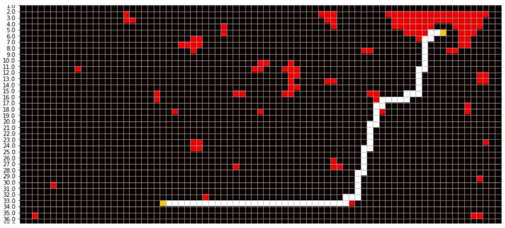
In [5]:
```
def aStar(start, goal, grid):
    #The open and closed sets
    openset = set()
    closedset = set()
    #Current point is the starting point
    current = start
    #Add the starting point to the open set
    openset.add(current)

    #While the open set is not empty
    while openset:
        #Find the item in the open set with the lowest G + H score
        current = min(openset, key=lambda o:o.G + o.H)
        #If it is the item we want, retrace the path and return it
        if current == goal:
            path = []
            while current.parent:
                path.append(current)
                current = current.parent
            path.append(current)
            return path[::-1]
```

Provide the coordinates for the start location and destination location in the grid graph for which you want to find the path.

In [10]:
```
start_pos=input("Enter start position in format x<space>y:") #(17 23) (5 69) (20 45)
```
```
Enter start position in format x<space>y:5 69
```

In [11]:
```
dest_pos=input("Enter destination position in format x<space>y:") #(7 61) (33 23) (0 15)
```
```
Enter destination position in format x<space>y:33 23
```

The output of the notebook is the A* path as shown in the figure.



10 Simulation

Rum *04.Simulation_A_Star.ipynb* file. Set the start and destination location for the simulation.

In [5]:
```
start_pos=input("Enter start position in format x<space>y:") #(17 23) (5 69) (20 45)
```
```
Enter start position in format x<space>y:5 69
```

In [6]:
```
dest_pos=input("Enter destination position in format x<space>y:") ##(7 61) (33 23) (0 15)
```
```
Enter destination position in format x<space>y:33 23
```

We implement an algorithm to find the greedy path. Notice here only G-cost is used which is the Manhattan distance of the square currently considered for the path from the start location.

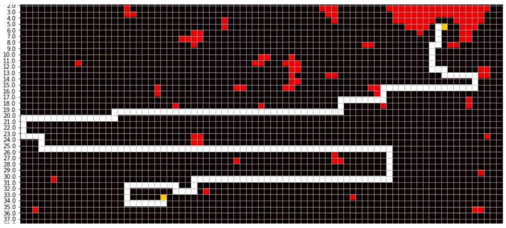
In [17]:
```
def findPath(start, goal, grid):
    #The open and closed sets
    openset = set()
    closedset = set()
    #Current point is the starting point
    current = start
    #Add the starting point to the open set
    openset.add(current)
```

Next, we check if the path returned by the greedy algorithm does not cross any crowd-density areas or obstacles in the grid graph. The red squares are the obstacles in the grid graph.

Check if path reaches any crowd density area

```
In [21]: normalpath = dict()
         i=1
         for node in path:
             x, y = node.point
             normalpath[i] = [x,y]
             i+=1
         normalpath
```

The output of the notebook is the plot for the path returned by the greedy algorithm.



11 Graph Coloring Algorithm

Run *05.GraphColoring.ipynb* file. Take the image of the terminal map and convert it to the adjacency matrix using the list of lists. The row and columns of the matrix are the names of the boarding gates in the node list.


```
In [3]: G = [[ 0, 1, 0, 0, 0, 1],
             [ 1, 0, 1, 0, 0, 0],
             [ 0, 1, 0, 1, 0, 0],
             [ 0, 0, 1, 0, 1, 0],
             [ 0, 0, 0, 1, 0, 1],
             [ 1, 0, 0, 0, 1, 0]]


        # initiate the name of node.
        node = ['307A','307','306','303','302','301']
```

After completion of the execution of the code, the output will be as shown in the figure.

```
Boardig Station -  301    :  Red
Boardig Station -  302    :  Blue
Boardig Station -  303    :  Red
Boardig Station -  306    :  Blue
Boardig Station -  307    :  Red
Boardig Station -  307A   :  Blue
```