Configuration Manual

MSc Research Project

Data Analytics

Ankit Kumar

Student ID: x23123061

School of Computing

National College of Ireland

Supervisor: Syed Muhammad Raza Abidi

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Ankit Kumar		
Student ID:	X23123061		
Programme:	MSc in Data Analytics	**Year:**	23-24
Module:	MSc Research Project		
Supervisor:	Syed Muhammad Raza Abidi		
Submission Due Date:	12/08/2024		
Project Title:	Configuration Manual - Object Detection for Visually Impaired Individuals in Different Weather Conditions		
Word Count:	1175, 12 Pages		

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: Ankit Kumar

Date: 12/08/2024

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):	

Configurational Manual

1 Technology Setup

For the technology part, Google Colab was used as it provides easy access to cloud data. As the research project is compute-power-intensive, Graphical Processing Unit (GPU) was required to be set up.

In the free version of the Google Colab, the TPU is not stable as it gets disconnected often and the code currently being processed, refreshes, losing all the runtime. For a stable connection with the GPU, Google Colab Pro +, worth €52/month was purchased. It gives access to a total of 500 Compute power per month to the users. The benefits of the Colab Pro + environment is explained in table 1 Below.

Table 1: Google Colab Pro+ benefits

Benefit	**Description**
Price	€52 Per Month
Compute Unit	Total 500 Compute units per month
GPU	Powerful and Premium GPU with priority access is provided
Execution	Background Execution, for up to 24 Hrs is provided, even after the browser is closed

Python 3 environment is provided by Google Colab and in the Python 3, the TPU was selected as shown in Figure 1 below. The compute units are exhausted, therefore the powerful GPUs are locked until next billing cycle, but the highlighted red box gives a snapshot of the current plan. For the current research, the fastest, NVIDIA A100 GPU was used.

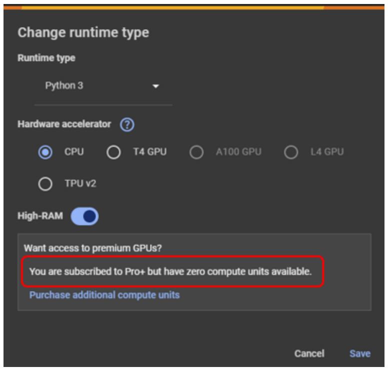

Figure 1: Available GPU

2 The Code

In this section, we will discuss the code and how was the research conducted on Google Colab.

2.1 Installing Ultralytics and Mounting G-Drive

The first step is to install the Ultralytics and mounting the Google drive in the Python Environment. Ultralytics is an important library for this project, as it has the current version of pre-trained YOLO, YOLOv8, which will be used for transfer learning. With this, Opencv was also installed, which is a popular library for Computer vision tasks such as image and video processing.

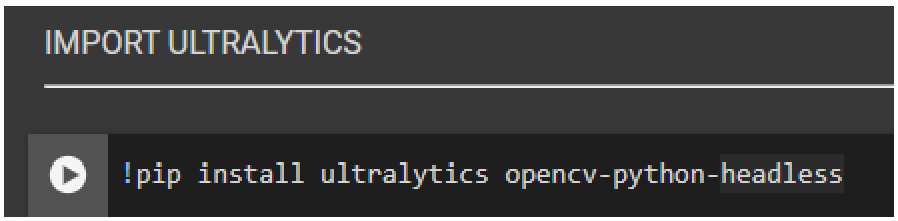

Figure 2: Installing Ultralytics and OpenCV

The next step is mounting the google drive, which has the dataset to be used for training and inference. Code shown in figure 3 depicts the code for the same.

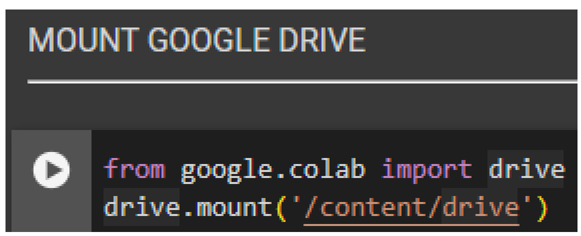

Figure 3: Code for mounting G-Drive

After mounting the google drive, on the left panel of the Google drive, we will be able to see the drive folder associated with the account being used with the Colab (Refer Figure 4).

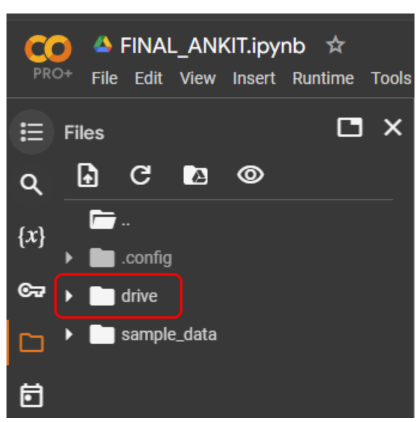

Figure 4: Confirming that the drive is mounted

2.2 Data Pre-Processing

After mounting the drive, defining the path to the actual folder, where the dataset is present was set. Post that, the total images were counted to set the scene for the pre-processing. Figure 5 shows the code, that was written in order to count total number of images.

COUNT TOTAL IMAGES OF VOC 2012 DATASET

```
[ ] import glob
    # count total images before processing

    total_images = len(glob.glob('/content/drive/MyDrive/VOC2012/VOCdevkit/VOC2012/JPEGImages/*.jpg'))
    print(total_images)

1964
```

Figure 5: Counting initial count of images in dataset before processing

The next step is to process the images for different weather conditions and then add them in different folders for each weather condition for training on each weather condition. The screenshot of the code below in Figure 6, shows how the images were processed, and artificial editing was done to mimic the images for a specific weather using Open CV.

```
import os
import cv2
import glob
import random
from tqdm import tqdm
import numpy as np
import xml.etree.ElementTree as ET

# Define paths
VOC_ROOT = '/content/drive/MyDrive/VOC2012/VOCdevkit/VOC2012/'
IMAGES_DIR = VOC_ROOT + 'JPEGImages/'
ANNOTATIONS_DIR = VOC_ROOT + 'Annotations/'
OUTPUT_DIR = '/content/drive/MyDrive/VOC2012/YOLO/'

# Create necessary directories
weather_conditions = ['sunny', 'cloudy', 'foggy', 'rainy', 'snowy']
# weather_conditions = ['snowy2']

for condition in weather_conditions:
    os.makedirs(f"{OUTPUT_DIR}/{condition}/images/train", exist_ok=True)
    os.makedirs(f"{OUTPUT_DIR}/{condition}/images/val", exist_ok=True)
    os.makedirs(f"{OUTPUT_DIR}/{condition}/labels/train", exist_ok=True)
    os.makedirs(f"{OUTPUT_DIR}/{condition}/labels/val", exist_ok=True)

# Image augmentation functions
def add_sunny(image):
    return cv2.convertScaleAbs(image, alpha=1.2, beta=30)

def add_cloudy(image):
    return cv2.convertScaleAbs(image, alpha=0.8, beta=-20)

def add_foggy(image):
    fog_layer = np.full_like(image, 255, dtype=np.uint8)
    alpha = 0.5
    return cv2.addWeighted(image, alpha, fog_layer, 1 - alpha, 0)

def add_rainy(image):
    rain_layer = np.zeros_like(image, dtype=np.uint8)
    num_drops = 1000
    for _ in range(num_drops):
        x = np.random.randint(0, image.shape[1])
        y = np.random.randint(0, image.shape[0])
        rain_layer = cv2.line(rain_layer, (x, y), (x, y + 10), (255, 255, 255), 1)
    rain_layer = cv2.blur(rain_layer, (3, 3))
    return cv2.addWeighted(image, 0.8, rain_layer, 0.2, 0)

def add_snowy(image):
    snow_layer = np.zeros_like(image, dtype=np.uint8)
    num_flakes = 1500  # Increased number of flakes for a stronger effect
    for _ in range(num_flakes):
        x = np.random.randint(0, image.shape[1])
        y = np.random.randint(0, image.shape[0])
        flake_size = np.random.randint(1, 4)  # Varied size of snowflakes
        snow_layer[y:y + flake_size, x:x + flake_size] = 255
    snow_layer = cv2.GaussianBlur(snow_layer, (7, 7), 0)  # Increased blur for a softer effect
    return cv2.addWeighted(image, 0.7, snow_layer, 0.3, 0)
```

```
def convert_voc_to_yolo_bbox(size, box):
    dw = 1. / size[0]
    dh = 1. / size[1]
    x = (box[0] + box[1]) / 2.0 - 1
    y = (box[2] + box[3]) / 2.0 - 1
    w = box[1] - box[0]
    h = box[3] - box[2]
    x = x * dw
    w = w * dw
    y = y * dh
    h = h * dh
    return (x, y, w, h)

def process_and_save_annotations(image_files, output_image_dir, output_label_dir, weather_effect_func):
    for image_file in tqdm(image_files, desc="Processing annotations"):
        annotation_file = ANNOTATIONS_DIR + os.path.basename(image_file).replace('.jpg', '.xml')
        tree = ET.parse(annotation_file)
        root = tree.getroot()
        size = root.find('size')
        w = int(size.find('width').text)
        h = int(size.find('height').text)

        yolo_annotations = []

        for obj in root.iter('object'):
            difficult = obj.find('difficult').text
            if int(difficult) == 1:
                continue
            cls = obj.find('name').text
            if cls not in classes:
                continue
            cls_id = classes.index(cls)
            xmlbox = obj.find('bndbox')
            b = (float(xmlbox.find('xmin').text), float(xmlbox.find('xmax').text),
                 float(xmlbox.find('ymin').text), float(xmlbox.find('ymax').text))
            bb = convert_voc_to_yolo_bbox((w, h), b)
            yolo_annotations.append(f"{cls_id} " + " ".join([str(a) for a in bb]) + '\n')

        if yolo_annotations:
            image = cv2.imread(image_file)
            image = weather_effect_func(image)

            cv2.imwrite(output_image_dir + os.path.basename(image_file), image)
            with open(output_label_dir + os.path.basename(image_file).replace('.jpg', '.txt'), 'w') as f:
                f.writelines(yolo_annotations)

# Define class names as in the VOC dataset
classes = ["aeroplane", "bicycle", "bird", "boat", "bottle",
           "bus", "car", "cat", "chair", "cow",
           "diningtable", "dog", "horse", "motorbike", "person",
           "pottedplant", "sheep", "sofa", "train", "tvmonitor"]
```

```
# List all image files
image_files = glob.glob(IMAGES_DIR + '*.jpg')

# Split data into train and validation sets
random.shuffle(image_files)
split_index = int(len(image_files) * 0.8)
train_files = image_files[:split_index]
val_files = image_files[split_index:]

# Process and save images for each weather condition
weather_funcs = {
    # 'sunny': add_sunny,
    # 'cloudy': add_cloudy,
    # 'foggy': add_foggy,
    # 'rainy': add_rainy,
    'snowy': add_snowy
}

for condition, func in weather_funcs.items():
    print(f"Processing {condition} dataset...")
    process_and_save_annotations(train_files, f"{OUTPUT_DIR}/{condition}/
    images/train/", f"{OUTPUT_DIR}/{condition}/labels/train/", func)
    process_and_save_annotations(val_files, f"{OUTPUT_DIR}/{condition}/
    images/val/", f"{OUTPUT_DIR}/{condition}/labels/val/", func)

print("VOC2012 dataset converted to YOLO format for all weather conditions successfully.")
```

Figure 6: Code for pre-processing images weather-wise (split in three columns of images)

After processing the images, it is important to take the count of the final images as we have successfully increased the training dataset for images 5 fold. Figure 7 below shows the code for counting the images

COUNT IMAGES AFTER PRE-PROCESSING

```python
def count_images_in_directory(directory):
    image_files = glob.glob(os.path.join(directory, '*.jpg'))  # Adjust the file extension if necessary
    return len(image_files)
```

```python
total_train_images = 0
total_val_images = 0

for condition in conditions:
    train_images_dir = os.path.join(output_dir, condition, 'images/train')
    val_images_dir = os.path.join(output_dir, condition, 'images/val')

    train_images_count = count_images_in_directory(train_images_dir)
    val_images_count = count_images_in_directory(val_images_dir)

    total_train_images += train_images_count
    total_val_images += val_images_count

    print(f"{condition.capitalize()} - Training images: {train_images_count}, Validation images: {val_images_count}")

total_images = total_train_images + total_val_images

print(f"Total training images: {total_train_images}")
print(f"Total validation images: {total_val_images}")
print(f"Total images in dataset: {total_images}")
```

```
Sunny - Training images: 1570, Validation images: 393
Cloudy - Training images: 1570, Validation images: 393
Foggy - Training images: 1570, Validation images: 393
Rainy - Training images: 1570, Validation images: 393
Snowy - Training images: 1570, Validation images: 393
Total training images: 7850
Total validation images: 1965
Total images in dataset: 9815
```

Figure 7: Count of images post processing

Before training the model, a YAML file must be created which will store the configuration of the model. As we will be training our model on al the weather conditions, a YAML file for each weather must be created and be stored in the weather folder. Figure 8 below shows the creation on YAML file in Python and once done, Figure 9 shows the creation of those YAML file in the google Drive.

```python
def create_yaml_files():
    for condition in conditions:
        yaml_content = f"""
path: {output_dir}/{condition}
train: images/train
val: images/val
nc: {len(classes)}
names: {classes}
"""
        with open(f"{output_dir}/{condition}/voc2012_{condition}.yaml", 'w') as f:
            f.write(yaml_content)

    print("YAML files created successfully.")

create_yaml_files()
```

```
YAML files created successfully.
```

Figure 8: Creation of YAML file in Python

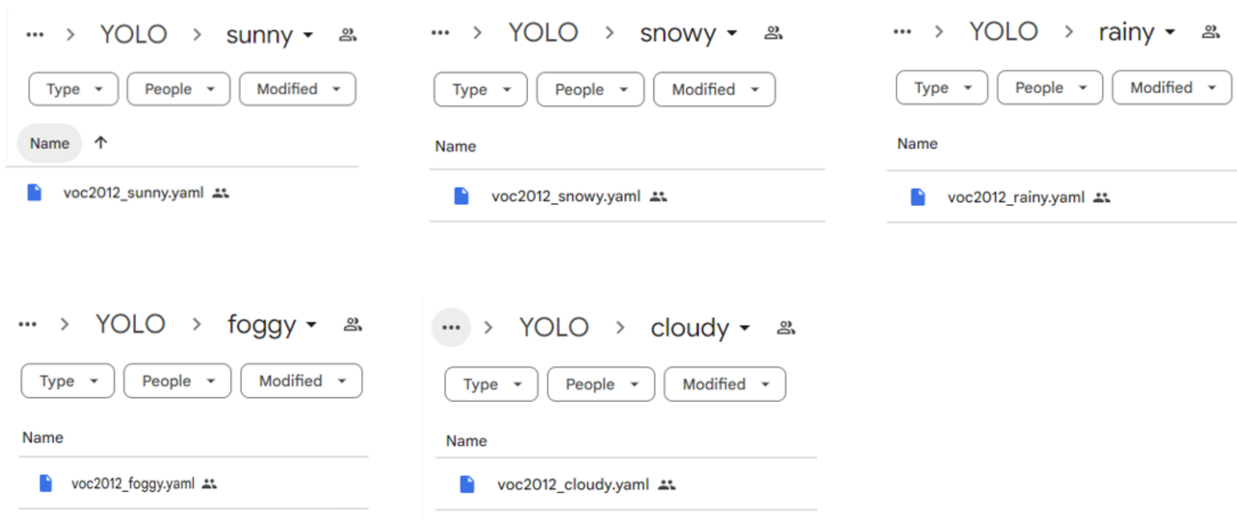

Figure 9: Creation of YAML files in the Drive

2.3 Hyper Parameter Tuning

Hyper Parameter tuning is a very important step in Machine Learning as finds the best combination of parameters, leading to more accurate and generalizable results. The hyperparameters that were set for this project is shown in the table below (Table 2). Figure 10 shows the code for hyperparameter tuning.

Table 2: Hyperparameter Tuning parameters

Hyperparameter	Values
Epochs	[10, 20]
Learning Rates	[0.001, 0.01, 0.1]
Batch Sizes	[16, 32, 64]
Image Sizes (pixels)	[640, 800, 1024]

```python
import os
from ultralytics import YOLO
import pandas as pd
import matplotlib.pyplot as plt
import torch

def train_model(condition, epochs, lr, batch_size, img_size):
    yaml_file = f"{output_dir}/{condition}/voc2012_{condition}.yaml"
    project_dir = f"{output_dir}/{condition}"
    model_name = f'yolov8_voc2012_{condition}_epochs_{epochs}_lr_{lr}_batch_{batch_size}_imgsz_{img_size}'

    # Check if the YAML file exists
    if not os.path.isfile(yaml_file):
        raise FileNotFoundError(f"Dataset YAML file '{yaml_file}' does not exist")

    model = YOLO('yolov8s.pt')

    # Check if CUDA is available and set the device
    # device = 'cuda' if torch.cuda.is_available() else 'cpu'

    # Train the model
    model.train(
        data=yaml_file,
        epochs=epochs,
        imgsz=img_size,
        batch=batch_size,
        name=model_name,
        lr0=lr,
        augment=True,
        project=project_dir,
        device= 'cpu'
    )
    print(f"Training for {condition} with epochs={epochs}, lr={lr}, batch_size={batch_size}, img_size={img_size} completed successfully.")

# Define the parameters
condition = 'sunny'
epochs_list = [10, 20]
learning_rates = [0.001, 0.01, 0.1]
batch_sizes = [16, 32, 64]
img_sizes = [640, 800, 1024]

# Iterate over the expanded hyperparameter grid
for epochs in epochs_list:
    for lr in learning_rates:
        for batch_size in batch_sizes:
            for img_size in img_sizes:
                train_model(condition, epochs, lr, batch_size, img_size)
```

```python
# Function to read the log files and return the data
def read_log_files(output_dir, condition, epochs_list, learning_rates, batch_sizes, img_sizes):
    logs = []
    for epochs in epochs_list:
        for lr in learning_rates:
            for batch_size in batch_sizes:
                for img_size in img_sizes:
                    log_dir = f"{output_dir}/{condition}/yolov8_voc2012_{condition}_epochs_{epochs}_lr_{lr}_batch_{batch_size}_imgsz_{img_size}/results.csv"
                    if os.path.exists(log_dir):
                        log_data = pd.read_csv(log_dir)
                        log_data.columns = log_data.columns.str.strip()  # Strip whitespace from columns
                        log_data['epochs'] = epochs
                        log_data['lr'] = lr
                        log_data['batch_size'] = batch_size
                        log_data['img_size'] = img_size
                        logs.append(log_data)
                    else:
                        print(f"Log file not found: {log_dir}")
    return pd.concat(logs, ignore_index=True)

# Function to plot losses
def plot_losses(logs):
    plt.figure(figsize=(12, 8))
    for (epochs, lr, batch_size, img_size), group in logs.groupby(['epochs', 'lr', 'batch_size', 'img_size']):
        plt.plot(group['epoch'], group['train/box_loss'], label=f'epochs={epochs} lr={lr} batch_size={batch_size} img_size={img_size}')

    plt.xlabel('Epoch')
    plt.ylabel('Box Loss')
    plt.title('Training Losses for Sunny Condition')
    plt.legend()
    plt.show()

# Read the log files and plot the losses
logs = read_log_files(output_dir, condition, epochs_list, learning_rates, batch_sizes, img_sizes)
plot_losses(logs)
```

Figure 10: Hyperparameter Tuning code

After this, training loss for 10 and 20 epochs was printed in graphical format, that led to the choosing of the appropriate hyperparameter of the model. Figure 11 below depicts the Training loss graph of Sunny condition with 10 and 20 epochs.

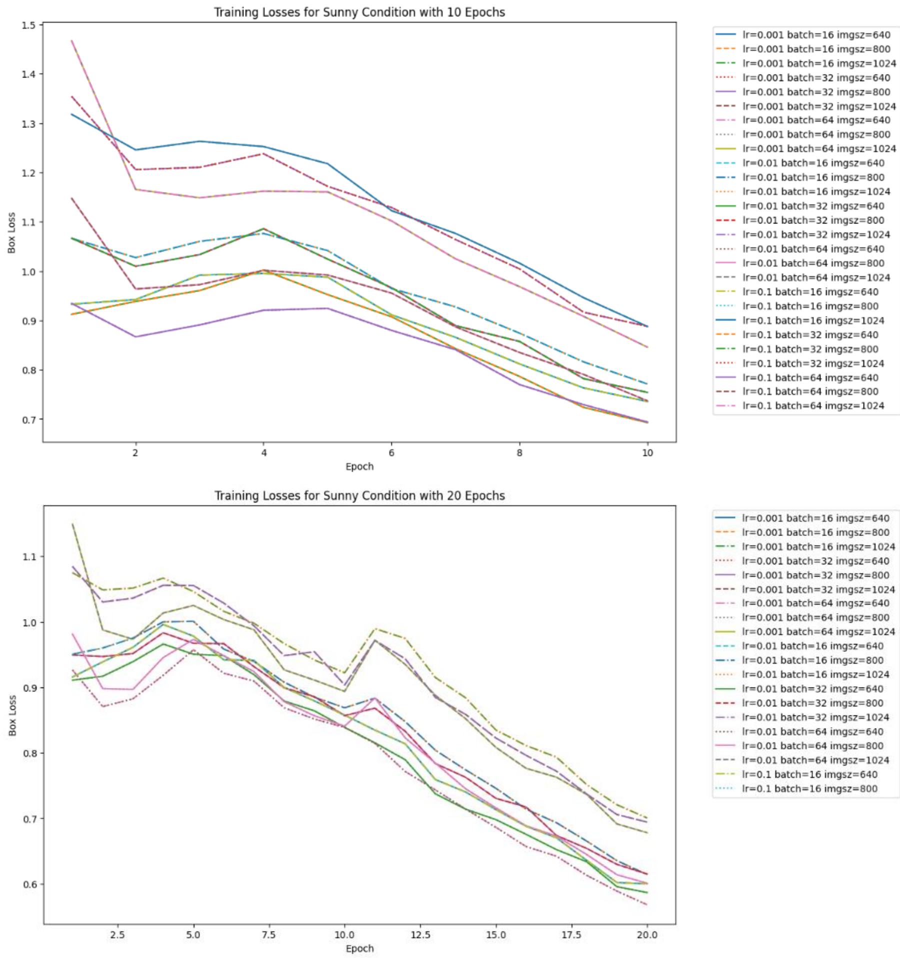

Figure 11: Training loss for Epochs 10 and 20

Looking at the graph, Table 3 below depicts the appropriate hyperparameter, that was chosen for the task at hand.

Table 3: Chosen Hyperparameter

Hyperparameter	Values
Epochs	20
Learning Rates	0.001
Batch Sizes	16
Image Sizes (pixels)	640

Post the hyperparameter tuning, the training of the data for all weather types began.

2.4 Data Training

Figure 11 below shows the code for the model training for all weather types. It loops the YAML file present in each of the weather folders and train the dataset (weather wise) with respect to the chosen hyperparameters.

TRAIN MODEL FOR ALL WEATHER TYPE

```
from ultralytics import YOLO

def train_model(condition):
    yaml_file = f"{output_dir}/{condition}/voc2012_{condition}.yaml"
    save_dir = f"{output_dir}/{condition}"
    model_name = f'yolov8_voc2012_{condition}'

    model = YOLO('yolov8s.pt')  # You can change this to 'yolov8n.pt', 'yolov8m.pt', etc.

    # Train the model
    model.train(
        data=yaml_file,
        epochs=20,
        imgsz=640,
        batch=16,
        name=model_name,
        augment=True,
        save_dir=save_dir,  # Specify the directory to save the weights
        project = '1',
        device = 'cuda'
    )
    print(f"Training for {condition} condition completed successfully.")
```

```
for condition in conditions:
    print(f"Starting training for {condition} condition...")
    train_model(condition)
    print(f"Finished training for {condition} condition.\n")
```

Figure 11: Training of the model on all-weather type

Once the data gets trained, next step is to run the inference on the validation images. Before the inferences, the novel idea of fetching the real-time weather data is done using the weather API.

2.5 The Novel weather idea

The real-time weather of the user, who is going to use the model for inference, will be fetched first for inference. The code in Figure 12 shows the logic that will be used to pick the proper weather for inference. For this example, Mumbai, India was taken as the location as Dublin was sunny and I needed to check for a place that is not sunny.

```python
def categorize_weather(api_weather):
    weather_mapping = {
        'sunny': ['clear', 'sun', 'sunny', 'mostly sunny', 'partly sunny', 'fair'],
        'cloudy': ['cloud', 'cloudy', 'overcast', 'mostly cloudy', 'partly cloudy'],
        'foggy': ['fog', 'mist', 'haze', 'foggy', 'misty'],
        'rainy': ['rain', 'rainy', 'drizzle', 'showers', 'thunderstorm', 'light rain', 'heavy rain'],
        'snowy': ['snow', 'snowy', 'sleet', 'blizzard', 'snow showers', 'light snow', 'heavy snow']
    }

    # Normalize the input to lowercase
    api_weather = api_weather.lower()

    # Check the weather description and map it to the predefined categories
    for category, keywords in weather_mapping.items():
        for keyword in keywords:
            if keyword in api_weather:
                return category

    # Default to 'cloudy' if no match is found
    return 'cloudy'
```

```python
import requests

def get_weather(api_key, location):
    url = f"http://api.weatherapi.com/v1/current.json?key={api_key}&q={location}&aqi=no"
    response = requests.get(url)
    if response.status_code == 200:
        weather_data = response.json()
        if 'current' in weather_data:
            condition = weather_data['current']['condition']['text']
            cetegorized_weather = categorize_weather(condition)
            return cetegorized_weather
        else:
            print("Weather data not found in the response.")
            return None
    else:
        print(f"Error fetching weather data: {response.status_code}")
        return None
```

```python
# Fetch current weather condition
api_key = 'aaa94b2ae6e8483a95a135306240108'  # Replace with your WeatherAPI key
location = 'mumbai'  # Specify the location
current_weather = get_weather(api_key, location)
print(current_weather)
```

```
rainy
```

Figure 12: Current weather selection for inference

2.6 The Inference

This is the final section of the code where now the inference will be made on the rainy weather images. As the current weather is set as rainy, the weights file from rainy folder will be used to make the inference. The code in figure 13, depicts the code for the description generation and inference and speech function, necessary to create a description of the photo and convert the output to audio.

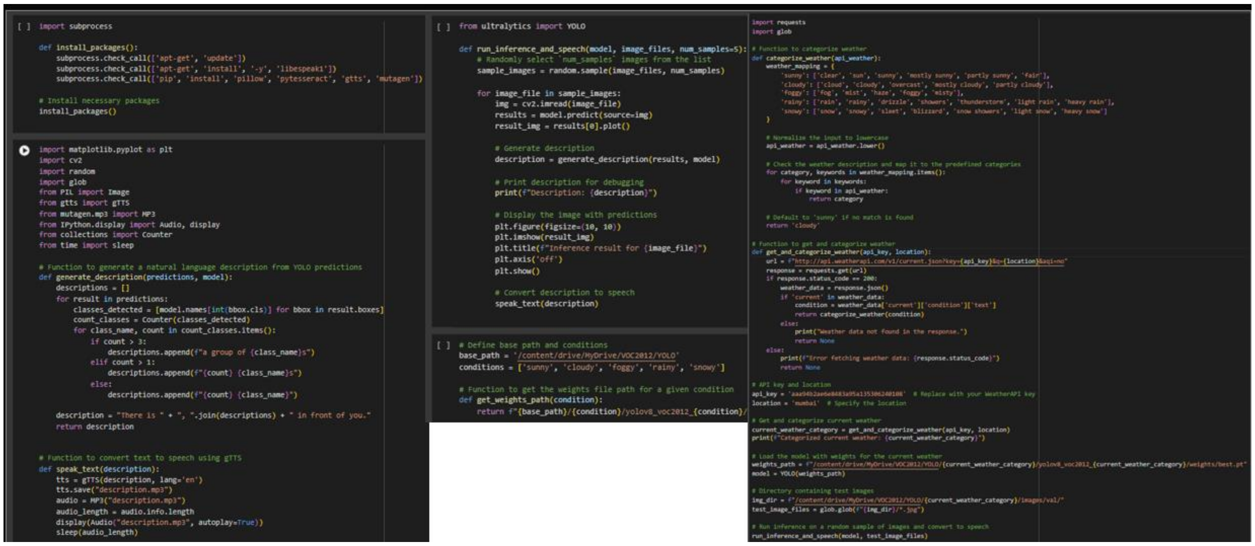

Figure 13 – Code for running inference and converting to Audio (in continuation in three columns)

Finally, the figure 14, 15, 16, 17, and 18 shows the inference done on 5 random images.

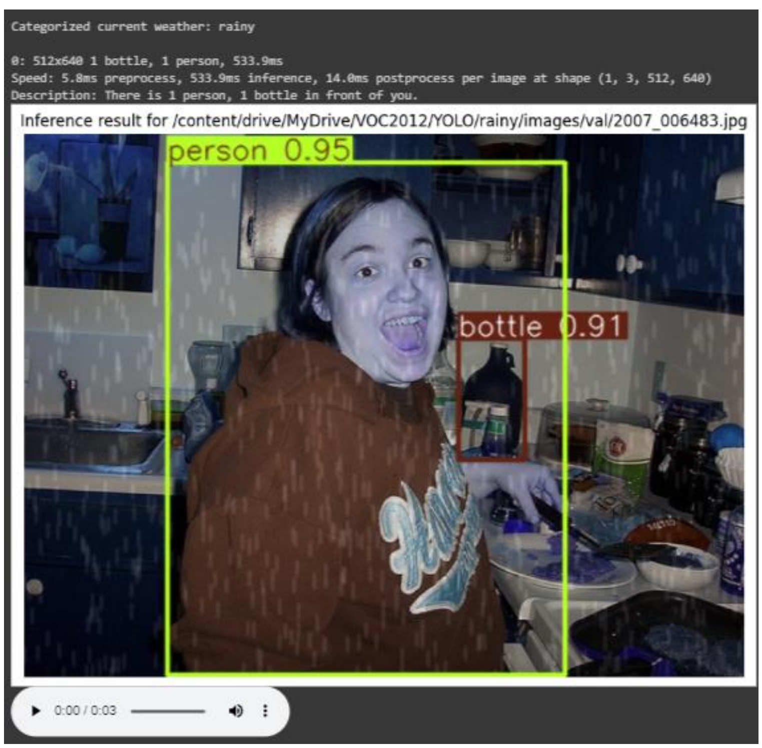

Figure 14: There is 1 person, 1 bottle in front of you

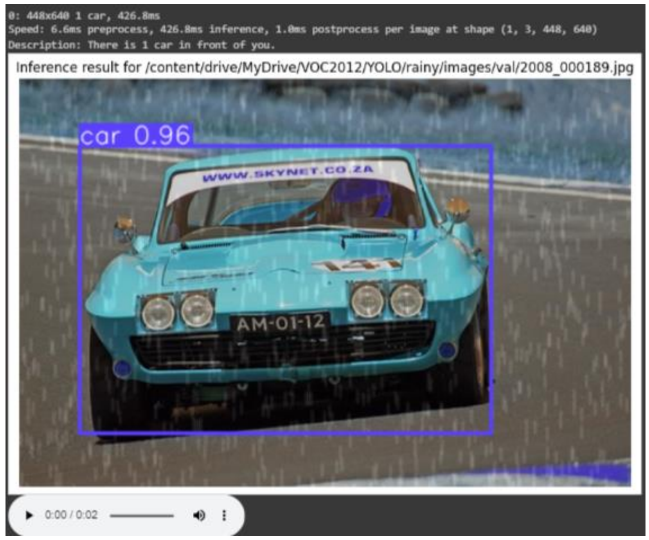

Figure 15: There is 1 car in front of you

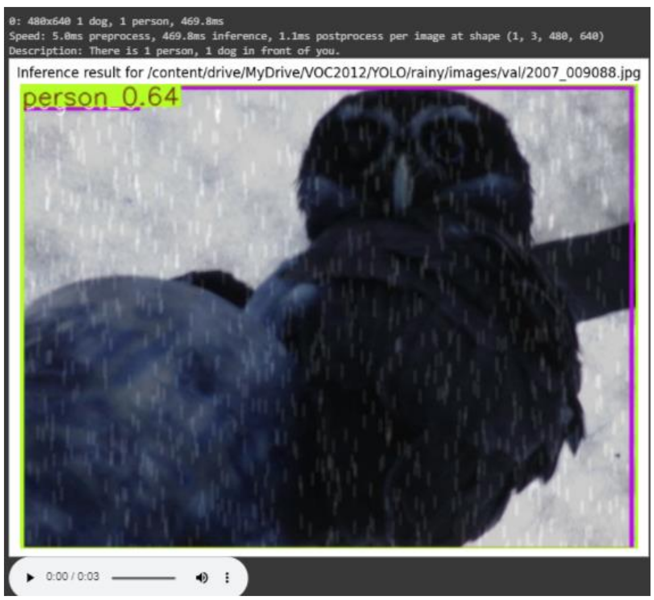

Figure 16: There is 1 person, 1 dog in front of you

Figure 17: There is 1 dining table in front of you

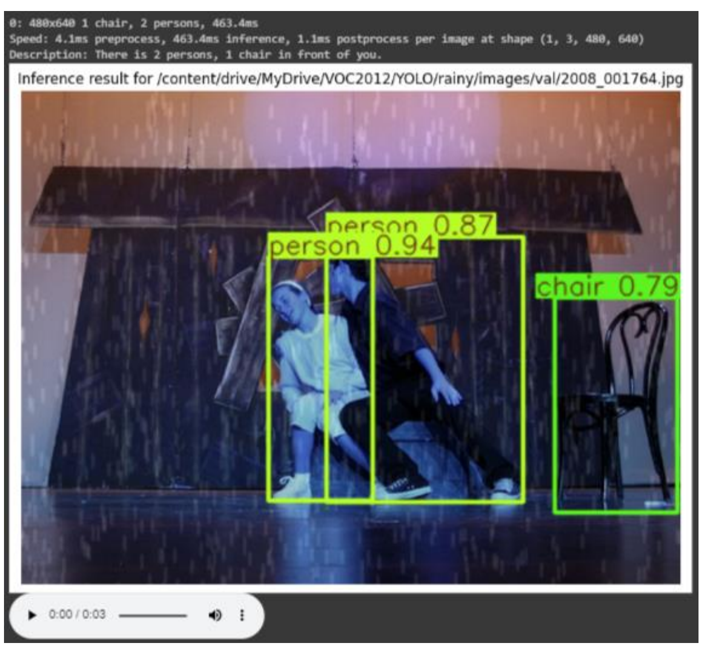

Figure 18: There is 2 persons, 1 chair in front of you