National College of Ireland

Configuration Manual

MSc Research Project
Data Analytics

Vinutha Nagaraju
Student ID: 23110686

School of Computing
National College of Ireland

Supervisor: Hamilton V. Niculescu

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Vinutha Nagaraju
Student ID:	23110686
Programme:	Data Analytics
Year:	2024
Module:	MSc Research Project
Supervisor:	Hamilton V. Niculescu
Submission Due Date:	12/08/2024
Project Title:	Configuration Manual
Word Count:	1349
Page Count:	10

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:	Vinutha Nagaraju
Date:	12th August 2024

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

Vinutha Nagaraju
23110686

1 Introduction

This setup guide includes all the necessary details, such as the equipment used, software and hardware specifications, key code snippets, and reproducibility criteria. It offers clear, step-by-step instructions for setting up the essential software and hardware and implementation details needed to build the complete system. These instructions will make it easier to replicate the study.

The following section are structured as

- Environmental Setup
- Required Libraries
- Steps to implement the models

2 Environmental Setup

This section provides the overview of software and hardware specification.

2.1 Software Specifications

The table 1 below outlines the software environment used for the development and execution of the project. This includes the online integrated development environment (IDE), the programming language, its version, and additional tools utilized.

Table 1: Softwares

Software	**Configuration**
Online IDE	Google Colab Pro
Coding Language	Python
Coding Language Version	Python 3.10.12
Additional Tools Used	RoboFlow

2.2 Hardware Specifications

The project used Google Colab Pro to avoid disconnections using the free google colab.It provides NVIDIA Tesla T4 GPU with 16 GB of GPU memory, which is ideal for deep learning and machine learning tasks. Additionally, it provides High-RAM of up to 32

GB. The system also includes approximately 100 GB of temporary local storage, useful for storing datasets and intermediate files during computation. These hardware specifications ensure efficient performance for computationally demanding tasks in a cloud-based environment like Google Colab Pro.Table 2 shows the hardware specifcations.

Table 2: Hardware Specifications

Hardware Component	**Specification**
GPU	- NVIDIA Tesla T4 - 16 GB GPU memory
RAM (Random Access Memory)	High-RAM: Up to 32 GB
Disk Space	Local Storage: 100 GB (temporary)

Google Drive is used in conjunction with Google Colab to provide persistent storage and facilitate easy access to datasets, code, and outputs.

3 Required Libraries

This project relies on essential libraries for data processing, visualization, and deep learning. Libraries like os, random, json, and math handle system operations, randomness, JSON data, and mathematical functions. For visualizations and numerical computations, we use matplotlib, seaborn, and numpy. Image processing is managed with cv2 (OpenCV) and PIL. PyTorch and torchvision is used to build, optimize, and deploy neural networks, including pre-trained models like VGG16 and Inception V3.Figure 1 shows the imported libraries.

Figure 1: Libraries used

Figure 2 shows the installtion of imagehash library.

Figure 2: Installing imagehash library

4 Implementation Details

4.1 Mounting Google Drive

The process of making the Google Drive available within the Google Colab environment is done using the below step shown in figure 3. In google drive the datasets are present.

```
[ ]  #connecting google drive to google colab
     from google.colab import drive
     drive.mount('/content/gdrive')

     Mounted at /content/gdrive
```

Figure 3: Mounting Google Drive

4.2 Annotation of Data Using RoboFlow

We need to create a project and have to define the classes which we need to detect in an image.I have used 2 classes bird and drone.We can make that as public dataset or can have the licence as well as shown in the figure 4.

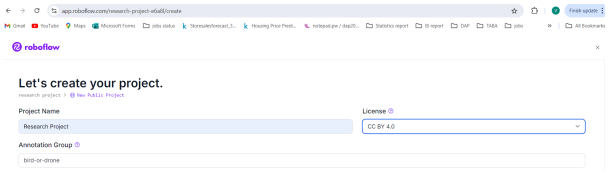

Figure 4: Creating Project in RoboFlow

After that we need to upload the data , then there are 3 options provided to annotate the data as shown in the figure 5.I have used the Auto Label, but after the labelling process is complete we need to ensure that the images are correctly labeled.

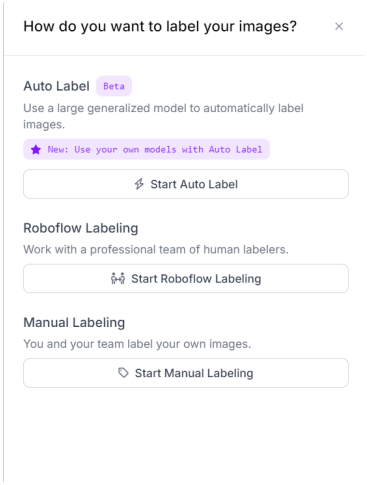

Figure 5: Annotating options

After annotation, the image looks like in the figure 6.

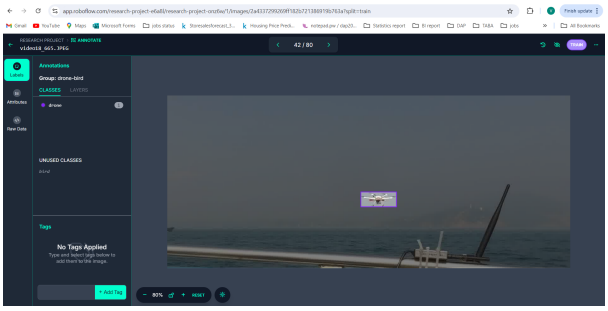

Figure 6: Image with bounding box

After verifying that the predicted classes are correct by the auto label, we can generate the dataset.There are options to split the data into train, tets and validation by mentioning the ratio. I have done that separately in a code to ensure that images to be shuffled randomly.No pre processing steps were applied with roboflow as shown in figure 7

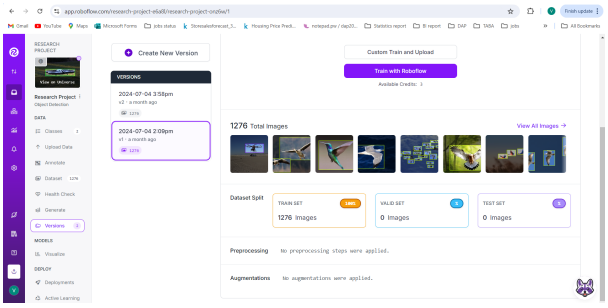

Figure 7: Prepared dataset

We can also maintain different versions based on the changes.This prepared dataset is exported using PyTorch YOLOv7 specification because we will be using the YOLOv7 in our research. The downloaded dataset consisting of images and label files in .txt format is uploaded to drive for further processing.

We have split the dataset into train, validation and test as needed by yolov7 where each folder has images and corresponding labels as shown in figure 8.

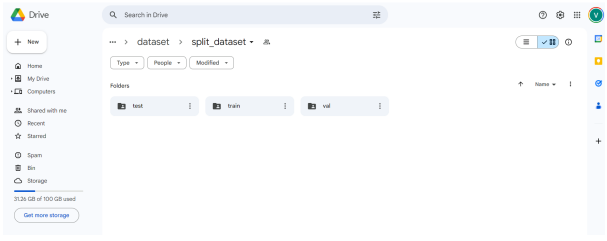

Figure 8: Dataset split

But for Classification models like ResNet18, vGG16 and InceptionV3 we have split the data where each folder of it contains different folders for each class i.e., drone and bird.

4.3 YOLOv7 Implementation

Figure 9 shows the cloning of the YOLOv7 from oficial repository on github to google colab and setting the directory to YOLOv7 to access further files in it.

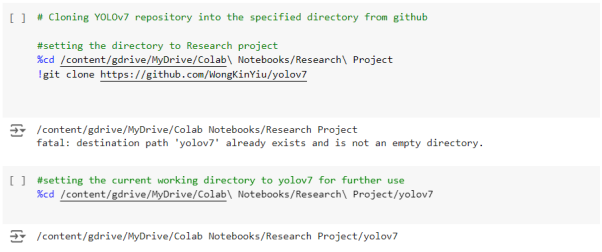

Figure 9: Cloning YOLOv7

The requirements needed by YOLOv7 should be installed as shown in the figure 10

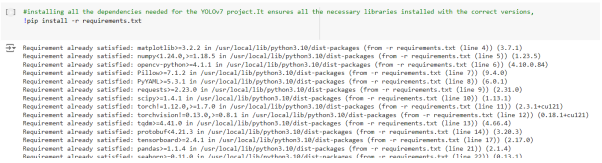

Figure 10: YOLOv7 requirements

In order to train the YOLOv7 to our requirements , we need to make changes in some of the files that is present in YOLOv7 folder.

We have to define the number of classes in the customyolov7.yaml file which has the architecture defined for YOLOv7 as shown in figure 11.

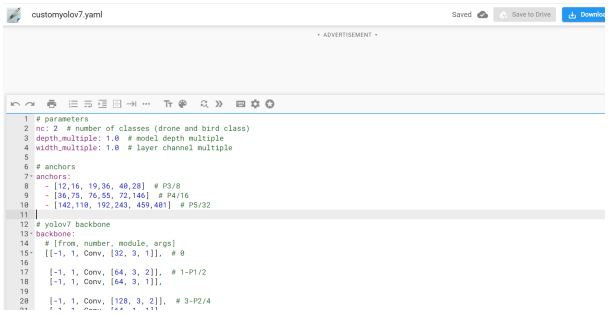

Figure 11: Customyolov7 file

In order to change any hyperparameter like learning rate or apply data augmentation , changes were made in hyp.scratch.custom.yaml file accordingly as shown in figure 12.

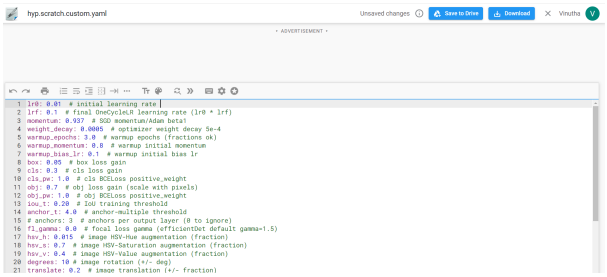

Figure 12: Hyperparameter file

In customdata.yaml we have defined the paths for train,test and validation folder and number of class, name of the classes needs to be defined as shown in figure 13 .

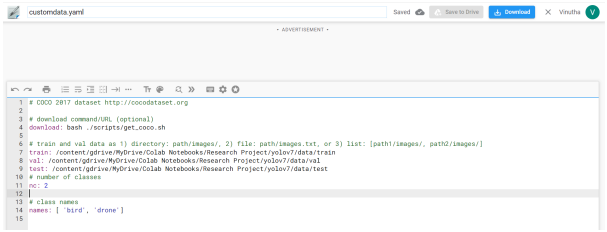

Figure 13: Customdata file

After setting up all this, the model can be trained using the changed files on the dataset of drone or bird as shown in figure 14.Also the model will be saved in saved models of YOLOv7.

Figure 14: Training YOLOv7

After training the model can be tested using test.py that is already present in YOLOv7 and using the best weights from the saved model .The path where the results of the test run is stored is also displayed in console as shown in the figure 15. It is stored in runs/test/exp4.

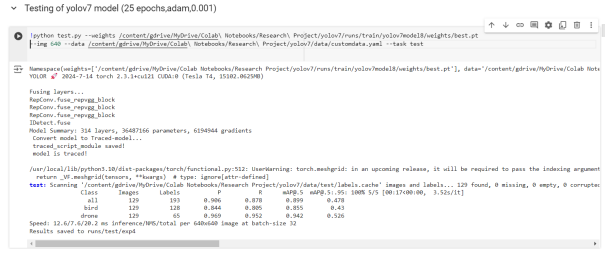

Figure 15: Testing YOLOv7

The detection on new image is performed using detect.py which is also already present by providing the image path on which it needs to perform detection as shown in figure 16.Everything related to dataset or any image needs to be stored under data of YOLOv7 for easy access.

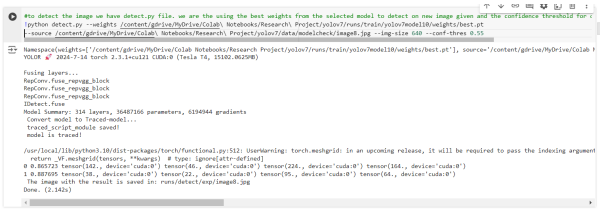

Figure 16: Detection using YOLOv7

4.4 Classification Algorithms Implementation

The Dataset used to train the YOLOv7 is also used for training the classification models. But only the images are considered, we didn't require the labels.

Figure 17 shows the training of ResNet18 with pre trained weights imported already using libraries.

```
# Training the model resnet18 with pretrained weights
model = torchvision.models.resnet18(pretrained=True)
num_ftrs = model.fc.in_features
model.fc = nn.Linear(num_ftrs, 2)  #considering our 2 classes drone and bird
```

Figure 17: ResNet18 Training

After training the model is saved using the specified directory as shown in figure 18.

```
# directory and file path for saving the model
model_path = '/content/gdrive/MyDrive/Colab Notebooks/Research Project/Saved Models/'
model_state_filename = 'model1_resnet_state.pth'
os.makedirs(model_path, exist_ok=True)
```

Figure 18: Saving of Model

The training of VGG16 is as shown in figure 19 with pre trained weights.

```
#training the vgg16 model with pretrained weights
model = torchvision.models.vgg16(weights=VGG16_Weights.IMAGENET1K_V1)
num_ftrs = model.classifier[6].in_features
model.classifier[6] = nn.Linear(num_ftrs, 2) #considering 2 classes
```

Figure 19: Training VGG16

The model is saved same as shown in the figure 18 under different name.

```
# Traing the model with pre trained weights
model = torchvision.models.inception_v3(weights=Inception_V3_Weights.IMAGENET1K_V1)
num_ftrs = model.fc.in_features
model.fc = nn.Linear(num_ftrs, 2)  # Assume 2 classes
```

Figure 20: Training InceptionV3

The model is trained and saved the same way.Figure 20 shows the training code.

The training and validation losses, accuracies are also saved using json file as shown in figure 21 which is needed later to plot the learning curve.

```
# Saving the metrics
metrics = {
    'train_losses': train_losses,
    'val_losses': val_losses,
    'train_accuracies': train_accuracies,
    'val_accuracies': val_accuracies
}

with open(model_path + 'metrics.json', 'w') as f:
    json.dump(metrics, f)
```

Figure 21: Saving learning metrics

After training of the models, they are loaded by defining the classifier again and providing the path for saved model and setting loaded model to evaluation mode for any further use as shown in figure 22.

```
# Defining the custom classifier
class CustomClassifier(nn.Module):
    def __init__(self):
        super(CustomClassifier, self).__init__()
        self.flatten = nn.Flatten()
        self.dense1 = nn.Linear(25088, 512)
        self.dropout = nn.Dropout(0.5)
        self.batchnorm = nn.BatchNorm1d(512)
        self.dense2 = nn.Linear(512, 2)

    def forward(self, x):
        x = self.flatten(x)
        x = self.dense1(x)
        x = self.dropout(x)
        x = self.batchnorm(x)
        x = self.dense2(x)
        return x

# Loading the pre-trained VGG16 model
base_model = torchvision.models.vgg16(weights=VGG16_Weights.IMAGENET1K_V1)
base_model.classifier = CustomClassifier()

# Moving model to GPU if available
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
base_model = base_model.to(device)

# Loading the saved model
model_path = '/content/gdrive/MyDrive/Colab Notebooks/Research Project/Saved Models/'
model_state_filename = 'model7_vgg16_state.pth'
base_model.load_state_dict(torch.load(model_path + model_state_filename))

# Defining  the loss function
criterion = nn.CrossEntropyLoss()

# Setting model to evaluation mode
base_model.eval()
```

Figure 22: Loading the Saved Model

Testing of selected model is done as in shown figure 23

```
#calculating for test loss and test accuracy
test_loss = 0.0
correct_test = 0
total_test = 0

# Evaluating the model
base_model.eval()
with torch.no_grad():
    for inputs, labels in test_loader:
        inputs, labels = inputs.to(device), labels.to(device)
        outputs = base_model(inputs)
        loss = criterion(outputs, labels)
        test_loss += loss.item() * inputs.size(0)
        _, predicted = torch.max(outputs, 1)
        total_test += labels.size(0)
        correct_test += (predicted == labels).sum().item()

test_loss /= len(test_loader.dataset)
test_accuracy = correct_test / total_test

print(f'Test Loss: {test_loss:.4f}, Test Acc: {test_accuracy:.2%}')
```

```
Test Loss: 0.1731, Test Acc: 98.45%
```

Figure 23: Testing of Model

4.5 YOLOv7 with VGG16

Figure 24 shows the code where the detections by YOLOv7 are extracted .

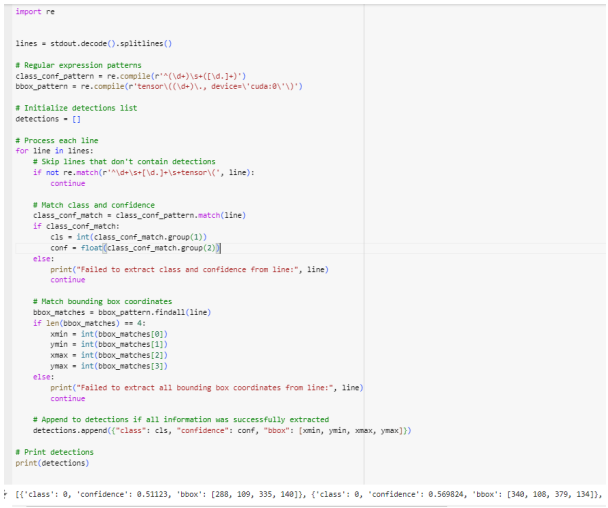

Figure 24: Extracting the Detections by YOLOv7

Figure 25 represents how the cropping of Region of interest(ROI) is performed.

Figure 25: Cropping of ROI

Figure 26 represents the transformations applied for cropped parts.

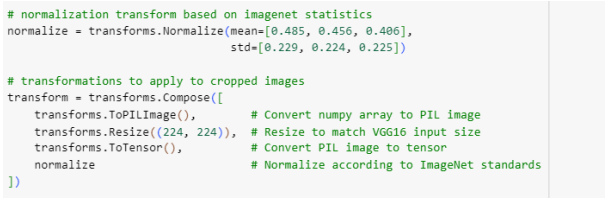

Figure 26: Transformations

Contrast enhancement is made before sending it the cropped images to the classification model as shown in figure 27.

```python
# Apply contrast enhancement (CLAHE)
lab = cv2.cvtColor(image, cv2.COLOR_RGB2LAB)
l, a, b = cv2.split(lab)
clahe = cv2.createCLAHE(clipLimit=3.0, tileGridSize=(8, 8))
l = clahe.apply(l)
lab = cv2.merge((l, a, b))
image = cv2.cvtColor(lab, cv2.COLOR_LAB2RGB)
```

Figure 27: Contrast Enhancement

Figure 28 shows the code for predicting the class on original image.

```python
# Process each cropped image and draw predictions on the original image
original_image = cv2.imread(image_path)
for detection in detections:
    # Extract the bounding box
    bbox = detection['bbox']
    xmin, ymin, xmax, ymax = bbox

    # Crop the image
    cropped_image = original_image[ymin:ymax, xmin:xmax]
    cropped_image = Image.fromarray(cropped_image)

    # Preprocess the image
    input_tensor = preprocess_image(cropped_image)

    # Move input tensor to device (CPU or GPU)
    input_tensor = input_tensor.to(device)

    # Forward pass through VGG16 model
    with torch.no_grad():
        output = base_model(input_tensor)

    # softmax activation to get class probabilities
    probabilities = torch.nn.functional.softmax(output, dim=1)

    # Get predicted class
    _, predicted_class = torch.max(probabilities, 1)
    predicted_class_name = class_names[predicted_class.item()]
    predicted_class_color = class_colors[predicted_class.item()]

    # text parameters
    font = cv2.FONT_HERSHEY_SIMPLEX
    font_scale = 0.5
    font_thickness = 1
    text_color = (255, 255, 255)  # White text color for better visibility

    # Draw the bounding box and label on the original image
    cv2.rectangle(original_image, (xmin, ymin), (xmax, ymax), predicted_class_color, 1)  # Setting thickness for a thinner box
    cv2.putText(original_image, predicted_class_name, (xmin, ymin - 10), font, font_scale, text_color, font_thickness)
```

Figure 28: Classification by VGG16

The evaluation of proposed pipeline is done on test dataset using the above methods to determine if it performs better than the original yolov7 in classifying drone V/s bird.

The figure 29 shows the overview of YOLOv7 folder. We have the train.py, test.py and detect.py files which are used to train, test and for interpretation on new data. In runs, we will have all the results and in data folder we have the dataset needed and images for interpretation.

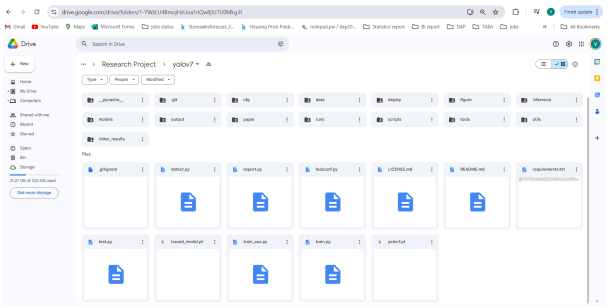

Figure 29: YOLOV7