National College of Ireland

# Configuration Manual

MSc Research Project
Programme Name

Arshil Deshmukh
Student ID: X23183063

School of Computing
National College of Ireland

Supervisor: Barry Haycock

**National College of Ireland**

**MSc Project Submission Sheet**

**School of Computing**

| | |
|---|---|
| **Student Name:** | Arshil Khalid Deshmukh |
| **Student ID:** | X23183063 |
| **Programme:** | Data Analytics **Year:** 2024 |
| **Module:** | MSc Research Project |
| **Lecturer:** | Barry Haycock |
| **Submission Due Date:** | 12/12/2024 |
| **Project Title:** | Configuration Manual |
| **Word Count:** | **895** **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:** Arshil Deshmukh

**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

Arshil Deshmukh
Student ID: X23183063
11-12-2024

# 1 Introduction

This configuration manual shows minimum hardware and software configurations to replicate the research project "**Improving Waste Sorting Systems: Enhancing Sustainability and Resource Recovery**". It also shows the libraries, code and each implementation steps with all the requirements. This documentation is designed to reproduce with accuracy and precision for further application of the research project.

# 2 System Requirements

## 2.1 Hardware Requirements

The below table1 shows the hardware requirements needed to for project execution.

| Components | Specifications |
|---|---|
| Processor | Intel Core i7 4.7Ghz |
| GPU | NVIDIA RTX 4050 |
| RAM | 16 GB |
| Storage | 1 TB SSD |
| Operating System | Windows |
| Display | 144Hz |

## 2.2 Software Requirements

The below table shows the software configurations

| Software Type | Software Name | Version |
|---|---|---|
| Programming Language | Python | 3.10.9 |
| Coding Environment | Jupyter Notebook | 6.5.1 |
| Platform | TensorFlow | 2.16.1 |

## 2.3 Required Libraries and packages

The below section shows us all the libraries and packages we have used in the process.

| Libraries | Version | Use |
|---|---|---|
| TensorFlow | v2.16.1 | Used to develop various models such as image recognition |
| Keras | 3 | Implementing Neural Networks |
| NumPy | 2.1.3 | a collection of high-level mathematical functions |
| scikit-learn | 1.6.0 | efficient tools for machine learning and statistical modeling |
| matplotlib.pyplot | 3.9.0 | used for creating static, animated, and interactive visualizations in Python |
| tensorflow.keras.applications | v2.16.1 | implementations of pre-trained deep learning models |
| tensorflow.keras.preprocessing.image | v2.16.1 | Is primarily used for loading, preprocessing, and augmenting image data |
| tensorflow.keras.utils | v2.16.1 | provides utility functions for various tasks |
| tensorflow.keras.models | v2.16.1 | It provides tools to create, save, load, and manage deep learning models |
| sklearn.metrics | 1.6.0 | It provides functions for evaluating model performance |
| sklearn.preprocessing | 1.6.0 | module is used for feature scaling, normalization, and transforming data |
| seaborn | 0.11.2 | Is used for creating informative and visually appealing statistical graphics |

**Figure 1 Libraries and packages**

# 3 Implementation of Code

In this section, we will see everything required to process the code successfully from scratch.

## 3.1 Creating Folder and loading all the required things.

The below figure2 shows the folder and all the codes files in it with data folder in it.

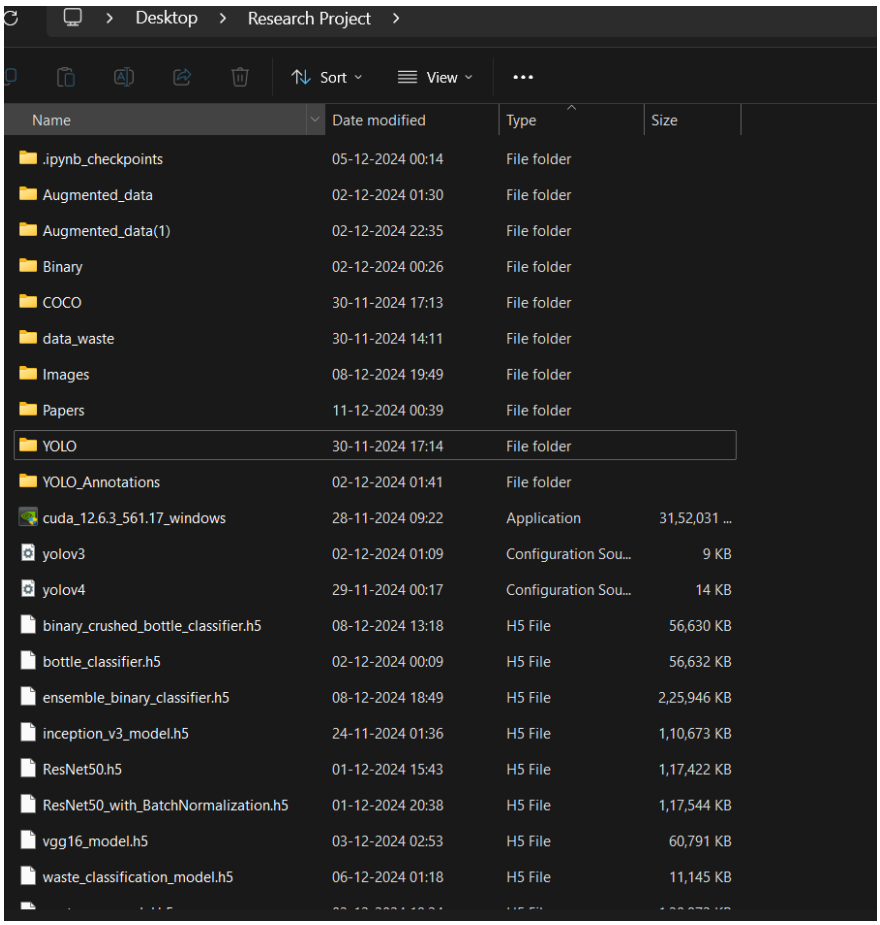


**Figure 2 Directory**

## 3.2 Data Preprocessing

In this process, we collected all the images and stored in a folder called data_waste. The collected data has been labelled with the same name as folder. In Fig 3 and Fig 4 Below

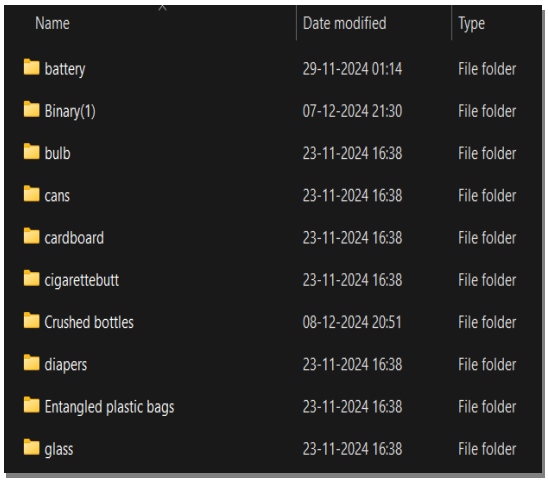


**Figure 3 Classes**

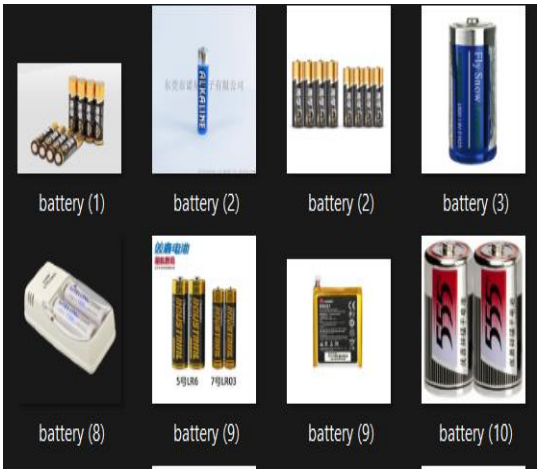


**Figure 4 Image labels**

## 3.3 Data splitting and Transformation

In this section, we did split the data in three categories train, valid and test. But first we did train and valid then out of it we made a test folder containing test images. Below Figure 5 shows the python code for train and valid data.

Splitting the data

```
import os
import shutil
from sklearn.model_selection import train_test_split

original_dataset_dir = 'C:/NCI/Research Project/Waste_data'
train_dir = 'C:/NCI/Research Project/train'
test_dir = 'C:/NCI/Research Project/test'

os.makedirs(train_dir, exist_ok=True)
os.makedirs(test_dir, exist_ok=True)

categories = os.listdir(original_dataset_dir)

test_size = 0.2

for category in categories:
    # Create class-specific directories in train/test folders
    os.makedirs(os.path.join(train_dir, category), exist_ok=True)
    os.makedirs(os.path.join(test_dir, category), exist_ok=True)

    # Get all image filenames for this category
    category_path = os.path.join(original_dataset_dir, category)
    images = os.listdir(category_path)

    # Split into train/test sets
    train_images, test_images = train_test_split(images, test_size=test_size)

    # Copy the images to the respective directories
    for image in train_images:
        shutil.copy(os.path.join(category_path, image), os.path.join(train_dir, category, image))
    for image in test_images:
        shutil.copy(os.path.join(category_path, image), os.path.join(test_dir, category, image))
```

**Figure 5 Splitting of data**

The above code shows splitting. One change is we use test_size for validation folder you can use valid_size and change as per requirements. Below Figure shows code for test folder creation. We took 5 images per class to keep data short.

```
import shutil
import random

# Correct the base directory path
base_dir = "C:/Users/arshi/Desktop/Research Project/data_waste"
train_dir = os.path.join(base_dir, "train")
test_dir = os.path.join(base_dir, "test")

# Create the test directory if it doesn't exist
os.makedirs(test_dir, exist_ok=True)

# Number of images to move per class
NUM_IMAGES = 5

# Check if the train directory exists
if not os.path.exists(train_dir):
    raise FileNotFoundError(f"Train directory does not exist: {train_dir}")

# Move images to the test folder
for class_name in os.listdir(train_dir):
    class_train_dir = os.path.join(train_dir, class_name)

    if not os.path.isdir(class_train_dir):
        continue  # Skip if it's not a directory

    files = os.listdir(class_train_dir)
    selected_files = random.sample(files, min(NUM_IMAGES, len(files)))

    for file in selected_files:
        src_path = os.path.join(class_train_dir, file)
        dst_path = os.path.join(test_dir, f"{class_name}_{file}")
        shutil.move(src_path, dst_path)

print("Test dataset created")
```

**Figure 6 Test Data folder creation**

Later on, we made an Augmented dataset out of it to make data more robust and better. Below Figure7 shows the Augmentation code.

```python
from tensorflow.keras.preprocessing.image import ImageDataGenerator, img_to_array, load_img
import os

# Initialize ImageDataGenerator with your augmentation settings
datagen = ImageDataGenerator(
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)

# Directory paths
input_dir = 'data_waste/train'
output_dir = 'Augmented_data'  # Output directory to save augmented images

# Loop through each subfolder in the input directory
for folder_name in os.listdir(input_dir):
    folder_path = os.path.join(input_dir, folder_name)

    if os.path.isdir(folder_path):
        # Create the same subfolder in the output directory if it doesn't exist
        save_folder = os.path.join(output_dir, folder_name)
        os.makedirs(save_folder, exist_ok=True)

        # Loop through each image in the subfolder
        for file_name in os.listdir(folder_path):
            file_path = os.path.join(folder_path, file_name)

            if file_name.lower().endswith(('jpg', 'jpeg', 'png')):  # Check for valid image files
                img = load_img(file_path)
                x = img_to_array(img)
                x = x.reshape((1,) + x.shape)

                # Generate augmented images and save them
                i = 0
                for batch in datagen.flow(x, batch_size=1,
                                          save_to_dir=save_folder,
                                          save_prefix=f"{folder_name}_aug",
                                          save_format='jpeg'):
                    i += 1
                    if i > 3:  # Change this number to generate more or fewer augmentations per image
                        break

print("Augmentation complete.")
```

**Figure 7 Augmentation data code**

We can change i>3 to get more or less images as per the requirements.

## 3.4 Final Code

In this section, we will start making and processing the model by loading the images first in required variables and then go on with all the implementations and training.

In this code what we did is loaded the data in the variables train_dir and val_dir to process the data. We made the model in the same folder where our data is you can change path as per your data folder. Then we used augmentation on our training set for better training and kept valid set as it is. We later gave the batch size which is 64 but we can use (16, 32, 64 or 128). The size of the images is kept at 224 x 224 but change as per the model requirements. Class mode is categorical because we are performing multi class classification.
Below figure 8 shows the code

```
from tensorflow.keras.applications import InceptionV3
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras import layers, models, optimizers

# Define directory paths for training and validation data
train_dir = "train"
val_dir = "valid"

# Data preprocessing and augmentation for training and validation datasets
train_datagen = ImageDataGenerator(
    rescale=1.0 / 255.0,
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode="nearest",
)
val_datagen = ImageDataGenerator(rescale=1.0 / 255.0)

# Load datasets
train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(224, 224),
    batch_size=64,
    class_mode="categorical",
)
val_generator = val_datagen.flow_from_directory(
    val_dir,
    target_size=(224, 224),
    batch_size=64,
    class_mode="categorical",
)

# Number of classes for the output layer
num_classes = len(train_generator.class_indices)
```

**Figure 8 Data loading**

In the below code, we started with loading the model. We loaded the pre-trained model of InceptionV3 with pre trained weights using ImageNet. In the next step we froze the base layer of the pre-trained model to retain trained features. Then we added the custom layers to the model for fine tune the unique features of the model. We used dropout layers to avoid overfitting and Dense layer using softmax activation for multi-class classification.

Later, we complied the model with Adam optimizer, loss as categorical crossentropy and metrics of accuracy to show the training process. Then, we moved to training the model with epochs 40 for better training and saved the model as .h5 file format.

In the end, we have written code to get the line charts for training and validation accuracy/loss for better understanding using matplotlib.pyplot. We can see in the **Figure 9** below.

```python
# Load the pre-trained InceptionV3 model
base_model = InceptionV3(weights="imagenet", include_top=False, input_shape=(224, 224, 3))

# Freeze the base model to retain pre-trained features
base_model.trainable = False

# Add custom classification layers
model = models.Sequential([
    base_model,
    layers.GlobalAveragePooling2D(),
    layers.Dense(1024, activation="relu"),
    layers.Dropout(0.5),
    layers.Dense(num_classes, activation="softmax"),
])

# Compile the model
model.compile(
    optimizer=optimizers.Adam(learning_rate=0.001),
    loss="categorical_crossentropy",
    metrics=["accuracy"],
)

# Train the model
history = model.fit(
    train_generator,
    steps_per_epoch=train_generator.samples // train_generator.batch_size,
    epochs=40,
    validation_data=val_generator,
    validation_steps=val_generator.samples // val_generator.batch_size,
)

# Save the model for future use
model.save("inception_v3_model.h5")

# Optionally, plot the training history
import matplotlib.pyplot as plt

# Plot training and validation accuracy
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.title('Training and Validation Accuracy')
plt.show()

# Plot training and validation loss
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.title('Training and Validation Loss')
plt.show()
```

**Figure 9 Model loading and training**

```
312/312 ━━━━━━━━━━━━━━━━━━━━ 2670s 9s/step - accuracy: 0.8066 - loss: 0.6072 - val_accuracy: 0.9119 - val_loss: 0.2776
Epoch 34/40
312/312 ━━━━━━━━━━━━━━━━━━━━ 7s 6ms/step - accuracy: 0.7656 - loss: 0.7670 - val_accuracy: 1.0000 - val_loss: 0.2078
Epoch 35/40
312/312 ━━━━━━━━━━━━━━━━━━━━ 2733s 9s/step - accuracy: 0.8030 - loss: 0.6149 - val_accuracy: 0.9058 - val_loss: 0.2859
Epoch 36/40
312/312 ━━━━━━━━━━━━━━━━━━━━ 6s 6ms/step - accuracy: 0.7969 - loss: 0.6712 - val_accuracy: 0.8750 - val_loss: 0.3442
Epoch 37/40
312/312 ━━━━━━━━━━━━━━━━━━━━ 2703s 9s/step - accuracy: 0.8035 - loss: 0.6145 - val_accuracy: 0.9109 - val_loss: 0.2711
Epoch 38/40
312/312 ━━━━━━━━━━━━━━━━━━━━ 6s 6ms/step - accuracy: 0.7969 - loss: 0.6569 - val_accuracy: 0.9167 - val_loss: 0.2162
Epoch 39/40
312/312 ━━━━━━━━━━━━━━━━━━━━ 2518s 8s/step - accuracy: 0.8079 - loss: 0.6147 - val_accuracy: 0.9121 - val_loss: 0.2689
Epoch 40/40
312/312 ━━━━━━━━━━━━━━━━━━━━ 6s 6ms/step - accuracy: 0.8750 - loss: 0.4082 - val_accuracy: 0.7917 - val_loss: 0.8179
```

**Figure 10 Training process**

In figure 10, we can see the training process of the above model.

```python
import numpy as np
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from sklearn.metrics import confusion_matrix, classification_report

# Directory paths
val_dir = "valid"

# Load the saved model
model_path = "inception_v3_model.h5"
model = load_model(model_path)

# Validation data preprocessing
val_datagen = ImageDataGenerator(rescale=1.0 / 255.0)
val_generator = val_datagen.flow_from_directory(
    val_dir,
    target_size=(224, 224),
    batch_size=64,
    class_mode="categorical",
    shuffle=False  # Ensures data order for confusion matrix computation
)

# Generate predictions on the validation dataset
val_generator.reset()
predictions = model.predict(val_generator, steps=val_generator.samples // val_generator.batch_size + 1)
predicted_classes = np.argmax(predictions, axis=1)

# Get true classes
true_classes = val_generator.classes

# Compute the confusion matrix
cm = confusion_matrix(true_classes, predicted_classes)
print("Confusion Matrix:\n", cm)

# Classification report
target_names = list(val_generator.class_indices.keys())
print("\nClassification Report:\n", classification_report(true_classes, predicted_classes, target_names=target_names))
```

**Figure 11 Graphs and charts**

In the figure 11, we can see the code for making confusion matrix and classification report for the above model.

```
model_path = "inception_v3_model.h5"
model = load_model(model_path)

# Define the labels (must match the labels used during training)
labels = list(val_generator.class_indices.keys())  # Assuming val_generator is defined as in the previous code

# Load and preprocess the image
def load_and_preprocess_image(img_path):
    img = image.load_img(img_path, target_size=(224, 224))  # Resize to match model input size
    img_array = image.img_to_array(img)  # Convert the image to a numpy array
    img_array = np.expand_dims(img_array, axis=0)  # Add an extra dimension for batch (1, height, width, channels)
    img_array /= 255.0  # Normalize the image
    return img_array

# Predict the label of an image
def predict_image_label(img_path):
    img = load_and_preprocess_image(img_path)
    prediction = model.predict(img)  # Get model predictions
    predicted_class = np.argmax(prediction, axis=1)  # Get the index of the highest probability
    predicted_label = labels[predicted_class[0]]  # Map index to class label

    # Display the image along with the prediction
    plt.imshow(image.load_img(img_path))
    plt.title(f"Predicted Label: {predicted_label}")
    plt.axis('off')
    plt.show()

# Test the function
img_path = "cigarettebutt_cigarettebutt (38).jpg"
img_path1 = "battery_battery (136).jpg"
img_path2 = "bulb_bulb (124).jpg"
img_path3 = "cardboard_carboard (250).jpg"
img_path4 = "Crushed bottles_crushed_bottle (24).jpeg"
img_path5 = "diapers_diaper (20).jpg"
img_path6 = "Greased pizza boxes_greased_pizza_box (3).jpg"
img_path7 = "Medical waste_medical_waste (205).jpg"
img_path8 = "paper_paper (269).jpg"
img_path9 = "Plastic Cup_plastic_cups (63).jpg"
img_path10 = "Used napkins_used_napkins (7).jpeg"
predict_image_label(img_path)
predict_image_label(img_path1)
predict_image_label(img_path2)
predict_image_label(img_path3)
predict_image_label(img_path4)
predict_image_label(img_path5)
predict_image_label(img_path6)
predict_image_label(img_path7)
predict_image_label(img_path8)
predict_image_label(img_path9)
predict_image_label(img_path10)
```

**Figure 12 Model testing**

Above in Figure 12, we can see the code for testing the model by input of images to test the model predictions of the image classes.

# 4 References

Hasan, M.K., Atta, A., Khan, M.A., Akram, A.S., Issa, G.F. Hassan, M., 2022. Smart Waste Management and Classification System for Smart Cities using Deep Learning. Proceedings of the 2022 International Conference on Business Analytics for Technology and Security (ICBATS), pp. 1–6.

Ma, X., Li, Z. Zhang, L., 2022. An Improved ResNet-50 for Garbage Image Classifi cation. Technical Gazette, 29(5), pp.1552–1559.

Mao, W.-L., Chen, W.-C., Fathurrahman, H.I.K. Lin, Y.-H., 2022. Deep learning networks for real-time regional domestic waste detection. Journal of Cleaner Production, 344, p.131096.

Sakr, G.E., Mokbel, M., Darwich, A., Khneisser, M.N., Hadi, A., 2016. Compar ing Deep Learning and Support Vector Machines for Autonomous Waste Sorting. Pro ceedings of the 2016 IEEE International Multidisciplinary Conference on Engineering Technology (IMCET).
Trivedi, N.K., Tiwari,