National College of Ireland

Configuration Manual

MSc Research Project
Data Analytics

Shubham Rajabhau Maske
Student ID: x19232551

School of Computing
National College of Ireland

Supervisor: Dr. Hicham Rifai

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Shubham Rajabhau Maske
Student ID:	x19232551
Programme:	Data Analytics
Year:	2021
Module:	MSc Research Project
Supervisor:	Dr. Hicham Rifai
Submission Due Date:	16/08/2021
Project Title:	Configuration Manual
Word Count:	757
Page Count:	8

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:	Shubham Rajabhau Maske
Date:	15th 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

Shubham Rajabhau Maske
x19232551

1 Introduction

The configuration manual document outlines the steps that were taken throughout the project's coding phase. Hardware and software configurations are specified to enable future replication of this study. This section details the programming and deployment stages necessary for efficient code execution, as well as the procedures necessary to run the code.

2 System Configuration

2.1 Hardware Configuration

Figure 2 shows the hardware specifications of the system on which the research study is being conducted.

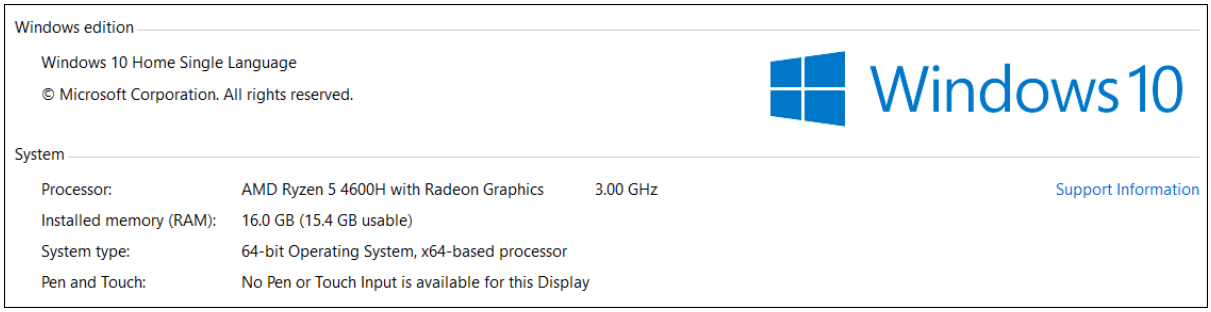

Figure 1: System Configuration

2.2 Software Configuration

The software and its specifications are presented in this section.

2.2.1 Google Colab

The research is conducted on Google's cloud infrastructure called as Google Colab. For executing the model, older tensorflow version is required. Below code is used to make the environment use older version of tensorflow.

The dataset is locate don google drive and to mount this drive into colab below code is used.

```
%tensorflow_version 1.x
import tensorflow as tf
```

Figure 2: Tensorflow Version

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

Figure 3: Google Drive Mount

When this code is executed it provides a link where we have to authorise our google drive login. Once we authorise it, an authorization code is provided which we have to copy and pastes it in the textbox in colab.

Keras version 2.2.5 is needed to execute the Mask R-CNN model. This is achieved by executing following code in google colab.

```
!pip install keras==2.2.5
```

Figure 4: Keras Version

Google colab pro version is used in this research as it provides more RAM, disk space and high processing GPUs. To set the environment to run in GPU mode below option is selected.

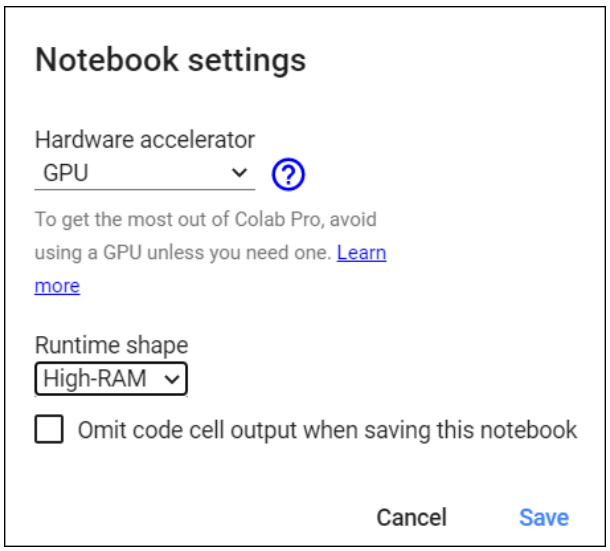

Figure 5: Setting GPU environment for colab notebook

2.2.2 Texstudio

Texstudio is used for project report documentation as shown in below figure.

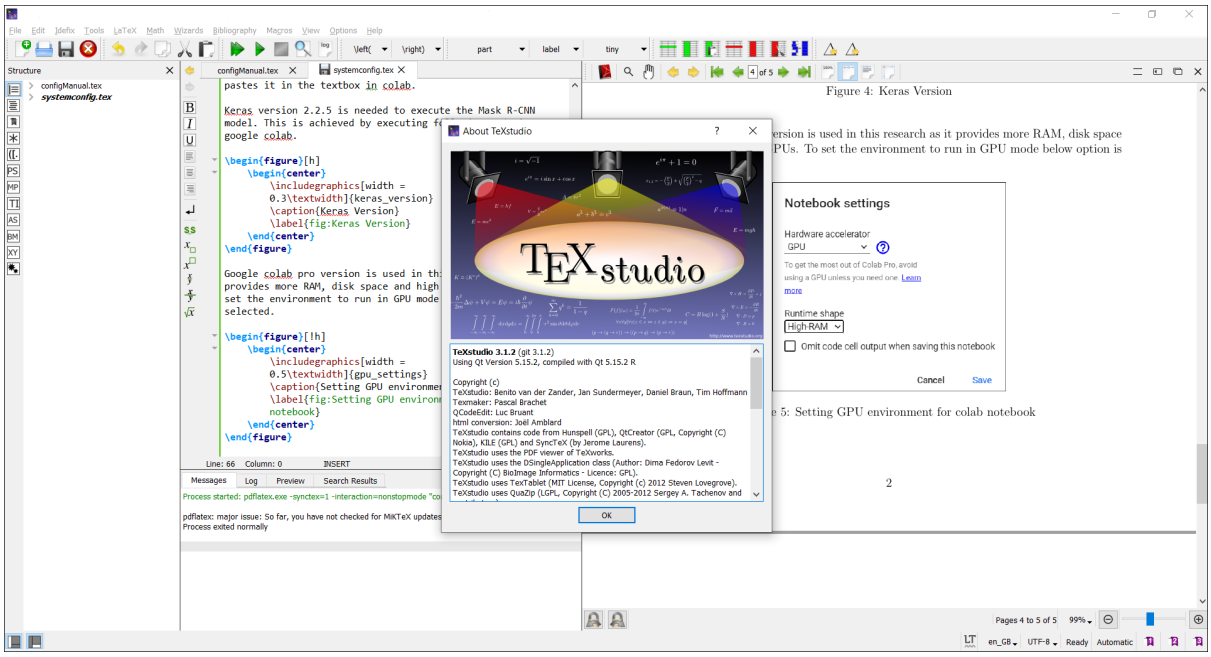

Figure 6: Texstudio

3 Data Preparation

3.1 Load Dataset

The dataset used in this research is downloaded from the github [1] which was created by the authors in (Zheng et al.; 2021).

Below function is used to load the dataset. This function is used to define classes and for defining the images. To define classes, *add_class* function from mrcnn library is used where *class_id* and *class_name* are defined. To define objects, *add_image* from mrcnn library is used where source of the data, class id and path to the images are defined.

```
def load_dataset(self, dataset_dir, path):
  self.add_class("dataset", 1, "microUAV")
  images_dir = dataset_dir + path
  annot_dir = dataset_dir + 'Annotations/'

  for filename in listdir(images_dir):
    image_id = filename[:-4]                #extract filename as image_id
    img_file_path = images_dir + filename
    ann_file_path = annot_dir + image_id + '.xml'
    self.add_image('dataset', image_id=image_id, path=img_file_path, annotation=ann_file_path)
```

Figure 7: Function to load dataset

[1] https://github.com/Jake-WU/Det-Fly

3.2 Extract Bounding Boxes

Each images in the dataset have its respective XML file which contains bounding box information. Below function is used ot extract the bounding boxes from the XML files. This function returns bounding box details along with width and height of the image.

```
def extract_boxes(self, filename):
  tree = ElementTree.parse(filename)
  root = tree.getroot()
  boxes = list()
  for box in root.findall('.//bndbox'):
    xmin = int(box.find('xmin').text)
    ymin = int(box.find('ymin').text)
    xmax = int(box.find('xmax').text)
    ymax = int(box.find('ymax').text)
    coor = [xmin, ymin, xmax, ymax]
    boxes.append(coor)

  width = int(root.find('.//size/width').text)
  height = int(root.find('.//size/height').text)

  return boxes, width, height
```

Figure 8: Function to extract bounding box

3.3 Adding mask

With the help of bounding box of the image, mask can be applied within that box. Below function will be used to this process.

```
def load_mask(self, image_id):
  info = self.image_info[image_id]
  path = info['annotation']

  boxes, w, h = self.extract_boxes(path)

  masks = zeros([h, w, len(boxes)], dtype='uint8')

  class_ids = list()
  for i in range(len(boxes)):
    box = boxes[i]
    row_s, row_e = box[1], box[3]
    col_s, col_e = box[0], box[2]
    masks[row_s:row_e, col_s:col_e, i] = 1
    class_ids.append(self.class_names.index('microUAV'))

  return masks, asarray(class_ids, dtype='int32')
```

Figure 9: Function to add mask to image

4 Model Implementation

The Mask R-CNN library is first installed in the colab notebook, so that its functions and classes can be utilized. The library is available on github [2]. This library is cloned using below command.

```
!git clone https://github.com/matterport/Mask_RCNN.git
```

Figure 10: Installing MRCNN library

To utilize the functions from this library, below python file need to be executed from this library.

```
!python setup.py install
```

Figure 11: Setting up mrcnn library

Below libraries will be required to implement several functionalities.

```
from os import listdir
from xml.etree import ElementTree
from numpy import zeros
from numpy import asarray
from mrcnn.utils import Dataset
from matplotlib import pyplot
from mrcnn.visualize import display_instances
from mrcnn.utils import extract_bboxes
from mrcnn.config import Config
from mrcnn.model import MaskRCNN
from mrcnn.utils import compute_ap
from mrcnn.model import load_image_gt
from mrcnn.model import mold_image
from mrcnn import visualize
from numpy import expand_dims
from numpy import mean
import pandas as pd
from matplotlib.patches import Rectangle
import mrcnn.model as modellib
import random
```

Figure 12: Importing libraries

4.1 Configuration for training model

Below code is used to define the configuration class for training the model.

[2] https://github.com/matterport/Mask_RCNN

```
# define configuration for training the model
class microUAVConfig(Config):

  # Naming the configuration
  NAME="microUAV_cfg"

  # Define number of classes (background + microUAV)
  NUM_CLASSES = 1 + 1

  # Number of training steps per epoch; steps_per_epoch = Number of training images / Batch Size (9696/2 = 4848)
  STEPS_PER_EPOCH = 4848
```

Figure 13: Configuration class for training model

The training and test set objects are created as well as the configuration object is also created as shown below.

```
train_set = microUavDataset()
train_set.load_dataset(root_dir,train_dir)
train_set.prepare()
print('Train: %d' % len(train_set.image_ids))

test_set = microUavDataset()
test_set.load_dataset(root_dir,valid_dir)
test_set.prepare()
print('Validation: %d' % len(test_set.image_ids))

# prepare config
config = microUAVConfig()
config.display()
```

Figure 14: Object Creation

4.2 Loading MS COCO weights

Below line of code is used to load the weights of MS COCO dataset that are generated using Mask R-CNN.

```
model.load_weights('/content/drive/MyDrive/microUAV_dataset/weight_file/mask_rcnn_coco.h5', by_name=True, exclude=["mrcnn_class_logits", "mrcnn_bbox_fc",  "mrcnn_bbox", "mrcnn_mask"])
```

Figure 15: Load MS COCO weights

4.3 Training model

The model is trained using 6 epochs and is implemented as follows:

```
model.train(train_set, test_set, learning_rate=config.LEARNING_RATE, epochs=6, layers='heads')
```

Figure 16: Training model

4.4 Evaluation

To plot the loss for each epoch, below code is used.

```
pyplot.figure(figsize=(17,5))

pyplot.subplot(131)
pyplot.plot(epochs, history["loss"], label="Train loss")
pyplot.plot(epochs, history["val_loss"], label="Valid loss")
pyplot.legend()
pyplot.subplot(132)
pyplot.plot(epochs, history["mrcnn_class_loss"], label="Train class ce")
pyplot.plot(epochs, history["val_mrcnn_class_loss"], label="Valid class ce")
pyplot.legend()
pyplot.subplot(133)
pyplot.plot(epochs, history["mrcnn_bbox_loss"], label="Train box loss")
pyplot.plot(epochs, history["val_mrcnn_bbox_loss"], label="Valid box loss")
pyplot.legend()

pyplot.show()
```

Figure 17: Code to plot loss

To evaluate the model, we need to execute it in inference mode. A separate configuration class is created for this and is implemented as below.

```
# define the prediction configuration
class PredictionConfig(Config):
    # define the name of the configuration
    NAME = "microUAV_pred_cfg"
    # number of classes (background + microUAV)
    NUM_CLASSES = 1 + 1
    # simplify GPU config
    GPU_COUNT = 1
    IMAGES_PER_GPU = 1
```

Figure 18: Configuration class for inference mode

The model for inference mode is defined as follow.

```
# define the inference model
model = MaskRCNN(mode='inference', model_dir='/content/drive/MyDrive/microUAV_dataset/predicted_model', config=cfg)
```

Figure 19: Defining model for inference mode

The weights of the trained model is loaded using below code as shown in figure 20.

```
# load model weights
model.load_weights('/content/drive/MyDrive/microUAV_dataset/microuav_cfg20210804T0858/mask_rcnn_microuav_cfg_0005.h5', by_name=True)
```

Figure 20: Loading wights from trained model

Once the mode is defined in inference mode, below function in figure 21 is used to calculate mean average precision (mAP). This function in turns calls the mrcnn library function *compute_ap* which returns the list of average precision.

```
# calculate the mAP of the model
def evaluate_model(dataset, model, cfg):
  APs = list()
  for image_id in dataset.image_ids:
    # load image, bounding boxes and masks for the image id
    image, image_meta, gt_class_id, gt_bbox, gt_mask = load_image_gt(dataset, cfg, image_id, use_mini_mask=False)
    # convert pixel values
    scaled_image = mold_image(image, cfg)
    # convert image into one sample
    sample = expand_dims(scaled_image, 0)
    # make prediction
    yhat = model.detect(sample, verbose=0)
    # extract results for first sample
    r = yhat[0]
    # calculate statistics, including AP
    AP, _, _, _ = compute_ap(gt_bbox, gt_class_id, gt_mask, r["rois"], r["class_ids"], r["scores"], r['masks'])
    # store
    APs.append(AP)
  # calculate the mean AP across all images
  mAP = mean(APs)
  return mAP
```

Figure 21: Function to calculate mAP

To detect micro-UAV on random images using the trained model, below code is implemented. Here the ground truth and the prediction are presented using below code.

```
# Detect micro-UAV on random image
for i in range(2):
  # pick random image
  image_id = random.choice(test_set.image_ids)
  original_image, image_meta, gt_class_id, gt_bbox, gt_mask = modellib.load_image_gt(test_set, cfg, image_id, use_mini_mask=False)
  print("\n Ground Truth:")
  visualize.display_instances(original_image, gt_bbox, gt_mask, gt_class_id, test_set.class_names, figsize=(10,10) )

  results = model.detect([original_image], verbose=1)
  r = results[0]
  print("\n Prediction:")
  visualize.display_instances(original_image, r['rois'], r['masks'], r['class_ids'],test_set.class_names, r['scores'], figsize=(10,10))
```

Figure 22: Code for testing the model on random image

References

Zheng, Y., Chen, Z., Lv, D., Li, Z., Lan, Z. and Zhao, S. (2021). Air-to-air visual detection of micro-uavs: An experimental evaluation of deep learning, *IEEE Robotics and Automation Letters* **6**(2): 1020–1027.