National College of Ireland

Configuration Manual

MSc Research Project
Data Analytics

Shajo Varghese
Student ID: 22115943

School of Computing
National College of Ireland

Supervisor: Bharat Agarwal

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Shajo Varghese Eramplackal
Student ID:	22115943
Programme:	Data Analytics **Year:** 2023
Module:	MSc Research Project
Lecturer:	Bharat Agarwal
Submission Due Date:	14/12/2023
Project Title:	Configuration Manual
Word Count:	..... **Page Count:** .....

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: Shajo Varghese

Date: 14/12/2023

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

Shajo Varghese
Student ID: 22115943

1 Introduction

This is the configuration manual describing guidelines to implement the research project ' Comparative Analysis of YOLO Variants for Object Detection in Thermal Images". The hardware and software requirements are specified in this report.

2 Hardware Configuration

The hardware requirements for the project are given below.

- Processor: Intel(R) Core(TM) i5-10300H CPU @ 2.50GHz 2.50 GHz
- RAM: 8GB - 16GB RAM is typically preferred
- Storage: 128/ 500 GB SSD

3 Software Configuration

- Google Colab

4 Environment Setup

Colab Setup

Open Google Colab:. https://colab.research.google.com/
Create a new notebook or open an existing one.

Setup of Code

1. PreprocessingFile.ipynb-Code for pre-processing
2. yolov5_model_training.ipynb-Code for implementing YOLOv5
3. yolov7_model_training.ipynb- Code for implementing YOLOv7
4. yolov8_model_training.ipynb- Code for implementing YOLOv7
5. data.yaml-Configuration file for YOLOv5
6. yolov7_data.yaml- Configuration file for YOLOv7
7. yolov8_data.yaml- Configuration file for YOLOv8

Dataset

A labelled dataset was collected from universe-roboflow dataset with bounding box annotations for the objects, car and person. The dataset had 10037 images for training and 2237 for validation. Some pre-processing steps were already applied like resizing (stretched to 640x480), grayscale and Filter Null which requires at least 66 percent of the images to be annotated.

Dataset link: https://universe.roboflow.com/thermal-imaging-0hwfw/flir-data-set/dataset/13

The dataset was uploaded to Google drive and was he project directory was mounted from Google drive for every model i.e YOLOv5,YOLOv7 and YOLOv8.

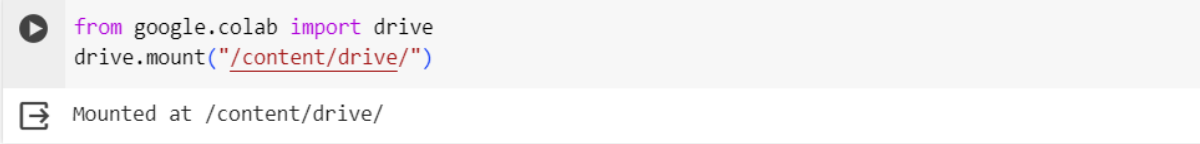

Figure 1

The ".yaml" contains details for the model to train like path to training images and validation images and no of classes.

The yaml file for code files are as follows:

1. yolov5_model_training.ipynb-----→ data.yaml
2. yolov7_model_training.ipynb-----→ yolov7_data.yaml
3. yolov8_model_training.ipynb-----→ yolov8_data.yaml

The data in ".yaml" is as follows for YOLOv5 model :

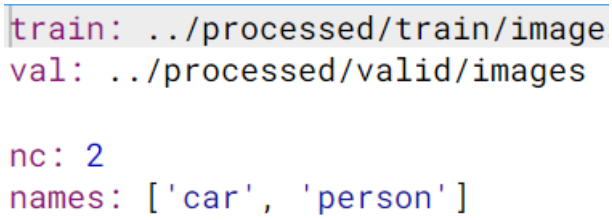

Figure 2

These are all the requisite files necessary for the successful execution of this research project.

So, the final data structure should be :

1. PreprocessingFile.ipynb
2. yolov5_model_training.ipynb
3. yolov7_model_training.ipynb
4. yolov8_model_training.ipynb
5. data.yaml
6. yolov7_data.yaml
7. yolov8_data.yaml
8. Dataset

5 Implementation

1.Pre -Processing

Libraries Imported :

```
import os
import cv2
import numpy as np
from tqdm import tqdm
```

```
def FolderStructure(processed_folder):
    if not os.path.exists(processed_folder):
        os.makedirs(processed_folder)

    train_folder = os.path.join(processed_folder, 'train')
    valid_folder = os.path.join(processed_folder, 'valid')

    if not os.path.exists(train_folder):
        os.makedirs(train_folder)
    if not os.path.exists(valid_folder):
        os.makedirs(valid_folder)

    train_images_folder = os.path.join(train_folder, 'images')
    train_labels_folder = os.path.join(train_folder, 'labels')

    valid_images_folder = os.path.join(valid_folder, 'images')
    valid_labels_folder = os.path.join(valid_folder, 'labels')

    if not os.path.exists(train_images_folder):
        os.makedirs(train_images_folder)
    if not os.path.exists(train_labels_folder):
        os.makedirs(train_labels_folder)

    if not os.path.exists(valid_images_folder):
        os.makedirs(valid_images_folder)
    if not os.path.exists(valid_labels_folder):
        os.makedirs(valid_labels_folder)
```

```
FolderStructure('processed')
```

Figure 3

The code in the Figure 3 is the code to import important libraries and create a new "**processed**" folder with the structure as input to store the processed images.

```
def Image_Processer(file_path, outpath):
    image = DataLoading(file_path)
    image = Denoising(image)
    image = ImageSmoothening(image)
    image = ImageSharpening(image)
    image = ImageDetailEnhancer(image)
    image = ImageTextureEnhancer(image)
    ImageSaver(image, outpath)
    return None
```

Figure 4

All the functions were defined inside the custom user defined functions and the user defined function Image_Processor() Function is used to apply Pre-processing to all the images.

```
def DataLoading(file_path):
    image = cv2.imread(file_path)
    return image
```

```
def Denoising(source):
    denoised_image = cv2.fastNlMeansDenoisingColored(source,None,1,1,1,1)
    return denoised_image
```

```
def ImageSmoothening(source):
    smoothened_image = cv2.GaussianBlur(source, (3, 3), 0)
    return smoothened_image
```

```
def ImageSharpening(source):
    sharpening_kernel = np.array([[-1, -1, -1],
                                  [-1,  9, -1],
                                  [-1, -1, -1]])

    sharpened_image = cv2.filter2D(source, -1, sharpening_kernel)
    return sharpened_image
```

```
def ImageDetailEnhancer(source):
    detail_enhanced_image = cv2.detailEnhance(source, sigma_s=10, sigma_r=0.15)
    return detail_enhanced_image
```

Figure 5

Figure 5 shows a small glimpse of the preprocessing functions applied to thermal Images.

```
%%time

for folder_name in os.listdir(MAIN_FOLDER):
    folder_path = os.path.join(MAIN_FOLDER, folder_name)
    sub_folder_name = os.listdir(folder_path)
    if 'images' in sub_folder_name:
        sub_folder_path = os.path.join(folder_path, 'images')
        all_images = os.listdir(sub_folder_path)
        for image_name in tqdm(all_images):
            image_path = os.path.join(sub_folder_path, image_name)
            output_path = os.path.join('processed', folder_name, 'images', image_name)
            Image_Processer(image_path, output_path)
    else:
        print("There is no 'images' folder")
```

Figure 6

This is the final step in the Pre-processing Step.The MAIN_Folder is the input folder.The images are fetched from the input folder, applied pre-processing and then stored to preprocessed folder.The tqdm library will help us to keep track of progress and will also give us the ETA for completing the given piece of code.The labels should be uploaded manually since they does not require any preprocessing.

2. YOLOv5 Implementation.

Initially, the project data structure is imported into the Google Colab Environment.

```
!git clone https://github.com/ultralytics/yolov5
```

Figure 7

The Yolov5 model is cloned form the YOLOv5 repository from the Ultralytics GitHub repository.

Importing torch

```
[ ]  import torch
```

torch.hub.load() function is used to load the YOLOv5 model from the Github repository.

```
[ ]  model = torch.hub.load('ultralytics/yolov5','yolov5s')

     Using cache found in /root/.cache/torch/hub/ultralytics_yolov5_master
     YOLOv5 🚀 2023-12-12 Python-3.10.12 torch-2.1.0+cu118 CUDA:0 (Tesla T4, 15102MiB)

     Downloading https://github.com/ultralytics/yolov5/releases/download/v7.0/yolov5s.pt to yolov5s.pt...
     100%|██████████| 14.1M/14.1M [00:00<00:00, 120MB/s]

     Fusing layers...
     YOLOv5s summary: 213 layers, 7225885 parameters, 0 gradients, 16.4 GFLOPs
     Adding AutoShape...
```

Figure 8

The torch library is imported and used to load the YOLOv5 model from the Github.The Figure 8 also contains version details about the Python and Torch versions used by the YOLOv5.

```
!python yolov5/train.py --img 640 --batch 32 --epochs 5 --data data.yaml --weights 'yolov5s.pt'
```

Figure 9

The final step consisting of training the YOLOv5 model with specifying the Image size, batch , epochs and configuration file using pretrained weights downloaded in earlier step.

3.YOLOv7 Implementation.

Initially, the project data structure is imported into the Google Colab Environment.
Since YOLOv7 was being too computationally expensive, a new “processed1” dataset was created for successful training of the YOLOv7 model.It could be carried out by transferring some 5050 images from preprocessed folder to “processed1”.If the file name needs to be changed, then it should also be updated in the “yolov7_data.yaml”

```
!git clone https://github.com/WongKinYiu/yolov7.git
```

Figure 10

We start the YOLOv7 implementation by cloning the YOLOv7 repository from WongKinYiu's GitHub

```
!wget https://github.com/WongKinYiu/yolov7/releases/download/v0.1/yolov7.pt
```

Figure 10

We are using the wget function to download the YOLOv7 model weights from the release.

```
!python yolov7/train.py --workers 2 --device 0 --batch-size 16 --data yolov7_data.yaml --img 640 640 --weights 'yolov7.pt' --epochs 20
```

Figure 11

In the last step , we train our YOLOv7 models by specifying input field like no of workers device, batch size, configuration file, image size , weights and epochs.

Version Details:
YOLOR 🚀 v0.1-128-ga207844 torch 2.1.0+cu118 CUDA:0 (Tesla T4, 15101.8125MB)

4. YOLOv8 Implementation.

Initially, the project data structure is imported into the Google Colab Environment.

We start the YOLOv8 implementation step by importing the YOLO class from the Ultralytics library.

```
from ultralytics import YOLO
```

Figure 12

```
model = YOLO("yolov8s.pt")
```

Figure 13

In figure 13, we create an instance of YOLOv8 using pre-trained weights.

```
results = model.train(data="yolov8_data.yaml", epochs=5, imgsz=640)
```

Figure 13

Training the YOLOv8 model with providing the configuration file, epochs and image size as input.

Version Details:
Ultralytics YOLOv8.0.227 🚀 Python-3.10.12 torch-2.1.0+cu118 CUDA:0 (Tesla T4, 15102MiB)

6 Results

The YOLOv5 results will be saved in the **yolov5/runs/train/exp.**
The YOLOv7 and YOLOv8 saves results by creating a “runs” folder.
It is necessary to change the name of the runs folder after implementing either YOLOv8 or YOLOv7. Else, the folder results will be over-rided.