National College of Ireland

Configuration Manual

MSc Research Project
MSc. Data Analytics

Ajay Varma Chekuri
Student ID: x23213451

School of Computing
National College of Ireland

Supervisor: Dr David Hamill

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: Ajay Varma Chekuri

Student ID: x23213451

Programme: MSc. Data Analytics **Year:** 2024-2025

Module: MSc. Research Project

Lecturer: Dr David Hamill

Submission Due Date: 12-12-2024

Project Title: AI Applications in Precision Agriculture: Improving Crop Management, Yield Estimation, and Environmental Sustainability

Word Count: 451 Page Count: 6

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: Ajay Varma Chekuri

Date: 11-12-2024

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

Ajay Varma Chekuri
Student ID: x23213451

1 Introduction

This manual provides a comprehensive guide to the system setup, environment configuration, and workflow used in implementing two separate projects: a deep learning-based system for crop pest and disease detection and a machine learning-based system for crop yield prediction. Both projects aim to analyze agricultural data for predictive modeling and visual insights.

2 System Configuration

This section outlines the hardware and software requirements necessary for executing the projects.

2.1 Hardware Specifications

Component	Specification
Operating System	Windows 10
RAM	16 GB
Processor	Intel Core i7
Disk Space	10 GB (approx.)

2.2 Software Specifications

Software	Version/Tool
Programming Language	Python 3.9
Libraries	TensorFlow, Keras, Scikit-learn, Pandas, Matplotlib, Seaborn, PIL, OpenCV
IDE/Tools	Jupyter Notebook, Anaconda
Dataset Formats	CSV, Image Files
Browsers	Google Chrome, Edge

3 Environment Setup

3.1 Launching Jupyter Notebook

1. Open **Anaconda Navigator**.

2. Launch **Jupyter Notebook** from the dashboard.
3. Create a new .ipynb file to begin coding.

3.2 Data Preparation

- **For Pest and Disease Detection**: Place all the image datasets in a folder named Crop Pest and Disease Detection. (https://www.kaggle.com/datasets/nirmalsankalana/crop-pest-and-disease-detection)
- **For Crop Yield Prediction**: Use a CSV file named yield_df.csv. (https://www.kaggle.com/datasets/patelris/crop-yield-prediction-dataset?select=yield_df.csv)

3.3 Importing Necessary Libraries

The following libraries are essential for implementing the projects:

```
# Import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import os
from PIL import Image
from sklearn.model_selection import train_test_split
from sklearn.utils.class_weight import compute_class_weight
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, mean_absolute_error, mean_squared_error
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.callbacks import EarlyStopping
import matplotlib.image as mpimg
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
import cv2
import warnings
from PIL import ImageFile
```

4 Workflow and Implementation

4.1 Pest and Disease Detection

Step 1: Image Augmentation and Preprocessing

- Use `ImageDataGenerator` for augmenting and normalizing training and validation datasets.
- Example:

```
train_datagen = ImageDataGenerator(
    rescale=1./255,                # Normalize pixel values between 0 and 1
    rotation_range=40,             # Random rotation between 0 and 40 degrees
    width_shift_range=0.2,         # Randomly shift images horizontally
    height_shift_range=0.2,        # Randomly shift images vertically
    shear_range=0.2,               # Shearing transformation
    zoom_range=0.2,                # Random zoom in
    horizontal_flip=True,          # Randomly flip the image horizontally
    fill_mode='nearest',           # Fill in missing pixels with nearest ones
    validation_split=0.2           # Split the dataset into training (80%) and validation (20%)
)
```

Step 2: Model Architecture

- A convolutional neural network (CNN) with the following layers:
  - Convolutional and MaxPooling layers.
  - Fully connected Dense layers.
  - Dropout to prevent overfitting.
- Model compiled using the Adam optimizer and categorical crossentropy loss.

Step 3: Training and Evaluation

- Use early stopping to prevent overfitting:

```
# Define early stopping to avoid overfitting
early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)

# Train the model using the custom generator
history = model.fit(
    train_generator_safe,
    steps_per_epoch=train_generator.samples // train_generator.batch_size,
    epochs=20,
    validation_data=validation_generator_safe,
    validation_steps=validation_generator.samples // validation_generator.batch_size,
    callbacks=[early_stopping]
)
```

Step 4: Results Visualization

- Evaluation of the model:

```
# Evaluate the model
val_loss, val_acc = model.evaluate(validation_generator)
print(f"Validation Accuracy: {val_acc * 100:.2f}%")
print(f"Validation Loss: {val_loss:.4f}")
```

- Plot training and validation accuracy and loss:

```
# Plot training & validation accuracy and loss values
plt.figure(figsize=(12, 6))

# Accuracy plot
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(loc='upper left')

# Loss plot
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend(loc='upper left')
```

4.2 Crop Yield Prediction

Step 1: Data Preprocessing

- Clean the data, handle missing values, and apply transformations (e.g., log transformation on hg/ha_yield).
- Standardize the features:

```
# Standardize the features (mean = 0, variance = 1)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Step 2: Model Training

- Train machine learning models:
  - **Random Forest**

```
# Initialize the Random Forest Regressor
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)

# Train the model on the training data
rf_model.fit(X_train_scaled, y_train)
```

  - **Gradient Boosting**

```
# Gradient Boosting Model
gb_model = GradientBoostingRegressor(n_estimators=100, random_state=42)
gb_model.fit(X_train_scaled, y_train)
```

Step 3: Hyperparameter Tuning

- Use GridSearchCV for optimizing model parameters:

```
# Hyperparameter Tuning for Gradient Boosting
param_grid = {
    'n_estimators': [100, 200, 300],
    'learning_rate': [0.01, 0.1, 0.2],
    'max_depth': [3, 5, 7]
}

# Initialize GridSearchCV with Gradient Boosting Regressor
gb_regressor = GradientBoostingRegressor(random_state=42)
grid_search = GridSearchCV(estimator=gb_regressor, param_grid=param_grid, cv=3, scoring='neg_mean_squared_error', n_jobs=-1, verbose=2)

# Fit the model to the training data
grid_search.fit(X_train_scaled, y_train)
```

```
# Define the parameter grid for tuning Random Forest
rf_param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [10, 20, 30],
    'min_samples_split': [2, 5, 10]
}

# Initialize Random Forest Regressor
rf_regressor = RandomForestRegressor(random_state=42)

# Set up GridSearchCV with the parameter grid and cross-validation
rf_grid_search = GridSearchCV(estimator=rf_regressor, param_grid=rf_param_grid, cv=3, scoring='neg_mean_squared_error', n_jobs=-1, verbose=2)

# Fit the model to the training data
rf_grid_search.fit(X_train_scaled, y_train)
```

Step 4: Model Evaluation

- Evaluate models using metrics like MAE, RMSE, and R^2.
- Plot feature importance for Gradient Boosting:

```
# Plot feature importance
plt.figure(figsize=(8, 6))
sns.barplot(x=feature_importances, y=features)
plt.title('Feature Importance')
plt.xlabel('Importance')
plt.ylabel('Features')
plt.show()
```

References

Linaza, M.T., Posada, J., Bund, J., Eisert, P., Quartulli, M., Döllner, J., Pagani, A., G. Olaizola, I., Barriguinha, A., Moysiadis, T. and Lucat, L., 2021. Data-driven artificial intelligence applications for sustainable precision agriculture. Agronomy, 11(6), p.1227.

Sishodia, R.P., Ray, R.L. and Singh, S.K., 2020. Applications of remote sensing in precision agriculture: A review. Remote sensing, 12(19), p.3136.

Van Dijk, M., Morley, T., Rau, M.L. and Saghai, Y., 2021. A meta-analysis of projected global food demand and population at risk of hunger for the period 2010–2050. Nature Food, 2(7), pp.494-501.