National College of Ireland

Configuration Manual

MSc Research Project
Data Analytics

Jeseema Farhath Vesakkar Ansari
Student ID: x23129557@student.ncirl.ie

School of Computing
National College of Ireland

Supervisor: Vladimir Milosavljevic

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: Jeseema Farhath Vesakkar Ansari

Student ID: x23129557@student.ncirl.ie

Programme: Data Analytics **Year:** 2023-2024

Module: MSc Research Project

Lecturer: Vladimir Milosavljevic

Submission Due Date: 12-08-2024

Project Title: Transfer Learning and Fine-Tuned Faster R-CNN for Improved Insect Detection in Agriculture

Word Count: 1488 **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: Jeseema Farhath Vesakkar Ansari

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):	

Configuration Manual

Jeseema Farhath Vesakkar Ansari
Student ID: X23129557

1 Introduction

This research aims to enhance insect detection in agricultural fields using transfer learning models and a fine-tuned Faster R-CNN model. The project involves training several pre-trained models and implementing hyperparameter tuning to improve accuracy and efficiency. This configuration manual details all necessary steps for replication, from environment setup to model evaluation.

2 System Configuration

The project was implemented in Google Collaboratory. For Python scripts, Google provides free cloud-based servers; nevertheless, there are limitations. Google Collab Pro can be used with more RAM and GPU use. An Intel(R) Core (TM) i5-1035G1 CPU running at 1.00GHz 1.19 GHz, a Tesla T4 GPU with 2496 cores, 12 gigabytes of DDR5 VRAM, 30 gigabytes of accessible disk space, and 13GB of RAM are all part of the system configuration at the Google Collaboratory for this project.

3 Data Collection

The dataset utilized for this research comprises 15 distinct classes of insects, each posing significant threats to agricultural practices and crop production. The dataset contains a total of 1662 images, meticulously labeled to facilitate accurate detection and identification, with each class representing a specific type of insect. The primary objective is to develop a robust system for effectively detecting and classifying these insects, benefiting both residential areas and farms by mitigating the detrimental effects on crop quality and improving farmers' earnings(Madau *et al.*, 2020). This structured dataset, sourced from various open datasets, was split into training, validation, and testing sets, and subjected to transformations such as random flips, normalization, and resizing to prepare them for model training and evaluation. This approach ensures that the models are trained on high-quality data, leading to better performance and more accurate insect detection in agricultural fields.

4 Environment Setup

The project was carried out on google collab, to use the dataset on the collab notebook, after downloading the dataset from Kaggle and unzipping it, the was uploaded into google drive as the Figure 1 illustrates. As a result, the machine need is no longer limited. Later, to access the dataset, it was connected to the Google Collab notebook. For this, as seen in Figure 2, we connected the Google Drive using the predetermined code that Google provided. As a security measure during the mounting google disk, Google Collab will ask

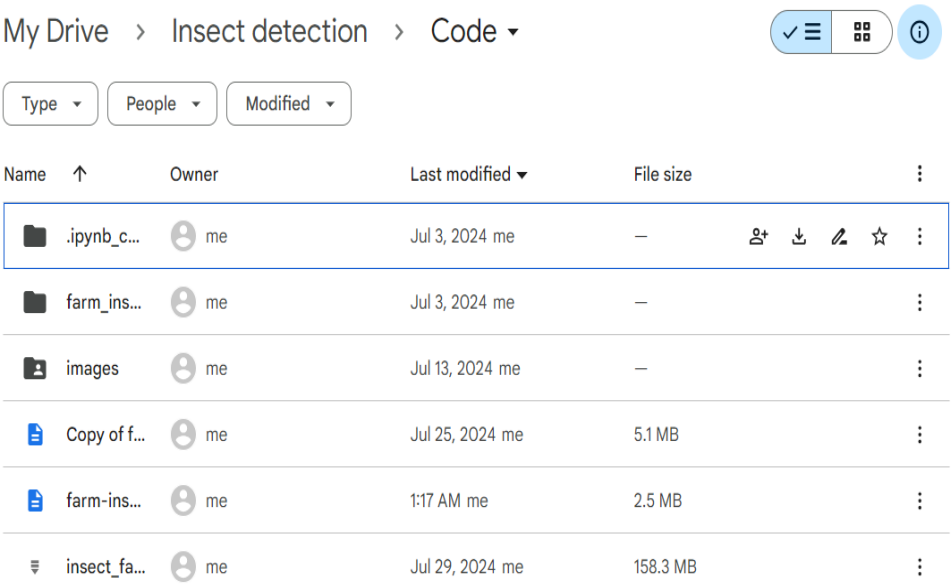

Figure 1: Saved files in Google Drive

you to authenticate your account before allowing you to access the drive while mounting Google Disk as shown in the Figure 2. Since our data consists of images, switching the runtime environment to the GPU could aid in enhancing performance. The coding is done with Python 3.9.3, and Google Collab provides a ready-made Jupyter notebook configuration.

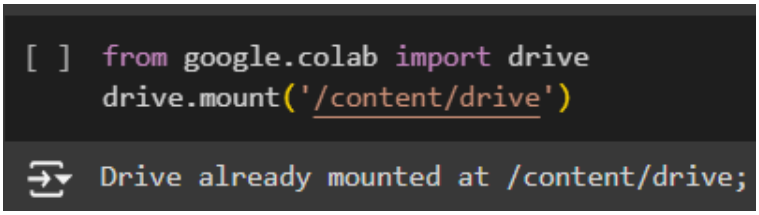

Figure 2: Mounting to drive

earnings. This structured dataset, sourced from various open datasets, was split into training, validation, and testing sets, and subjected to transformations such as random flips, normalization, and resizing to prepare them for model training and evaluation. This approach ensures that the models are trained on high-quality data, leading to better performance and more accurate insect detection in agricultural fields

5 Data Exportation

5.1 Libraries Imports

As illustrated in the Figure 3, the necessary libraries that are used throughout the projects are installed and imported into the collab notebook

```
[ ]  # Management Modules
     import os
     import random
     from glob import glob
     from tqdm import tqdm
     from collections import Counter

     # Data loading and transformation
     import numpy as np
     import pandas as pd
     import tensorflow as tf
     from tensorflow import keras
     from sklearn.model_selection import train_test_split
     from sklearn.utils.class_weight import compute_class_weight

     # Data Visualization
     import plotly.express as px
     import matplotlib.pyplot as plt

     # Pre-trained models
     from tensorflow.keras.applications import ResNet50V2, ResNet152V2, MobileNetV2, Xception

     # Model Layers
     from tensorflow.keras import layers
     from tensorflow.keras import Sequential
     from tensorflow.keras.layers import Dense, Dropout, GlobalAveragePooling2D

     # Model Checkpoints
     from tensorflow.keras.callbacks import EarlyStopping
     from tensorflow.keras.callbacks import ModelCheckpoint

     # Model Hyertunig and Metrices
     from keras_tuner.tuners import Hyperband
     from sklearn.metrics import classification_report, confusion_matrix

[ ]  !pip install keras-tuner
```

Figure 3: Python Library Imports

6 Exploratory Data Analysis

6.1 Checking Distribution of Classes

```
[ ]  # Compute the sample size for each class
     class_sample_size = [len(os.listdir(directory_path + class_name))
                          for class_name in class_names]

     # Calculate the total number of images
     total_n_images = sum(class_sample_size)

     # Calculate the percentage of sample size for each class
     percentage_sample_size = [value / total_n_images *
                               100 for value in class_sample_size]

     # Sort the class sample sizes in descending order
     sorted_class_sample_size = sorted(class_sample_size, reverse=True)

     # Display the class names, data size, percentage, and rank
     class_distribution_df = pd.DataFrame({
         "Class Names": class_names,
         "Data Size": class_sample_size,
         "Percentage(%)": percentage_sample_size
     })

     # Calculate the rank of each class based on data size
     class_distribution_df["Rank"] = class_distribution_df["Data Size"].rank(
         method="dense", ascending=False)

     # Show the data
     class_distribution_df
```

Figure 4: Exploratory Data Analysis

The EDA we can do with an image dataset is on the class distribution, namely the Binary Class and Multi Class distributions.

6.2 Class weights

One of the ways of handling the class imbalance issue during the training process is presented. Computing and analysing class weights as shown in the Figure 5 standardize the instances of each class in the dataset by giving a weight to each class in a dataset. With help of weights, the model pays more attention to the minority class during training and the class imbalance issue is less severe.

```
# Computer class weights.
class_weights = [total_n_images / (N_CLASSES * class_samples)
                 for class_samples in class_sample_size]

# Add this data into the data frame
class_distribution_df["Class Weights"] = class_weights

# Show the updated data
class_distribution_df
```

Figure 5: Computing Class Weights

7 Data Preprocessing

In data preparation for this project, all the image paths and corresponding class labels were gathered, after which the data was split using the train-test split function from scikit-learn with stratified sampling. This split made sure that class proportion was also maintained in the training set as 80%, and the rest of the 20% was split 90% validation and 10% test set as depicted in the Figure 6. The train data also went through an exercise of cross validation to ensure that the class label was correctly stratified based on the percentage indicated. Furthermore, to overcome the problem of imbalance in the given dataset, class weights were also determined with a prerequisite to train the model with the same importance of every type of insect class. These weights were then re-checked with the previous weights to ensure the correctness of the applied pre-processing.

```
[ ]  # Initialize empty lists to collect image paths and class labels
     all_image_paths = []
     class_labels = []

     # Collect all image paths and corresponding class labels
     for class_name in class_names:
         paths = glob(directory_path + f"{class_name}/*")
         all_image_paths.extend(paths)
         class_labels.extend([class_name] * len(paths))

     # Perform stratified data split
     train_images, valid_test_images = train_test_split(
         all_image_paths, train_size=0.8, test_size=0.2, stratify=class_labels)
     valid_images, test_images = train_test_split(
         valid_test_images, train_size=0.9, test_size=0.1)
```

Figure 6: Train, Test Split of the Dataset

8 Data Transformation

In preparation for feeding into the model, the data is subjected to several processes. Images are read, converted from BGR to RGB format and resized as per the common size 256 X 256 or 224 X 224 and then scaled between [0,1]. The class labels are obtained from the image file names and are either the actual names of the classes applied or indices of the classes. If specified, these labels are one-hot encoded as the Loss function can be categorical cross-entropy(Teixeira *et al.*, 2023). Likewise, the class weights are considered according to the occurrence for each class to handle the class imbalance issue, where classes that occur infrequently are given proper representation during training.

```python
class FarmInsects:
    """
    A class for loading and processing farm insect images dataset.
    """

    def __init__(self, class_names: list, n_classes: int, image_size: tuple = (256, 256, 3)):
        """
        Initialize FarmInsects class.

        Args:
            class_names (list): List of class names.
            n_classes (int): Number of classes.
            image_size (tuple): Size of the images (height, width, channels).
        """
        self.image_size = image_size
        self.class_names = class_names
        self.n_classes = n_classes

    def load_image(self, image_path: str) -> tf.Tensor:
        """
        Load and preprocess an image.

        Args:
            image_path (str): Path to the image file.

        Returns:
            tf.Tensor: Preprocessed image tensor.
        """
        image_file = tf.io.read_file(image_path)
        image = tf.image.decode_image(image_file, channels=3)
        image = tf.image.resize(image, self.image_size[:2])
        image = image / 255.

        return image
```

Figure 7: Data Transformation

9 Data Modelling

In this study, 5 models were employed: ResNet50V2, ResNet152V2, MobileNetV2, Xception. nd Faster R-CNN as shown in the Figure 8. Specific setups and hyperparameters were applied to each model during training in order to maximize performance on the emotion detection task. Two of the best models were saved to the specified folder as presented in the Figure 9.

ResNet152V2 model

```
# Initialize the ResNet152V2 model
resnet152_model = Model(model_name="ResNet152V2")

# Train the model
resnet152_model.train(
    trainXs, trainYs,
    validation_data=(validXs, validYs),
    verbose=0,
)

# Evaluate model performance on testing data.
resnet152_model.evaluate(testXs, testYs, visualize=True, verbose=0)
```

ResNet50V2 model

```
# Initialize the ResNet50V2 model
resnet50_model = Model(model_name="ResNet50V2")

# Train the model
resnet50_model.train(
    trainXs, trainYs,
    validation_data=(validXs, validYs),
    verbose=0,
)

# Evaluate model performance on testing data.
resnet50_model.evaluate(testXs, testYs, visualize=True, verbose=0)
```

MobileNetV2 model

```
# Initialize the MobileNetV2 model
mobilenetv2_model = Model(model_name="MobileNetV2")

# Train the model
mobilenetv2_model.train(
    trainXs, trainYs,
    validation_data=(validXs, validYs),
    verbose=0,
)

# Evaluate model performance on testing data.
mobilenetv2_model.evaluate(testXs, testYs, visualize=True, verbose=0)
```

Xception model

```
# Initialize the Xception model
xception_model = Model(model_name="Xception")

# Train the model
xception_model.train(
    trainXs, trainYs,
    validation_data=(validXs, validYs),
    verbose=0,
)

# Evaluate model performance on testing data.
xception_model.evaluate(testXs, testYs, visualize=True, verbose=0)
```

Faster R-CNN

```
import torch
from torchvision.models.detection import fasterrcnn_resnet50_fpn
from torchvision.models.detection.faster_rcnn import FastRCNNPredictor
from tqdm.notebook import tqdm
import torch.optim as optim

# Check device
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# Load the pre-trained Faster R-CNN model
model = fasterrcnn_resnet50_fpn(pretrained=True)

# Define number of classes
num_classes = len(class_names) + 1  # Assuming 'class_names' is defined elsewhere

# Modify the box predictor to have the right number of output classes (15 + 1 for background)
in_features = model.roi_heads.box_predictor.cls_score.in_features
model.roi_heads.box_predictor = FastRCNNPredictor(in_features, num_classes)

# Move the model to the correct device
model = model.to(device)

# Define the optimizer
optimizer = optim.SGD(model.parameters(), lr=0.005, momentum=0.9, weight_decay=0.0005)
num_epochs = 10z
```

Figure 8: Data Models

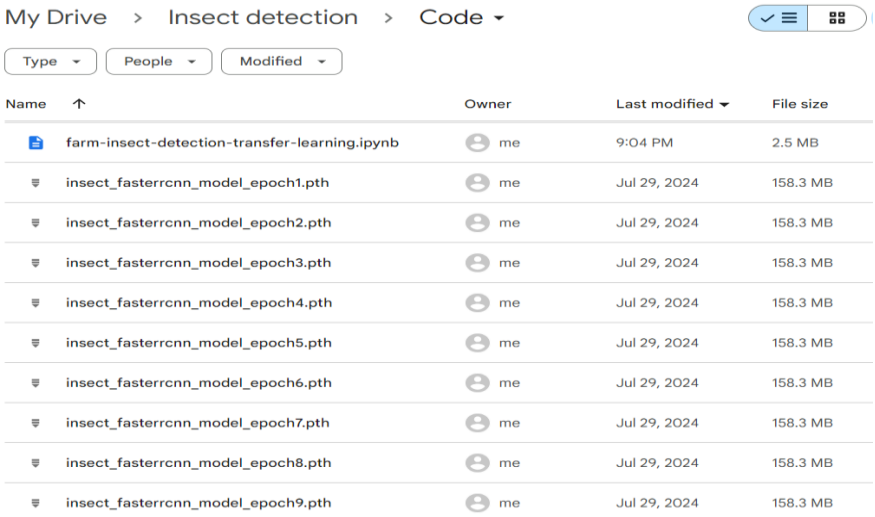

Figure 9: Model Saving

10 Training Model

Both pre-processed data sets were used for extended training of each model and the Figure 10 shows the training of Faster R-CNN model. To be evaluated on accuracy, loss, precision, recall, and F1 scores, the top-performing models were kept.

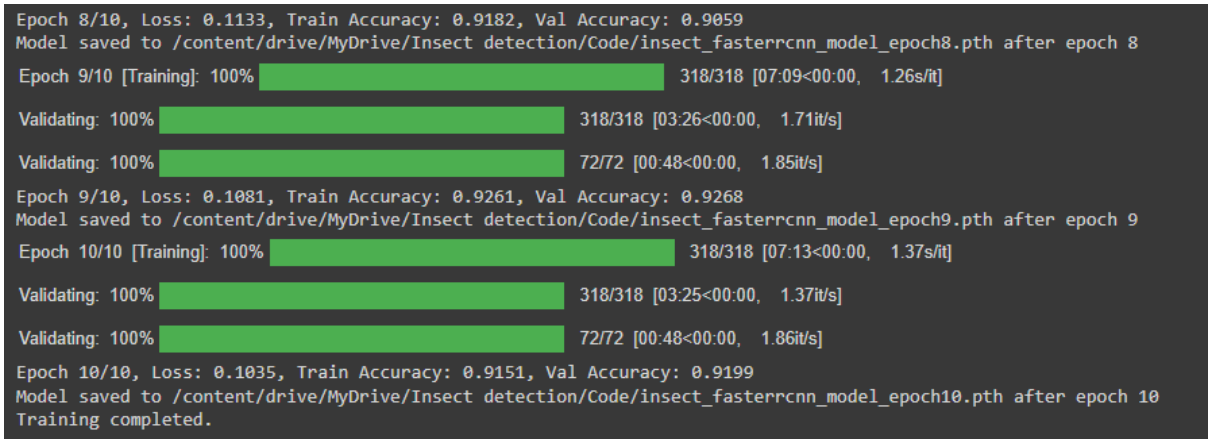

Figure 10: Training the Model

11 Evaluation

The model evaluation used several critical methods to determine the model's performance. To analyse the method's performance, Accuracy was used, which calculates the overall rate of correct predictions; furthermore, the Confusion matrix displayed the correct and incorrect predictions of each class. Accuracy of the positive predictions was measured using precision and a measure of how all the positive cases were identified using recall. The option of F1 Score, which represents the harmonic average of the entities of Precision and Recall was used as the measure of the overall performance of the model. To compare the effectiveness of distinct models.

References

Madau, F.A. *et al.* (2020) 'Insect Farming for Feed and Food Production from a Circular Business Model Perspective', *Sustainability*, 12(13), p. 5418. Available at: https://doi.org/10.3390/su12135418.

Teixeira, A.C. *et al.* (2023) 'Using deep learning for automatic detection of insects in traps', *Procedia Computer Science*, 219, pp. 153–160. Available at: https://doi.org/10.1016/j.procs.2023.01.276.