

**Title: Waste Classification Using Convolutional Neural Networks (CNN)**

**Submitted by:** Rohan Kumar Jha (22112025)  
**Date:** 10/11/2024

**1. Introduction and Objectives**

Efficient waste classification is crucial for recycling and waste management. The aim of this project is to develop a Convolutional Neural Network (CNN) that can classify waste images as either *Organic* or *Recyclable*. CNNs are well-suited for image classification due to their ability to automatically learn spatial hierarchies of features, which makes them ideal for distinguishing between different types of waste.

**2. Dataset Processing and Description**

**Dataset and Preprocessing**

The dataset is organized into training and testing directories:

* **Training Directory**: Contains labeled images of both classes—*Organic* and *Recyclable*.
* **Test Directory**: Contains images to evaluate the model’s accuracy on unseen data.

We use ImageDataGenerator for data augmentation to increase dataset variability and improve model robustness. The code snippet below loads and prepares the data:

python

Copy code

import os

import numpy as np

import pandas as pd

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout

# Directory paths

train\_dir = 'C:/Users/asus/Downloads/waste\_classification/DATASET/DATASET/TRAIN'

test\_dir = 'C:/Users/asus/Downloads/waste\_classification/DATASET/DATASET/TEST'

# Data augmentation for image preprocessing

datagen = ImageDataGenerator(rescale=1.0/255.0, validation\_split=0.2)

train\_generator = datagen.flow\_from\_directory(train\_dir, target\_size=(128, 128), batch\_size=32, class\_mode='binary', subset='training')

validation\_generator = datagen.flow\_from\_directory(train\_dir, target\_size=(128, 128), batch\_size=32, class\_mode='binary', subset='validation')

* **Explanation**: ImageDataGenerator resizes, normalizes (scales pixel values to [0,1]), and augments the images. The training data is split into 80% training and 20% validation.
* **Justification**: Data augmentation increases dataset variety and helps reduce overfitting, making the model more generalizable.

**Class Distribution Visualization**

A pie chart displays the distribution of *Organic* and *Recyclable* waste images in the dataset, helping identify any class imbalance:

# Example label counts (replace with actual values from your dataset)

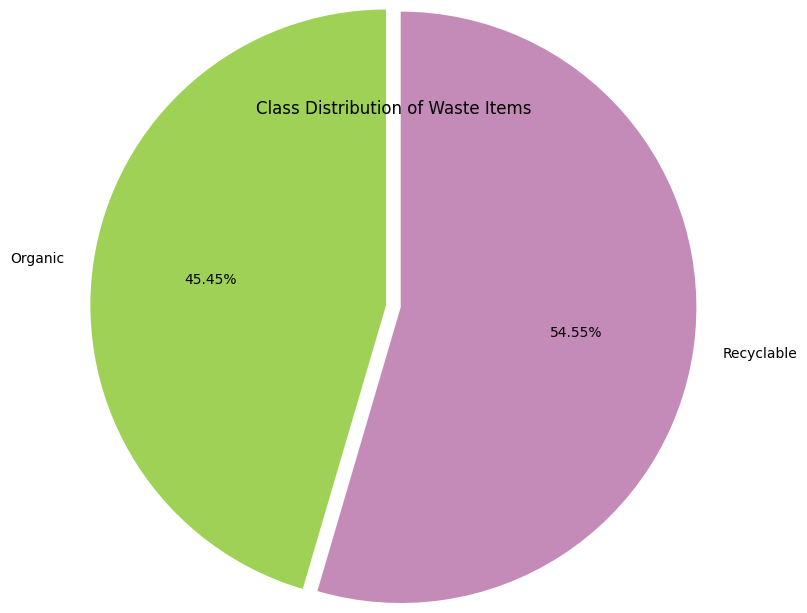
label\_counts = [1000, 1200] # Replace with actual counts of 'Organic' and 'Recyclable' labels

# Plotting the pie chart

plt.pie(label\_counts, startangle=90, explode=[0.05, 0.05], autopct='%0.2f%%', labels=['Organic', 'Recyclable'], colors=['#a0d157', '#c48bb8'], radius=2)

plt.title("Class Distribution of Waste Items")

plt.show()



**3. Model Description**

The CNN model architecture consists of convolutional, pooling, and fully connected layers:

cnn\_model = Sequential([

Conv2D(32, (3, 3), activation='relu', input\_shape=(128, 128, 3)),

MaxPooling2D((2, 2)),

Conv2D(64, (3, 3), activation='relu'),

MaxPooling2D((2, 2)),

Conv2D(128, (3, 3), activation='relu'),

MaxPooling2D((2, 2)),

Flatten(),

Dense(128, activation='relu'),

Dropout(0.5),

Dense(1, activation='sigmoid')

])

# Compile the model

cnn\_model.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])

* **Explanation**:
  + **Convolutional Layers**: Conv2D layers capture spatial features from images, while MaxPooling2D reduces the spatial dimensions to avoid overfitting.
  + **Flatten Layer**: Converts the 2D feature maps into a 1D vector.
  + **Fully Connected Layers**: Dense layers with relu activation classify the image.
  + **Dropout**: Prevents overfitting by randomly dropping 50% of nodes during training.
  + **Output Layer**: A single node with sigmoid activation for binary classification.
* **Justification**: This architecture is designed to balance complexity with efficiency, ideal for image classification.

**4. Model Implementation, Training, and Optimization**

**Training the Model**

The CNN is trained for 5 epochs with the following code:

history = cnn\_model.fit(train\_generator, epochs=5, validation\_data=validation\_generator, verbose=1)

* **Explanation**:
  + The model learns from the train\_generator and is validated on the validation\_generator.
  + Training proceeds through 5 epochs, where each epoch completes a pass over the entire dataset.
* **Justification**: Training accuracy and validation accuracy are tracked, which helps identify overfitting if validation accuracy plateaus or declines.

**Optimization Techniques**

* **Data Augmentation**: Generates variations of images to prevent overfitting.
* **Dropout Layer**: Regularizes the model by dropping nodes, enhancing generalization.
* **Adam Optimizer**: Used for adaptive learning rate, which improves convergence speed and performance.

**5. Results and Evaluation**

**Model Evaluation**

The test accuracy is calculated using unseen data:

test\_generator = datagen.flow\_from\_directory(test\_dir, target\_size=(128, 128), batch\_size=32, class\_mode='binary')

cnn\_loss, cnn\_accuracy = cnn\_model.evaluate(test\_generator)

print(f"CNN Model - Accuracy: {cnn\_accuracy \* 100:.2f}%")

* **Explanation**: The evaluate function computes loss and accuracy on test data.
* **Interpretation**: Test accuracy reflects the model’s generalization ability, which is crucial for real-world applications.

**Training and Validation Accuracy and Loss Visualization**

Accuracy and loss plots provide insights into the model’s learning behavior:

1. **Accuracy Plot**:

plt.figure(figsize=[10,6])

plt.plot(history.history["accuracy"], label="Training Accuracy", color='blue')

plt.plot(history.history["val\_accuracy"], label="Validation Accuracy", color='orange')

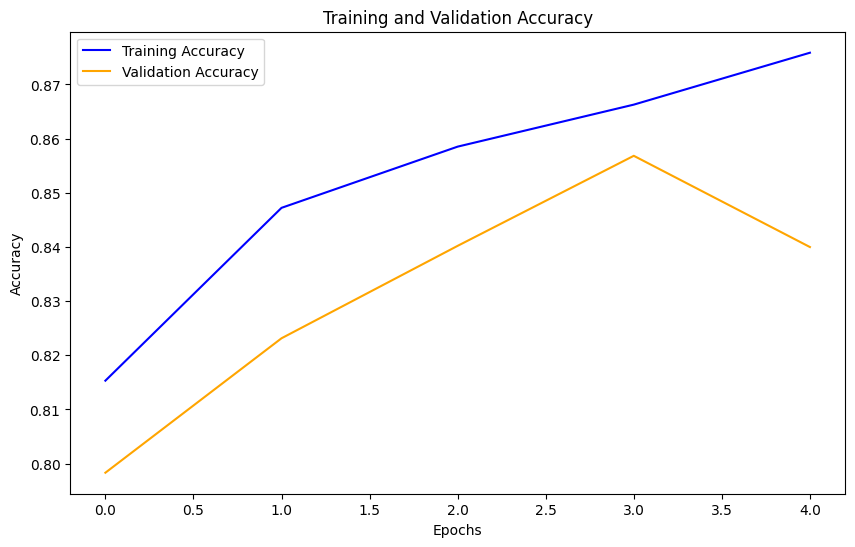
plt.title("Training and Validation Accuracy")

plt.xlabel("Epochs")

plt.ylabel("Accuracy")

plt.legend()

plt.show()



* + **Interpretation**: Ideally, training and validation accuracies should converge to similar values. Large gaps indicate potential overfitting.

1. **Loss Plot**:

plt.figure(figsize=(10, 6))

plt.plot(history.history['loss'], label="Training Loss", color='red')

plt.plot(history.history['val\_loss'], label="Validation Loss", color='green')

plt.title("Training and Validation Loss")

plt.xlabel("Epochs")

plt.ylabel("Loss")

plt.legend()

plt.show()



* + **Interpretation**: Decreasing loss indicates effective learning. A divergence between training and validation loss may signal overfitting.

**Prediction Example**

A sample image is classified by the trained model:

import cv2

import matplotlib.pyplot as plt

import numpy as np

# Load and preprocess the image

test\_image\_path = 'C:/Users/asus/Downloads/waste\_classification/DATASET/DATASET/TEST/O/O\_13015.jpg'

preprocessed\_image = preprocess\_image(test\_image\_path)

prediction = cnn\_model.predict(preprocessed\_image)

# Interpret and display result

if prediction[0][0] > 0.5:

print("The waste item is classified as: Recyclable")

else:

print("The waste item is classified as: Organic")

plt.figure(figsize=(5,5))

plt.imshow(cv2.cvtColor(cv2.imread(test\_image\_path), cv2.COLOR\_BGR2RGB))

plt.axis('off')

plt.show()

Output-

1/1 [==============================] - 0s 114ms/step

The waste item is classified as: Organic



* **Explanation**: The model outputs a probability for *Recyclable*; if this value is above 0.5, the item is classified as *Recyclable*, otherwise *Organic*.
* **Justification**: Testing with a real example demonstrates the model’s practical application.