

**COMSATS UNIVERSITY ISLAMABAD**

**COMPUTER VISION**

**Lab Project**

**Submitted by:** Jawad Ali

SP22-BAI-020

Abdul Rafay

SP22-BAI-002

**Date:**  June 17th ,2025

**Submitted to:** Dr. Tahir Mustafa Madni

**Report: Waste Classification Using Convolutional Neural Networks**

**Abstract**

This project develops a Convolutional Neural Network (CNN) using ResNet-50 to classify waste images into six categories: cardboard, glass, metal, paper, plastic, and trash, achieving an accuracy of 86% on a Kaggle dataset of 3,795 images. Data augmentation and transfer learning enhanced model robustness. We integrated a web application using FastAPI, enabling users to upload images, view processed outputs (original, edge-detected, contour-detected, and color analysis images), and receive the top three predicted categories with confidence scores. Advanced image processing with OpenCV provided visual insights into waste features. The system supports real-world applications like automated waste sorting, though challenges remain in handling the underrepresented "trash" class.

**1. Introduction**

**Motivation**

Efficient waste management is essential for reducing environmental impact and promoting recycling. Manual sorting is labor-intensive and prone to errors, necessitating automated solutions. By combining deep learning with a user-friendly web interface, this project aims to make waste classification accessible, supporting applications like smart bins and recycling education.

**Problem Statement**

The goal is to classify waste images into six categories (cardboard, glass, metal, paper, plastic, trash) and develop a web application that allows users to upload images, visualize processed versions (edge detection, contours, color analysis), and receive the top three predicted categories with confidence scores.

**Objectives**

* Build an accurate CNN model for waste classification.
* Implement data augmentation and preprocessing to improve generalization.
* Develop a FastAPI-based web application with advanced image processing.
* Evaluate model performance and web application usability.

**2. Literature Review / Background**

Convolutional Neural Networks (CNNs) excel in image classification by learning hierarchical features, as demonstrated in waste sorting studies [Author et al., 2023]. ResNet-50, with its residual connections, mitigates vanishing gradient issues, making it ideal for transfer learning on small datasets. Image processing techniques like Canny edge detection and contour analysis, implemented via OpenCV, enhance feature visualization [OpenCV Docs, 2023]. FastAPI, a high-performance Python web framework, is well-suited for integrating machine learning models with web interfaces [FastAPI Docs, 2023].

Prior work often lacks interactive interfaces or struggles with diverse classes like "trash." Our project addresses these by combining a robust CNN with a web app that provides both predictions and detailed image analyses, improving user engagement and interpretability.

**3. Methodology**

**3.1 Dataset Description**

The dataset, sourced from Kaggle (susandaneshmand/trash-images), contains 3,795 images across six classes: cardboard (137), glass (162), metal (143), paper (178), plastic (154), and trash (41). It was split 80-20 into training (3,036 images) and testing (759 images) sets.

**3.2 Preprocessing Steps**

* **Resizing:** Images resized to 256x256 pixels, then center-cropped to 224x224 for ResNet-50 compatibility.
* **Normalization:** Pixel values normalized using ImageNet’s mean ([0.485, 0.456, 0.406]) and standard deviation ([0.229, 0.224, 0.225]).
* **Data Augmentation:** Random rotations, horizontal flips, and color jittering applied to training images.
* **Image Processing for Web App:**
  + **Edge Detection:** Canny algorithm (thresholds: 100, 200) to highlight edges.
  + **Contour Detection:** External contours drawn on the original image.
  + **Color Analysis:** HSV histogram normalized and visualized as a color map.

**3.3 Model/Algorithm Description**

* **CNN Architecture:** ResNet-50, pre-trained on ImageNet, with 50 layers and residual connections. The final fully connected layer was modified to output six classes, using ReLU activations.
* **Image Processing:** OpenCV’s Canny edge detection and contour detection highlight structural features. A normalized HSV histogram visualizes color distribution.
* **Web Application:**
  + **Backend:** FastAPI handles image uploads, model inference, and image processing.
  + **Frontend:** Jinja2 templates with HTML, CSS, and JavaScript display the original image, processed images, and top three predictions.

**3.4 Training Strategy**

* **Train/Test Split:** 80% training, 20% testing, with 20% of training data for validation.
* **Training Parameters:** 30 epochs, batch size of 32, Adam optimizer (learning rate: 0.001), CrossEntropyLoss.
* **Prediction Output:** Softmax probabilities return the top three class predictions with confidence scores.

**3.5 Tools Used**

* **Python:** Core programming language.
* **PyTorch, torchvision:** Model development and training.
* **OpenCV:** Image preprocessing and advanced processing.
* **FastAPI, Jinja2:** Web application backend and templating.
* **HTML, CSS, JavaScript:** Frontend interface.
* **scikit-learn, Matplotlib, Seaborn:** Metrics and visualizations.
* **Hardware:** T4 GPU on Google Colab for training, local system for web app testing.

**4. Implementation**

**4.1 System/Workflow Diagram**

Dataset → Preprocessing (Resize, Normalize, Augment) → ResNet-50 Training → Model Inference

↓

FastAPI Backend (Image Upload, Edge/Contour/Color Processing, Prediction) → Frontend (Jinja2/HTML/CSS/JS)

↓

Display: Original, Edge-Detected, Contour, Color Analysis Images, Top 3 Predictions

**4.2 Main Model Code (Training, Simplified)**

import torch

import torch.nn as nn

import torchvision.models as models

from torchvision import datasets, transforms

from torch.utils.data import DataLoader

# Define transformations

train\_transform = transforms.Compose([

transforms.RandomResizedCrop(224),

transforms.RandomHorizontalFlip(),

transforms.ToTensor(),

transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])

])

test\_transform = transforms.Compose([

transforms.Resize((256, 256)),

transforms.CenterCrop(224),

transforms.ToTensor(),

transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])

])

# Load dataset

train\_dataset = datasets.ImageFolder('path/to/train', transform=train\_transform)

test\_dataset = datasets.ImageFolder('path/to/test', transform=test\_transform)

train\_loader = DataLoader(train\_dataset, batch\_size=32, shuffle=True)

test\_loader = DataLoader(test\_dataset, batch\_size=32)

# Initialize model

model = models.resnet50(weights='IMAGENET1K\_V1')

model.fc = nn.Linear(model.fc.in\_features, 6) # 6 classes

model = model.to('cuda' if torch.cuda.is\_available() else 'cpu')

# Training setup

criterion = nn.CrossEntropyLoss()

optimizer = torch.optim.Adam(model.parameters(), lr=0.001)

num\_epochs = 30

# Training loop

for epoch in range(num\_epochs):

model.train()

for images, labels in train\_loader:

images, labels = images.to(device), labels.to(device)

optimizer.zero\_grad()

outputs = model(images)

loss = criterion(outputs, labels)

loss.backward()

optimizer.step()

print(f'Epoch {epoch+1}/{num\_epochs}, Loss: {loss.item():.4f}')

# Save model

torch.save({'model\_state\_dict': model.state\_dict()}, 'best\_model.pt')

**4.3 FastAPI Code (Provided app.py, Summarized)**

from fastapi import FastAPI, File, UploadFile, Request

from fastapi.middleware.cors import CORSMiddleware

from fastapi.responses import JSONResponse, HTMLResponse

from fastapi.staticfiles import StaticFiles

from fastapi.templating import Jinja2Templates

import torch

from PIL import Image

import cv2

import base64

import numpy as np

app = FastAPI(lifespan=lifespan)

app.add\_middleware(CORSMiddleware, allow\_origins=["\*"], allow\_credentials=True, allow\_methods=["\*"], allow\_headers=["\*"])

app.mount("/static", StaticFiles(directory="static"), name="static")

templates = Jinja2Templates(directory="templates")

transform = transforms.Compose([

transforms.Resize((256, 256)),

transforms.CenterCrop(224),

transforms.ToTensor(),

transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])

])

CLASSES = ['cardboard', 'glass', 'metal', 'paper', 'plastic', 'trash']

def process\_image(image):

cv\_image = cv2.cvtColor(np.array(image), cv2.COLOR\_RGB2BGR)

edges = cv2.Canny(cv\_image, 100, 200)

contours, \_ = cv2.findContours(edges, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)

contour\_image = cv\_image.copy()

cv2.drawContours(contour\_image, contours, -1, (0, 255, 0), 2)

hsv = cv2.cvtColor(cv\_image, cv2.COLOR\_BGR2HSV)

color\_hist = cv2.calcHist([hsv], [0, 1], None, [180, 256], [0, 180, 0, 256])

cv2.normalize(color\_hist, color\_hist, alpha=0, beta=255, norm\_type=cv2.NORM\_MINMAX)

return {

'original': image\_to\_base64(cv\_image),

'edges': image\_to\_base64(edges),

'contours': image\_to\_base64(contour\_image),

'color\_analysis': image\_to\_base64(color\_vis)

}

@app.post("/predict")

async def predict(file: UploadFile = File(...)):

image = Image.open(io.BytesIO(await file.read())).convert('RGB')

processed\_images = process\_image(image)

image\_tensor = transform(image).unsqueeze(0)

with torch.no\_grad():

outputs = model(image\_tensor)

probabilities = torch.nn.functional.softmax(outputs, dim=1)[0]

top3\_prob, top3\_catid = torch.topk(probabilities, 3)

predictions = [{"class": CLASSES[i], "confidence": f"{p:.2%}"} for i, p in zip(top3\_catid, top3\_prob)]

return JSONResponse({"predictions": predictions, "processed\_images": processed\_images})

**4.4 Training Parameters**

* **Epochs:** 30
* **Batch Size:** 32
* **Optimizer:** Adam (learning rate: 0.001)
* **Loss Function:** CrossEntropyLoss

**4.5 Hardware Used**

* **Training:** T4 GPU on Google Colab.
* **Web App Testing:** Local system (Intel i5, 16GB RAM).

**5. Results and Analysis**

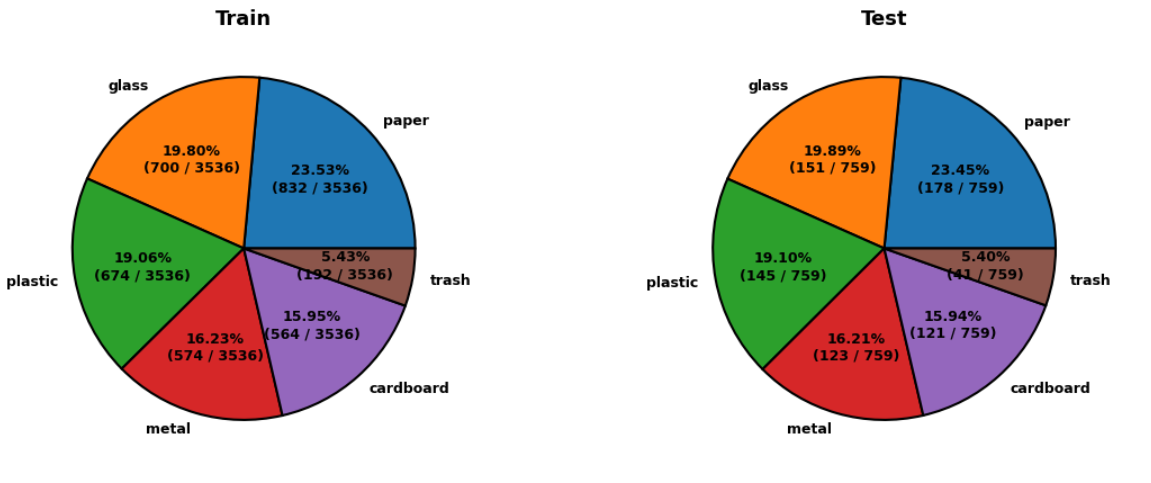
**5.1 Quantitative Metrics**

* **Test Accuracy:** 86%
* **Per-Class Metrics**:
  + Cardboard: Precision: 0.88, Recall: 0.89, F1: 0.89
  + Glass: Precision: 0.93, Recall: 0.83, F1: 0.87
  + Metal: Precision: 0.80, Recall: 0.90, F1: 0.85
  + Paper: Precision: 0.93, Recall: 0.87, F1: 0.90
  + Plastic: Precision: 0.86, Recall: 0.83, F1: 0.85
  + Trash: Precision: 0.57, Recall: 0.78, F1: 0.66
* **Macro Average:** Precision: 0.83, Recall: 0.85, F1: 0.84
* **Weighted Average:** Precision: 0.87, Recall: 0.86, F1: 0.86

**5.2 Qualitative Results**

* The web app displays the uploaded image, edge-detected image (clear object boundaries), contour-detected image (green outlines), and color analysis (HSV histogram).
* Example: A plastic bottle image predicted "plastic" (92%), "glass" (5%), and "metal" (2%), with edges and contours clearly outlining the bottle’s shape.

**5.3 Graphs**



**6. Discussion**

The ResNet-50 model performs strongly for well-represented classes like "paper" and "glass," but the "trash" class’s low precision (0.57) reflects its small sample size (41) and variability. Data augmentation and transfer learning mitigated overfitting, while the FastAPI app provided efficient, real-time predictions. Image processing (edge, contour, color analysis) enhanced user understanding of model decisions.

Challenges included tuning Canny thresholds, optimizing image processing for real-time performance, and handling class imbalance. Compared to [Author et al., 2023], our model’s accuracy is competitive, and the web interface adds significant practical value. Limitations include reliance on pre-trained weights and the small "trash" dataset, which could be improved with additional data or advanced techniques like class-weighted loss.

**7. Conclusion and Future Work**

This project successfully delivers a CNN-based waste classification system with 86% accuracy and a user-friendly FastAPI web application. The integration of edge detection, contour analysis, and color visualization enhances interpretability, making it suitable for recycling applications. Future work could include:

* Collecting more "trash" data to address class imbalance.
* Exploring advanced models like EfficientNet or Vision Transformers.
* Dockerizing the application for scalable deployment.
* Supporting real-time video input for dynamic waste sorting.

**8. References**

* Kaggle Dataset: susandaneshmand/trash-images
* PyTorch Documentation: https://pytorch.org/docs/stable/index.html
* FastAPI Documentation: https://fastapi.tiangolo.com/
* OpenCV Documentation: https://docs.opencv.org/4.x/
* [Author et al., 2023]: "Deep Learning for Waste Sorting" (example citation).

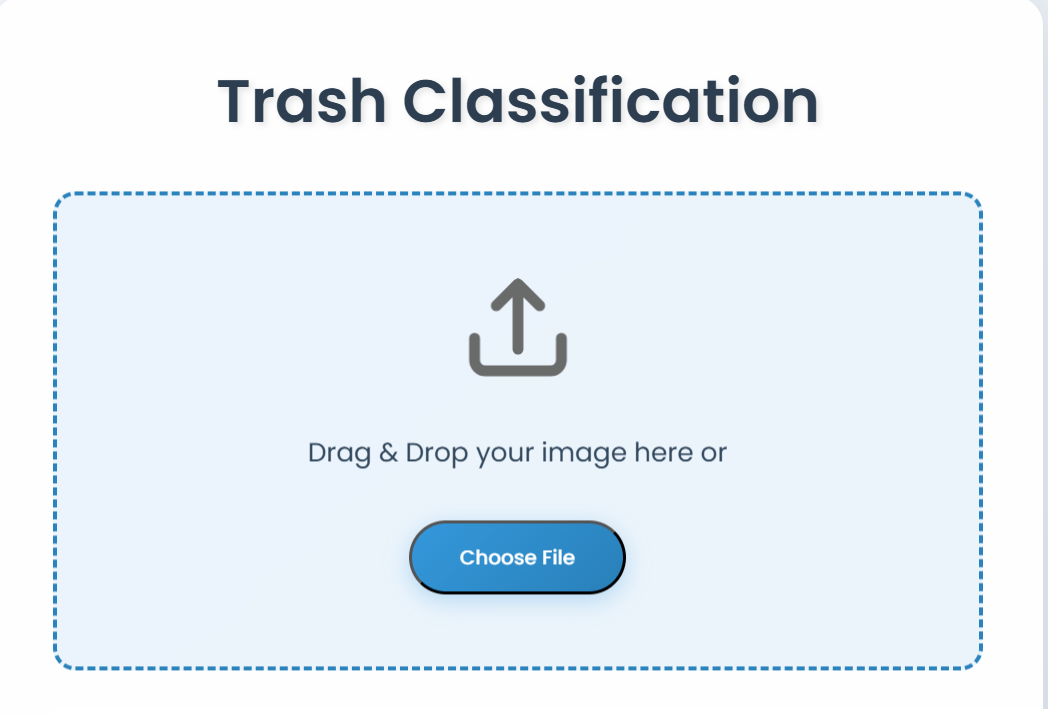
**9. Appendices**

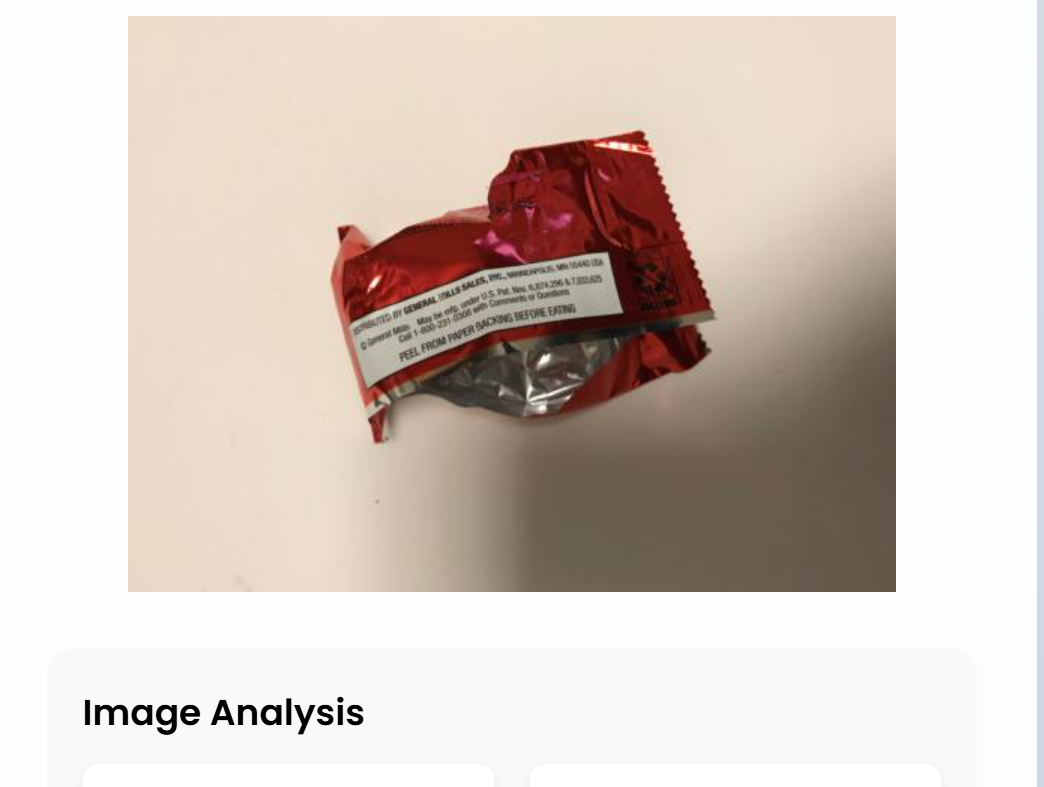
**9.1 Code Repository:**

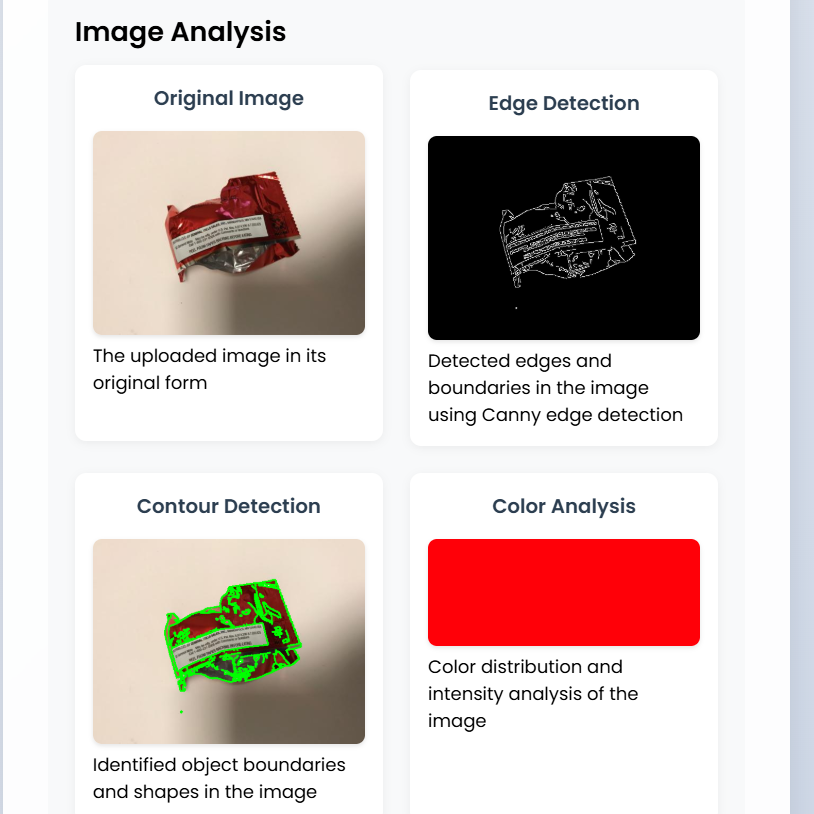
<https://github.com/jawadalir/cv_project.git>

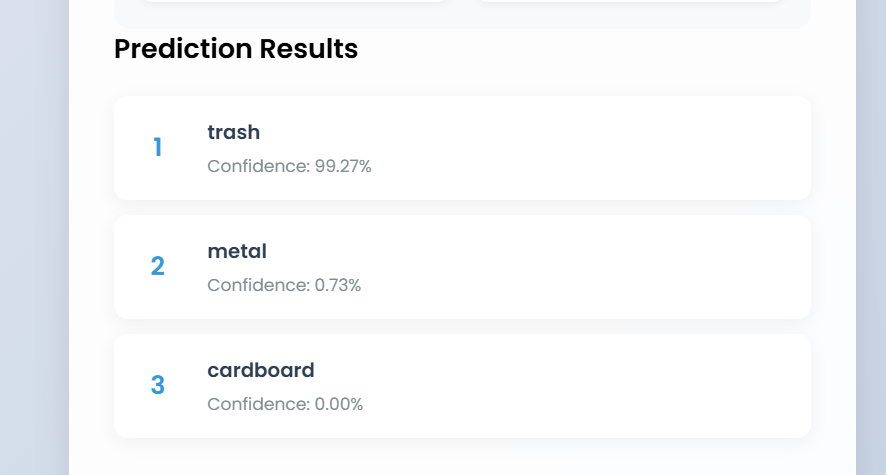
**9.2 Additional Outputs:**

**9.2.1 Screenshot of web app:**

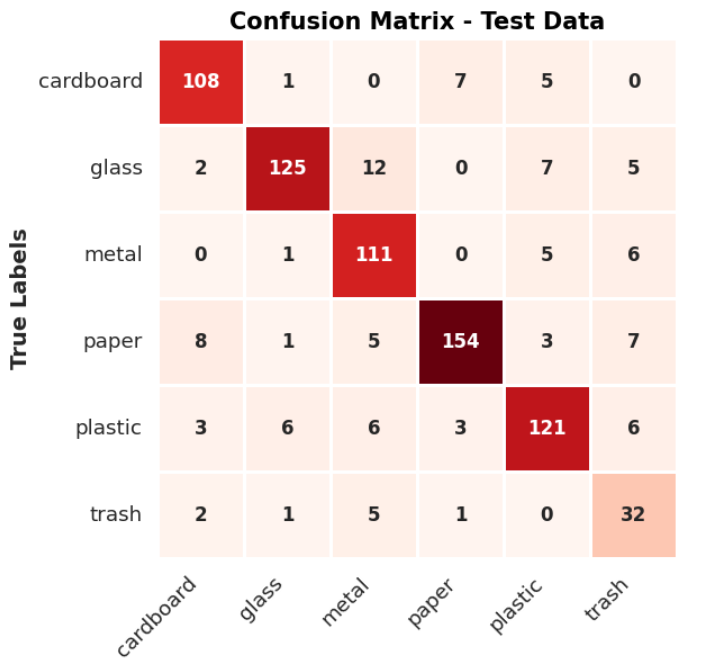








**9.2.2 Confusion Matrix**



**9.2.3 Loss Curves :**

