National College of Ireland

Configuration Manual

MSc Research Project
Data Analytics

Rutvik Mendjoge
Student ID: 20127855

School of Computing
National College of Ireland

Supervisor: Dr. Catherine Mulwa

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Rutvik Mendjoge
Student ID:	20127855
Programme:	Data Analytics
Year:	2021
Module:	MSc Research Project
Supervisor:	Dr. Catherine Mulwa
Submission Due Date:	16/08/2021
Project Title:	Configuration Manual
Word Count:	928
Page Count:	13

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:	
Date:	16th August 2021

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

Rutvik Mendjoge
20127855

16/08/2021

1 Introduction

The configuration manual is a step-by-step guide for the project 'Synthetic Defocus Generation Using Deep Learning' given in the technical report's creation, installation, implementation, and deployment. The purpose of this report is to support and guide you through each stage of the process so that you may get the required output and results, which are given in a technical report. Multiple technologies, libraries, hardware, and software combinations are used to complete the project.

1.1 Project Overview

The aim of the project is to generate Synthetic Defocus Images using Deep Learning. The algorithms used are DeepLab, Mask-RCNN and Xception. PSNR, SSIM and MSE are the evaluaion metrics used to compare and analyse the performance of the models. DeepLab and Mask-RCNN performed well and gave high PSNR and SSIM values.

1.2 Prerequisites

1.2.1 Hardware Used

- Device Name - Dell 7000 Gaming 7567
- Processor - Intel i7-7th Generation
- RAM - 16 GB
- GPU - NVIDIA GeForce GTX 1050 ti
- ROM - 4 GB
- OS - Windows

1.2.2 Software Used

- Programming Language - Python
- Development Tools - Notepad ++, CMD

2 Python Virtual Environment Creation

2.1 Anaconda Installation

A Python Virtual Environment is created by using Anaconda. CMD and Notepad were used for development. Local GPU of the system is used and installed all the software and packages on the local system itself. The latest version of Anaconda which is used. (2020.11)
[1]

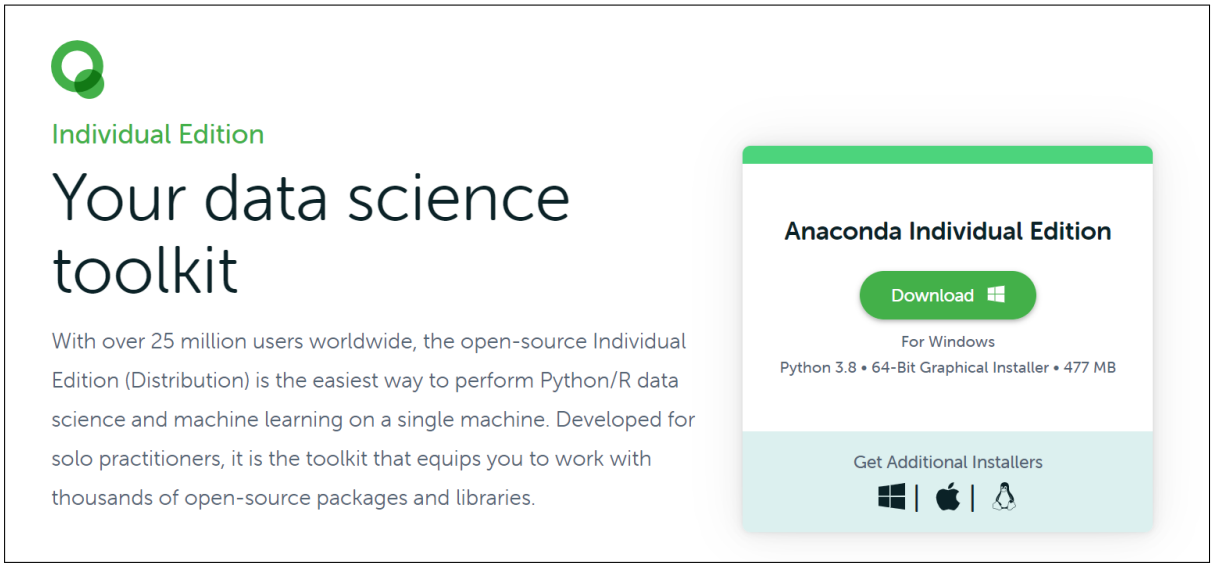

Figure 1: Anaconda Installation

Figure 1 shows the steps needed to download Anaconda. Also, link for the software is included in the Foot Notes.

2.2 Environment Setup

As shown in Figure 2, open the Anaconda Command Prompt from Windows Search function. (RUN AS ADMINISTRATOR)

Activate the anaconda virtual environment (thesis) by using the command shown in the Figure 3.

2.3 Python Libraries, Packages and Frameworks

The main libraries, packages and frameworks are listed in the Table 1.
[2]

[1] https://www.anaconda.com/products/individualwindows
[2] https://www.tensorflow.org/install

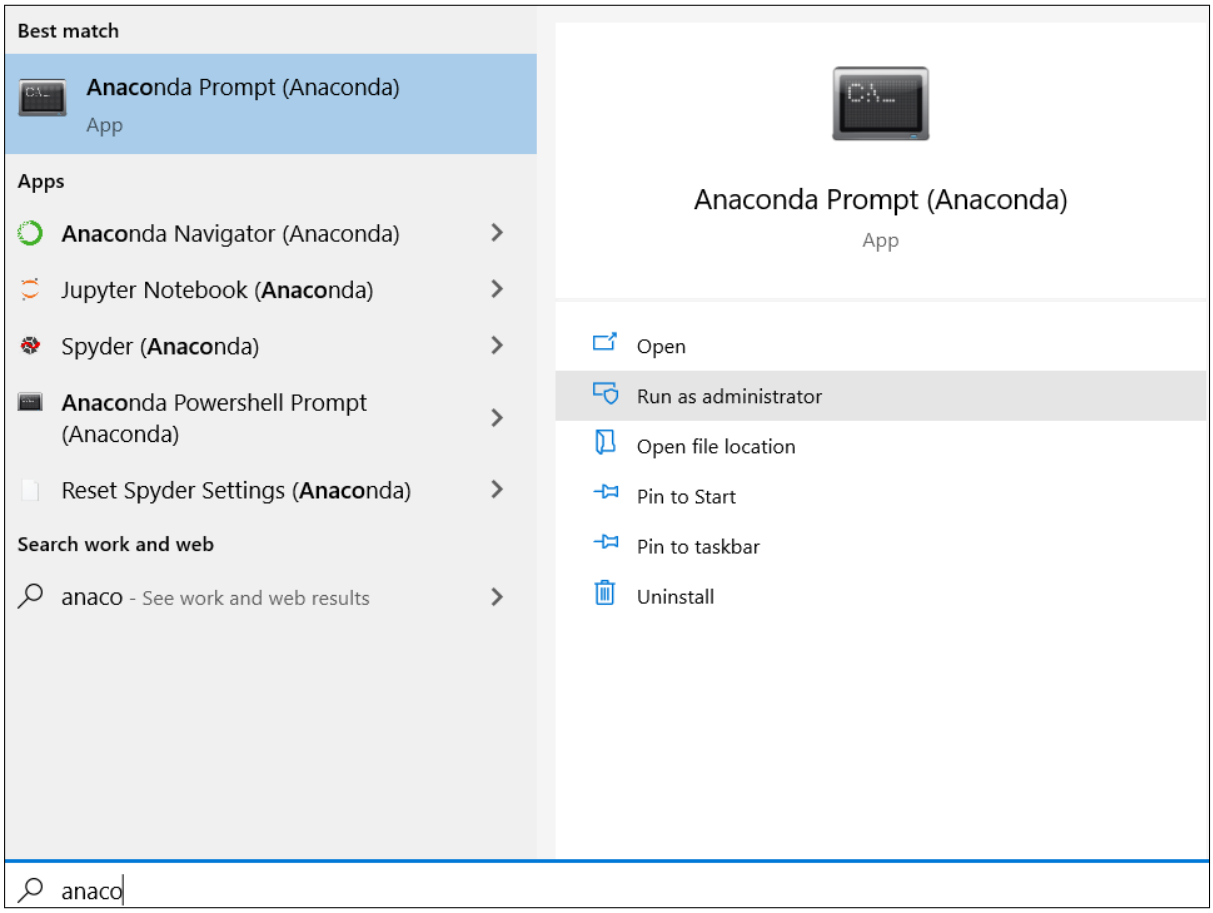

Figure 2: Anaconda Command Prompt

3 Data set Pre-processing

Not much Data Pre-Processing is required because, model specific pre-processing is done in the implementation process. We are using the data set called as EBB which is publicly available on the ETH Zurich website. The Data set is included in the code artifact zip folder.
[3]

4 Implementation

Total 3 Deep Learning models were used DeepLab, Mask-RCNN and Xception.

4.1 DeepLab Model

Figure 4 depicts the necessary libraries and packages which are to be imported. predict-depth.py includes all the parameters which are necessary to pass to the model. Those will be passed as an argument in monodepth-model.py. Hence, importing both of them is necessary.

[3]https://developer.nvidia.com/cuda-11.0-download-archive

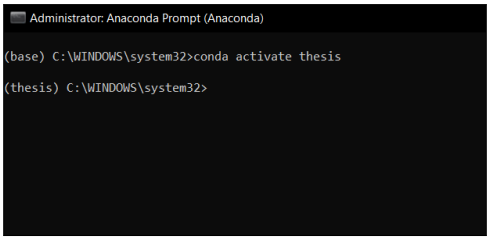

Figure 3: Activate the 'thesis' environment

Table 1: Evaluation Metrics

Libraries	**Version**
TensorFlow	2.4.1
torch	1.8.0
Python	3.8.5
NumPy	1.19.5
CUDA	11.0
CuDNN	8.2.1
CV2	4.5.3

For this project, EBB data set is used. It takes a lot of memory to generate the results. Sometimes it demands more memory to process such huge files. Windows will allocate limited GPU usage for each process. We have to manually increase the GPU memory allocation, otherwise it will throw an not enough memory error. For that, the above code which is mentioned in the Figure 5 is used. That will allow automatic GPU memory allocation according to the requirement. This code solved the issue of GPU memory allocation.

After that for testing the model is any image, the path of the image has to be changed. (image-path)

In the above Figure 6, an image will be resized, converted to RGB format and sent to the predict depth function to calculate the depth of the image.

Figure 7 shows that how segmentation mask is created and applied by using Gaussian Blur function. Hence, the output files will be saved in the output directory.

Figure 8 and Figure 9 depicts the calculation of Evluation Metrics like PSNR, MSE and SSIM.

4.2 Mask-RCNN Model

Figure 10 shows the imports which we need to import. PyTorch will be used as a Deep Learning Framework.

Figure 11 shows the calculation of IoU. Later, an IoU Mask will be calculated and merged together to get the segmentation mask.

Gaussian Blur is applied with OpenCV to blur an image. Gamma value chosen is 0.25. It is shown in Figure 12.

PSNR, MSE and SSIM is calculated using the same code which is used to calculate the scores for DeepLab. Only input image and compressed image path will be changed.

```
import matplotlib.pyplot as plt
import numpy as np
import os
import cv2
import predict_depth
from deeplab_model import DeepLabModel
from math import log10, sqrt
import cv2
import numpy as np
import argparse
from skimage.metrics import structural_similarity as ssim
import tensorflow as tf
```

Figure 4: imports

```
config = tf.compat.v1.ConfigProto()
config.gpu_options.allow_growth = True
sess= tf.compat.v1.Session(config=config)
```

Figure 5: GPU Memory Allocation

```
orig_img = cv2.imread(image_path)
orig_img = cv2.cvtColor(orig_img, cv2.COLOR_BGR2RGB)
H, W, C = orig_img.shape
resize_ratio = 1.0 * MAX_INPUT_SIZE / max(H, W)
H, W = int(resize_ratio * H), int(resize_ratio * W)
orig_img = cv2.resize(orig_img, (W, H), cv2.INTER_AREA)

disp_pp = predict_depth.predict(pretrained_monodepth_path, orig_img)
disp_pp = cv2.resize(disp_pp.squeeze(), (W, H))
disp_pp = disp_pp / disp_pp.max()

model = DeepLabModel(pretrained_deeplabv3_path)
```

Figure 6: Predict the depth map

```
result = orig_img.copy()
mask_viz = np.ones_like(obj_mask, dtype=np.float32)
threshs = [0.8, 0.5, 0.3]
kernels = [5, 9, 11]
fg_masks = [disp_pp < thresh for thresh in threshs]
for i, fg_mask in enumerate(fg_masks):
    kernel_size = kernels[i]
    blurred = cv2.GaussianBlur(orig_img, (kernel_size, kernel_size), 0)
    result[fg_mask] = blurred[fg_mask]
    mask_viz[fg_mask] = 1.0 - ((i + 1) / len(threshs))
result[obj_mask] = orig_img[obj_mask]
merged_mask = np.max([obj_mask.astype(np.float32),
                      mask_viz], axis=0)

output_directory = os.path.dirname(image_path)
output_name = os.path.splitext(os.path.basename(image_path))[0]

plt.imsave(os.path.join(output_directory, "{}_disp.png".format(
    output_name)), disp_pp, cmap='gray')
plt.imsave(os.path.join(output_directory, "{}_fg.png".format(
    output_name)), mask_viz, cmap='gray')
plt.imsave(os.path.join(output_directory, "{}_segmap.png".format(
    output_name)), obj_mask, cmap='gray')
plt.imsave(os.path.join(output_directory, "{}_mask.png".format(
    output_name)), merged_mask, cmap='gray')
plt.imsave(os.path.join(output_directory, "{}_blurred.png".format(output_name)), result)
plt.imsave(os.path.join(output_directory, "{}_resized.png".format(
    output_name)), orig_img)
```

Figure 7: Segmentation Mask

4.3 Xception Model

For the third and last model, TensorFlow, Gaussian Blur, CV2, Numpy are used to generate Synthetic defocus images.
Almost similar packages are used here and imported them. All the imports necessary to run Exception model are shown in Figure 13.

A Frozen inference graph is necessary to run this model and to load the necessary weights to generate the synthetic defocus images. This file shown in Figure 14 should be saved in the same folder as of code.

Segmentation Map with respect to the localized objects is generated as shown in Figure 15.

After calculating the segmentation map, the input image will be classified in 0 and 255 based on the classes detected. Using Numpy Unique values are calculated and the output image (Segmentation Map) will be resized to the original image shape.
After resizing the image, the unique values won't be limited to 0, 255. Here, Otsu's Binarization comes into the picture. It puts a filter on the range of the values. The output of the Otsu's Binarization is an image which will have only 2 unique values. Again, after applying the Gaussian Filter, the final Portrait image is stored in the same output directory. Which is shown in Figure 16 and Figure 17.

```
#PSNR

def PSNR(original, compressed):
 mse = np.mean((original - compressed) ** 2)
 if(mse == 0):
  return 100
 max_pixel = 255.0
 psnr = 20 * log10(max_pixel / sqrt(mse))
 return psnr


# Import images
original = cv2.imread(image_path)
compressed = cv2.imread(os.path.join(output_directory, "{}_blurred.png".format(output_name)))

# Check for same size and ratio and report accordingly
ho, wo, _ = original.shape
hc, wc, _ = compressed.shape
ratio_orig = ho/wo
ratio_comp = hc/wc
dim = (wc, hc)

if round(ratio_orig, 2) != round(ratio_comp, 2):
 print("\nImages not of the same dimension. Check input.")
 exit()

# Resize original if the compressed image is smaller
elif ho > hc and wo > wc:
 print("\nResizing original image for analysis...")
 original = cv2.resize(original, dim)

elif ho < hc and wo < wc:
 print("\nCompressed image has a larger dimension than the original. Check input.")
 exit()

value = PSNR(original, compressed)
print("\nPeak Signal-to-Noise Ratio (PSNR) value is", value, "dB")
```

Figure 8: PSNR

```
#SSIM
def mse(imageA, imageB):
 # the 'Mean Squared Error' between the two images is the sum of the squared difference between the two images
 mse_error = np.sum((imageA.astype("float") - imageB.astype("float")) ** 2)
 mse_error /= float(imageA.shape[0] * imageA.shape[1])
 # return the MSE. The lower the error, the more "similar" the two images are.
 return mse_error

def compare(imageA, imageB):
 # Calculate the MSE and SSIM
 m = mse(imageA, imageB)
 s = ssim(imageA, imageB)

 # Return the SSIM. The higher the value, the more "similar" the two images are.
 return s

# Import images
image1 = cv2.imread(image_path)
image2 = cv2.imread(os.path.join(output_directory, "{}_blurred.png".format(output_name)))

# Convert the images to grayscale
gray1 = cv2.cvtColor(image1, cv2.COLOR_BGR2GRAY)
gray2 = cv2.cvtColor(image2, cv2.COLOR_BGR2GRAY)

# Check for same size and ratio and report accordingly
ho, wo, _ = image1.shape
hc, wc, _ = image2.shape
ratio_orig = ho/wo
ratio_comp = hc/wc
dim = (wc, hc)

if round(ratio_orig, 2) != round(ratio_comp, 2):
 print("\nImages not of the same dimension. Check input.")
 exit()

# Resize first image if the second image is smaller
elif ho > hc and wo > wc:
 #print("\nResizing original image for analysis...")
 gray1 = cv2.resize(gray1, dim)

elif ho < hc and wo < wc:
 print("\nCompressed image has a larger dimension than the original. Check input.")
 exit()

if round(ratio_orig, 2) == round(ratio_comp, 2):
 mse_value = mse(gray1, gray2)
 ssim_value = compare(gray1, gray2)
 print("MSE:", mse_value)
 print("SSIM:", ssim_value)
```

Figure 9: MSE and SSIM

```
import os
import numpy as np
import torch
import torch.nn as nn
import torch.utils.data
from PIL import Image
import cv2
import torchvision
import argparse
from typing import List
import keras
import tensorflow as tf
from math import log10, sqrt
from skimage.metrics import structural_similarity as ssim
import matplotlib.pyplot as plt
```

Figure 10: Mask-RCNN Imports

```
def IoU(prediction, id1, id2):
    """Compute IoU of two bounding boxes whose indices are id1 and id2."""
    # First extract the bounding boxes for each mask.
    bbox1 = prediction['boxes'][id1]
    bbox2 = prediction['boxes'][id2]
    device = bbox1.device

    # Compute U.
    xmin = min(bbox1[0], bbox2[0])
    ymin = min(bbox1[1], bbox2[1])
    xmax = max(bbox1[2], bbox2[2])
    ymax = max(bbox1[3], bbox2[3])
    u = (xmax - xmin) * (ymax - ymin)
    union_bbox = torch.FloatTensor([xmin, ymin, xmax, ymax]).to(device)

    # Compute I.
    xmin = max(bbox1[0], bbox2[0])
    ymin = max(bbox1[1], bbox2[1])
    xmax = min(bbox1[2], bbox2[2])
    ymax = min(bbox1[3], bbox2[3])
    i = (xmax - xmin) * (ymax - ymin)
    inter_bbox = torch.FloatTensor([xmin, ymin, xmax, ymax]).to(device)

    return i/u, inter_bbox, union_bbox
```

Figure 11: Calculate IoU

```python
def apply_blur(image, prediction, thres, degree=1/30, gamma=0.25):
    """Synthesize image with bokeh effect.

    @param degree (int) [0, 1]
           The larger it is, the more blurred the background
    """
    h, w = image.shape[0:2]
    ksize = int(min(h, w) * degree)
    if ksize % 2 == 0:
        ksize += 1

    mask = get_mask(prediction, thres=thres).detach().cpu().numpy()
    mask[mask > thres] = 1.0
    mask[mask < thres] = 0.0
    mask = cv2.erode(mask, np.ones((ksize//4, ksize//4), dtype=np.uint8))
    closing = cv2.morphologyEx(
        mask, cv2.MORPH_CLOSE, np.ones((ksize, ksize), dtype=np.uint8))
    mask = cv2.GaussianBlur(mask, (ksize, ksize), 0)
    mask_dilated = cv2.dilate(mask, np.ones((ksize, ksize),
                                            dtype=np.uint8)
                              )
    mask_dilated = cv2.GaussianBlur(mask_dilated, (ksize, ksize), 0)
    mask_dilated = np.expand_dims(mask_dilated, 2)

    # Bokeh effect on the whole image
    kernel = disc_shaped_kernel(ksize)

    # Get image gamma-corrected and apply disc-shape blur to get an
    # image with bokeh effect
    gamma_correction = ((image / 255.0) ** (1 / gamma)) * 255
    bokeh = cv2.filter2D(gamma_correction.astype(np.uint8), 3, kernel)
    bokeh = (((bokeh / 255.0) ** gamma) * 255).astype(np.uint8)
    # Keep only pixels with high value from bokeh image
    blurred = cv2.GaussianBlur(image, (ksize, ksize), 0)
    bokeh = bokeh * (1.0 - mask_dilated)
    bokeh = cv2.max(bokeh.astype(np.uint8), blurred)

    # Blend
    mask = np.expand_dims(mask, 2)
    image = image * mask + bokeh * (1 - mask)
    return image.astype(np.uint8)
```

Figure 12: Apply Blur

```
# -*- coding: utf-8 -*-

import os
from io import BytesIO
import tarfile
import tempfile
import cv2
from six.moves import urllib
from copy import deepcopy
from PIL import Image
from math import log10, sqrt
from skimage.metrics import structural_similarity as ssim
from copy import deepcopy
from matplotlib import gridspec
from matplotlib import pyplot as plt
import numpy as np
from IPython.display import Image as IMG
import tensorflow as tf
```

Figure 13: Imports for Xception

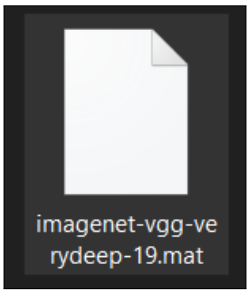

Figure 14: Frozen Inference Graph

```
def vis_segmentation(image, seg_map):
  """Visualizes input image, segmentation map and overlay view."""
  plt.figure(figsize=(15, 5))
  grid_spec = gridspec.GridSpec(1, 4, width_ratios=[6, 6, 6, 1])

  plt.subplot(grid_spec[0])
  plt.imshow(image)
  plt.axis('off')
  plt.title('input image')

  plt.subplot(grid_spec[1])
  seg_image = label_to_color_image(seg_map).astype(np.uint8)
  plt.imshow(seg_image)
  plt.axis('off')
  plt.title('segmentation map')

  plt.subplot(grid_spec[2])
  plt.imshow(image)
  plt.imshow(seg_image, alpha=0.7)
  plt.axis('off')
  plt.title('segmentation overlay')

  unique_labels = np.unique(seg_map)
  ax = plt.subplot(grid_spec[3])
  plt.imshow(
      FULL_COLOR_MAP[unique_labels].astype(np.uint8), interpolation='nearest')
  ax.yaxis.tick_right()
  plt.yticks(range(len(unique_labels)), LABEL_NAMES[unique_labels])
  plt.xticks([], [])
  ax.tick_params(width=0.0)
  plt.grid('off')
  plt.show()


LABEL_NAMES = np.asarray([
    'background', 'aeroplane', 'bicycle', 'bird', 'boat', 'bottle', 'bus',
    'car', 'cat', 'chair', 'cow', 'diningtable', 'dog', 'horse', 'motorbike',
    'person', 'pottedplant', 'sheep', 'sofa', 'train', 'tv'
])
```

Figure 15: Segmentation Map and the Label Names

```
person_not_person_mapping = deepcopy(numpy_image)
person_not_person_mapping[seg_map != 15] = 0
person_not_person_mapping[seg_map == 15] = 255

plt.imshow(person_not_person_mapping)
np.unique(person_not_person_mapping)
orig_imginal = Image.open(IMAGE_NAME)
orig_imginal = np.array(orig_imginal)
orig_imginal.shape
```

Figure 16: Calculating Unique Values

```
mapping_resized = cv2.resize(person_not_person_mapping,
                             (orig_imginal.shape[1],
                              orig_imginal.shape[0]),
                             Image.ANTIALIAS)

plt.imshow(mapping_resized)

np.unique(mapping_resized)

gray = cv2.cvtColor(mapping_resized, cv2.COLOR_BGR2GRAY)
blurred = cv2.GaussianBlur(gray,(15,15),0)
ret3,thresholded_img = cv2.threshold(blurred,0,255,cv2.THRESH_BINARY+cv2.THRESH_OTSU)
plt.imshow(thresholded_img, cmap="gray")
thresholded_img.ndim
mapping = cv2.cvtColor(thresholded_img, cv2.COLOR_GRAY2RGB)
plt.imshow(mapping)
np.unique(mapping)
blurred_original_image = cv2.GaussianBlur(orig_imginal,
                                          (251,251),
                                          0)

plt.imshow(blurred_original_image)
layered_image = np.where(mapping != (0,0,0),
                         orig_imginal,
                         blurred_original_image)

plt.imshow(layered_image)
im_rgb = cv2.cvtColor(layered_image, cv2.COLOR_BGR2RGB)
cv2.imwrite("Potrait_Image.jpg", im_rgb)

IMG("Potrait_Image.jpg")
```

Figure 17: Otsu's Binarization and Final Output