National College of Ireland

Configuration Manual

MSc Research Project
Artificial Intelligence

Muhammad Usama
Student ID: X23240636

School of Computing
National College of Ireland

Supervisor: Anderson Augusto Simiscuka

National College of Ireland
Project Submission Sheet
School of Computing

National College of Ireland

Student Name:	Muhammad Usama
Student ID:	X23240636
Programme:	Artificial Intelligence
Year:	2024
Module:	MSc Research Project
Supervisor:	Anderson Augusto Simiscuka
Submission Due Date:	12/12/2024
Project Title:	Configuration Manual
Word Count:	XXX
Page Count:	9

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:	12th December 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

Muhammad Usama
X23240636

1 Introduction

This paper presents a description of the setup and coding process required for the implementation of the project, namely *Solar Panel Automated Anomaly Detection and Localization Using Deep Learning*. It also identifies the hardware and software environment needed for research to achieve comparable results. This section contains procedures and essential programming environments as well as the necessary processes required for the effective implementation of the intended system. The manual ensures the correct implementation, training, evaluation, and fault localization of both the anomaly detection model and the localization model.

2 Setup and Configuration

2.1 Software Configurations

The implementation and training of models in this research were conducted using Python 3.10 as the primary programming language. The code was executed in Jupyter Notebooks Pro, which provided access to a high-RAM runtime environment to support efficient model training and experimentation.

2.1.1 Main Libraries

- **TensorFlow**: For developing and training the anomaly detection model (VGG19).
- **Ultralytics YOLOv8**: For implementing the fault localization model.
- **Other Supporting Libraries**: NumPy, Pandas, Matplotlib, and OpenCV for data preprocessing, visualization, and analysis.

3 Data Collection

Datasets have been selected and downloaded from the Kaggle source, which is mentioned here:(Dataset; 2024)

The annotation files were created manually and are stored on Google Drive. These can be accessed at the this link:(Annotation; 2024)

4 Data Transformation

The dataset for this project was downloaded from Kaggle and then uploaded to Google Drive for easy access within the Jupyter Notebook environment. For the anomaly detection task in the first part of the project, the dataset was preprocessed and categorized into six classes: Bird-drop, Clean, Dusty, Electrical-Damage, Physical-Damage, and Snow-Covered. These images were resized to 244x244 pixels to meet the input requirements of the VGG19 model, and pixel values were normalized using TensorFlow's `vgg19.preprocess_input` function.

In the second part of the project, focusing on anomaly localization, YOLOv8s was employed, which requires bounding box annotations in its specific format. To create these annotation files, the RoboFlow tool was utilized. Images corresponding to four classes—Physical-Damage, Electrical-Damage, Bird-Drop, and Snow-Covered—were uploaded to RoboFlow. Bounding boxes were manually drawn for each anomaly, and the annotations were exported in YOLO format. These annotation files were essential for training the YOLOv8s model to detect and localize faults effectively.

Here I upload the dataset for the annotation of the files:

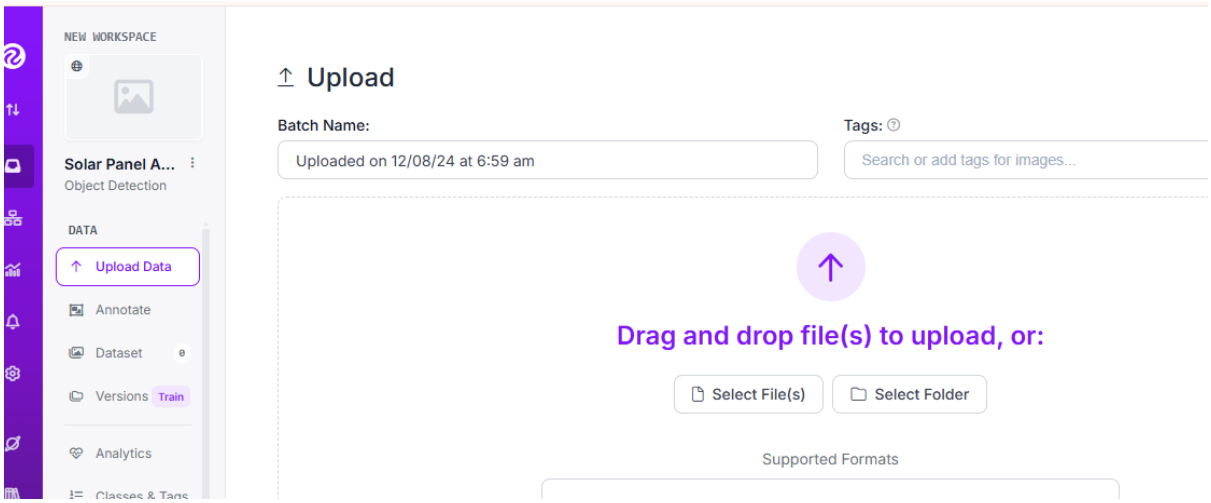

Figure 1: ROBOFlow

And after uploading the dataset I make the bounding boxes for finding the configuration values, After uploading the data, then next I did the manual annotation, because some images have the multiple faults, so I make bounding boxes on all faults manually. So here is how the annotation I started.

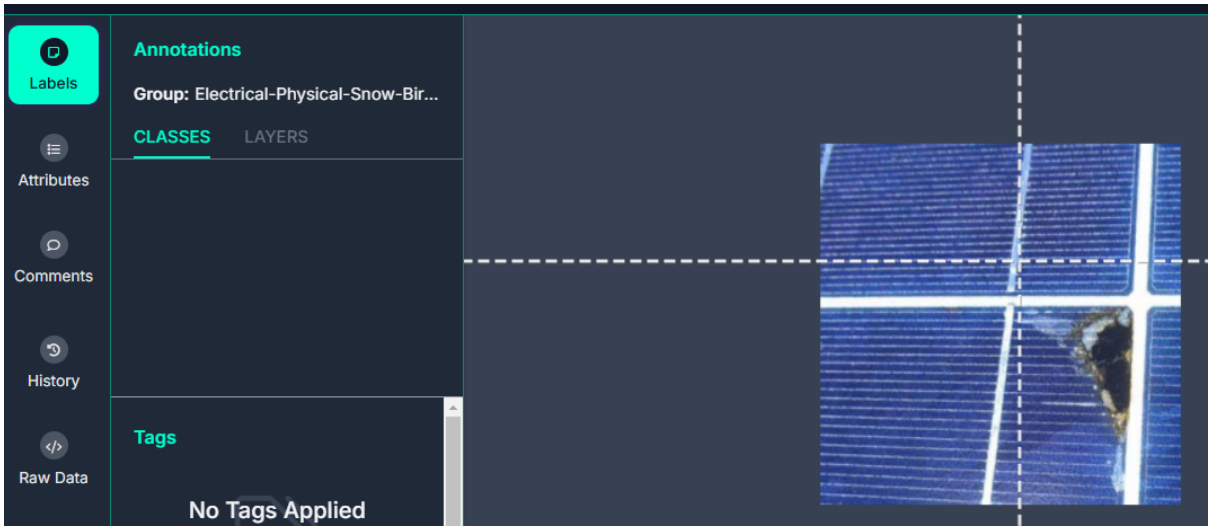

Figure 2: Dataset Annotation

After that the next step is to draw the bounding boxes,

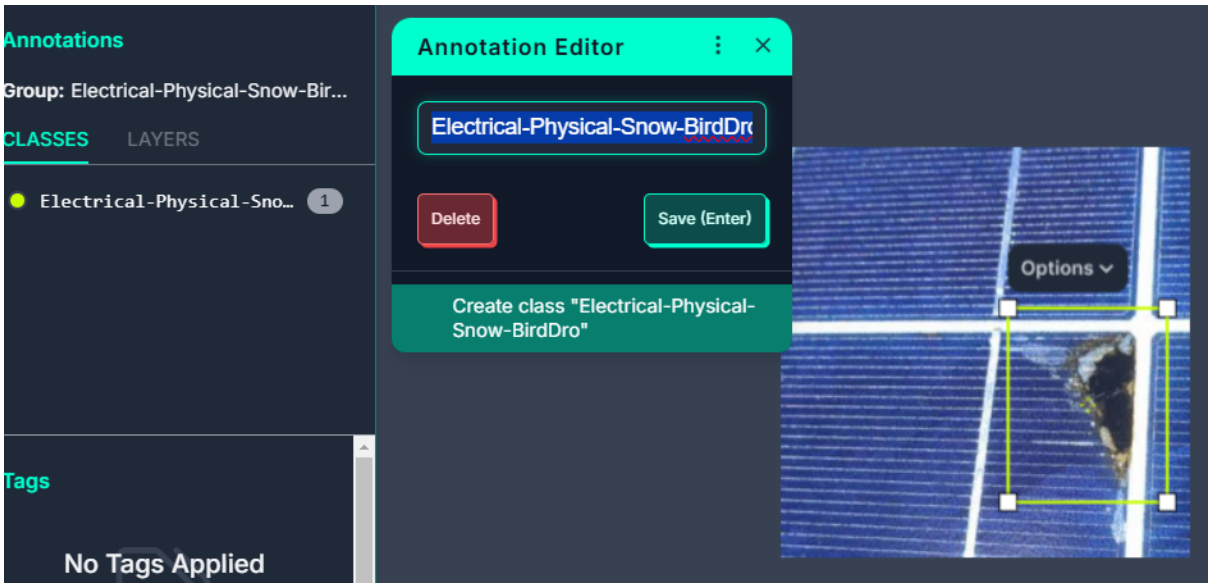

Figure 3: Annotation file

So after the bounding boxes, ROBOFlow will make the Annotation file, so YOLO V8s model detect the annotataion file.

5 Data Modeling

This section details the implementation of the anomaly detection and localization models. Both parts of the project depends on essential Python libraries, including `TensorFlow`, `OpenCV`, and the `Ultralytics YOLO` library. These libraries are prerequisites for running of the VGG19 and YOLOv8s models.

Assuming these libraries are already installed in the Python environment, they can be installed if required by using the following command:

```
pip install libraryName
```

The models were implemented and executed within a Jupyter Notebook, ensure the efficient and interactive development process.

```
import os
import random

import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

import cv2
from cv2 import resize
from glob import glob
from tqdm import tqdm

import tensorflow as tf
import keras
from keras.callbacks import EarlyStopping,ModelCheckpoint
from sklearn.metrics import confusion_matrix , accuracy_score
from sklearn.metrics import classification_report
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder

from tensorflow.keras import layers, models
from tensorflow.keras.applications import MobileNetV2

import warnings
warnings.filterwarnings("ignore")
```

Figure 4: Desired Libraries

```
import zipfile
import os
import cv2
import numpy as np
from sklearn.model_selection import train_test_split
from keras.utils import to_categorical
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import VGG16
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten  # Ensure Flatten i
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
import random
import matplotlib.pyplot as plt
```

Figure 5: Python Libraries

5.1 Data Loading

```
img_height = 244
img_width = 244
train_ds = tf.keras.utils.image_dataset_from_directory(
  "/content/drive/My Drive/VGG19/Faulty_solar_panel",
  validation_split=0.2,
  subset='training',
  image_size=(img_height, img_width),
  batch_size=32,
  seed=42,
  shuffle=True)

val_ds = tf.keras.utils.image_dataset_from_directory(
  "/content/drive/My Drive/VGG19/Faulty_solar_panel",
  validation_split=0.2,
  subset='validation',
  image_size=(img_height, img_width),
  batch_size=32,
  seed=42,
  shuffle=True)
```

```
Found 869 files belonging to 6 classes.
Using 696 files for training.
Found 869 files belonging to 6 classes.
Using 173 files for validation.
```

Figure 6: Data Laoding

5.2 Model Loading

```
base_model = tf.keras.applications.VGG19(
    include_top=False,
    weights='imagenet',
    input_shape=(img_height, img_width, 3)
)
base_model.trainable = False
```

```
Downloading data from https://storage.googleapis.com/tenso
80134624/80134624 ━━━━━━━━━━━━━━━━━━━━ 5s 0us/step
```

```
inputs = tf.keras.Input(shape=(img_height, img_width, 3))
x = tf.keras.applications.vgg19.preprocess_input(inputs)
x = base_model(x, training=False)
x = tf.keras.layers.GlobalAveragePooling2D()(x)
x = tf.keras.layers.Dropout(0.3)(x)
outputs = tf.keras.layers.Dense(90)(x)
model = tf.keras.Model(inputs, outputs)
```

Figure 7: Model VGG19

Model loaded with fine tunning

```
model.compile(optimizer=tf.keras.optimizers.Adam(0.001),
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

```
# fine tuning
base_model.trainable = True
for layer in base_model.layers[:14]:
    layer.trainable = False
model.summary()
```

Figure 8: Model with Fine Tunning

```
names:
  0: "Physical-Damage"
  1: "Electrical-Damage"
  2: "Bird-Drop"
  3: "Snow_Covered"

Ultralytics 8.3.43 🚀 Python-3.10.12 torch-2.5.1+cu121 CUDA:0 (NVIDIA A100-SXM4-40GB, 40514MiB)
engine/trainer: task=detect, mode=train, model=yolov8s.pt, data=/content/drive/My Drive/ROBO/data.yaml, epochs=60, time=None, patience=100, batch=16, imgsz=640,
Overriding model.yaml nc=80 with nc=4
```

Figure 9: Model

5.3 For Fault Localization

```
Loading Training Data...
Loaded 244 training images and 244 annotations.

Loading Validation Data...
Loaded 69 validation images and 69 annotations.

Loading Test Data...
Loaded 37 test images and 37 annotations.

First training image shape: (640, 640, 3)
First training image annotations: [(1.0, 0.40546875, 0.5453125, 0.15234375, 0.253125)]

First validation image shape: (640, 640, 3)
First validation image annotations: [(1.0, 0.3421875, 0.64921875, 0.17890625, 0.26484375)]

First test image shape: (640, 640, 3)
First test image annotations: [(1.0, 0.51484375, 0.4484375, 0.17265625, 0.37421875)]
```

Figure 10: Dataset loading

5.4 Model Summary

```
 21                  -1  1   1969152  ultralytics.nn.modules.block.C2f             [7
 22        [15, 18, 21]  1   2117596  ultralytics.nn.modules.head.Detect           [4
Model summary: 225 layers, 11,137,148 parameters, 11,137,132 gradients, 28.7 GFLOPs
```

Figure 11: Dataset loading

6 Evaluation

The evaluation of models for both classification and fault localization tasks yielded the following results:

1. For classification, the VGG19 model with transfer learning and fine-tuning achieved the highest accuracy of 96% (training) and 84% (validation), outperforming other models on the same dataset.

2. For fault localization, the YOLOv8s model performed well for single-class detection, achieving an mAP@50 of 75%. However, when trained on all four classes, the mAP@50 decreased to 60%, due to dataset quality and manual annotation limitations.

The evaluation results are summarized in the tables below.

Model	Techniques Used	Accuracy (%)
VGG16	None	67
VGG16	Data Augmentation	75
VGG16	Data Augmentation + Transfer Learning	79
VGG19	Transfer Learning + Fine-Tuning	**96 (Train)** / 84 (Validation)

Table 1: Comparison of VGG Models with Different Techniques

Model	Classes	mAP@5
YOLOv8s	Physical Damage Only	75
YOLOv8s	All Classes (Physical Damage, Electrical Damage, Snow Covered, Bird Drop)	60

Table 2: Fault Localization Results

7 Code Resources

The Jupyter Notebook files ('.ipynb') and other related resources for the evaluation and comparison of VGG models are available in the shared Google Drive folder. Access the folder using the link below:
Google Drive Folder - VGG Models and Resources

References

Annotation (2024). Google drive folder for solar panel annotations, `https://drive.google.com/drive/folders/1QiWAj0cpLELiVqmzm5wgXiOUssiPl4Dj?usp=drive_link`. Manually created annotations. Accessed: 2024-12-10.

Dataset, K. (2024). Solar panel clean and faulty images, `https://www.kaggle.com/datasets/pythonafroz/solar-panel-clean-and-faulty-images`. Dataset downloaded from Kaggle. Accessed: 2024-12-10.