Configuration Manual

MSc Research Project
MSCAI1 JAN23l – Research in Computing CA1

Rashi Dabhane

Student ID: 21176321

School of Computing
National College of Ireland

Supervisor: Mayank Jain

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Rashi Sunil Dabhane
Student ID:	X21173621
Programme:	MSCAI1 JAN23I – Research in Computing CA1 **Year:** 2023
Module:	MSc Research Project
Lecturer:	Mayank Jain
Submission Due Date:	14/12/2023
Project Title:	Organic and Recyclable waste classification using deep learning methods
Word Count:	1838 **Page Count:** 14

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: Rashi Sunil Dabhane

Date: 14/12/2023

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

Rashi Sunil Dabhane
Student ID: x21173621

Overview

This configuration manual provides detailed instructions for setting up and running the Waste Classification project. The project aims to classify waste images into two categories: Organic and Recyclable.

The script for the task uses PyTorch for deep learning tasks and leverages the VGG16 architecture for image classification.

Scope

The manual covers the entire project lifecycle, including data preparation, model training, evaluation, and result analysis.

Target Audience

This manual is intended for developers, learners, researcher, data scientists, and system administrators, etc.

The primary goals of this research are:

- Automated Waste Sorting
- Environmental Impact
- Educational Outreach
- Model Interpretability

Significance:

The Waste Classification research holds significance in addressing contemporary challenges related to waste management and environmental conservation. The significance is underscored by:

- Efficiency in Waste Management
- Reduced Environmental Footprint
- Technological Innovation for Sustainability

User Expectations:

By following the instructions provided in this manual, users can expect to achieve the following outcomes:

1. **Model Training and Evaluation:**
   - Understand the process of training a deep learning model for waste classification using a pre-trained VGG16 architecture.
   - Evaluate the model's performance on a test dataset, measuring accuracy, precision, recall, and F1 score.

2. **Visualization and Interpretation:**
   - Visualize the training and validation loss curves, providing insights into the model's learning behavior.

- Interpret and analyze a confusion matrix, gaining a deeper understanding of the model's classification performance.

3. **Application to New Data:**
- Extend the research by applying the trained model to new waste images, allowing users to assess its generalization capabilities.

4. **Interactive Image Upload and Prediction:**
- Interactively upload images for waste classification and receive predictions from the trained model, enhancing user engagement and understanding.

5. **Documentation and Community Support:**
- Access comprehensive documentation addressing common issues and solutions during setup, training, and testing phases.

Uploading Dataset to Google Drive

To efficiently handle the dataset, we upload it to Google Drive.

Mounting Google Drive in Colab

Mounting Google Drive allows the Colab notebook to access files stored in your Google Drive.

```
from google.colab import drive
drive.mount('/content/drive')
```

System Requirements

Hardware

- Adequate GPU resources (Used - NVIDIA GeForce RTX 3050)
- Memory: Minimum 8GB RAM (Used - 16GB)
- Storage: Disk space for dataset and model files (SSD – 1TB)
- System: 64-bit Operating system
- Operating System: Windows 11

Software

- Python 3.8 used for model building and implementation
- Required libraries (install using `pip install -r requirements.txt`):

a. Pillow: Image processing library used for opening, manipulating, and saving various image file formats.
pip install Pillow

b. numpy: Fundamental package for scientific computing with Python, providing support for large, multi-dimensional arrays and matrices, along with mathematical functions to operate on these arrays.
pip install numpy

c. pandas: Data manipulation and analysis library, offering data structures like DataFrame for efficient data handling and manipulation.
pip install pandas

d. matplotlib: 2D plotting library for creating static, animated, and interactive visualizations in Python.
pip install matplotlib

e. scikit-learn: Machine learning library providing simple and efficient tools for data mining and data analysis, including various algorithms for classification, regression, clustering, and more.
pip install scikit-learn

f. reportlab: Library for creating PDF documents, enabling the generation of dynamic, data-driven reports.
pip install reportlab

g. seaborn: Statistical data visualization library based on matplotlib, providing a high-level interface for drawing attractive and informative statistical graphics.
pip install seaborn

h. torch (PyTorch): Deep learning framework for building and training neural networks, offering flexibility and dynamic computation graphs.
pip install torch==1.10.0+cu111 torchvision==0.11.1+cu111 torchaudio==0.10.0+cu111 -f https://download.pytorch.org/whl/cu111.html

i. torchvision: PyTorch library providing utilities for computer vision tasks, including image transformations and pre-trained models.

These libraries collectively facilitate tasks such as image processing, data analysis, machine learning, deep learning, visualization, and report generation in the provided code.

Configuration Settings Loading and Preprocessing Dataset

Function to check if the file is a valid image

```
# 21173621_ Function to check if the file is a valid image
def is_valid_image(filename):
    try:
        # Attempt to open the image file
        img = Image.open(filename)
        # Close the image file
        img.close()
        return True
    except (IOError, UnidentifiedImageError):
        # Handle the case where the file is not a valid image
        return False
```

File Paths
Update the paths in the code to match the location of your dataset:
From the script,

```
# 21173621_ Load the train dataset
train = get_image_paths("/content/Waste dataset/DATASET/TRAIN")
```

```
# 21173621_ Load the test dataset
test = get_image_paths("/content/Waste dataset/DATASET/TEST")

# 21173621_ Convert string labels to binary values
conversion = {'O': 0, 'R': 1}
train.label = train.label.map(conversion)
test.label = test.label.map(conversion)
```

Creating Data Transformation Pipeline
Define the data transformation pipeline for image augmentation:

```
# 21173621_ Data transformation with augmentation
data_transform = transforms.Compose([
    transforms.Resize(256),
    transforms.RandomHorizontalFlip(),
    transforms.RandomRotation(degrees=15),
    transforms.CenterCrop(224),
    transforms.Lambda(lambda img: img.convert("RGB")),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
])
```

Creating Dataset Instances

```
augmented_dataset = WasteData(train, transform=data_transform)
```

Splitting Dataset and Creating Data Loaders

```
# 21173621_ Split train dataset into training and validation sets
train_dataset = WasteData(train, data_transform)
train_size = int(0.9 * len(train_dataset))
valid_size = len(train_dataset) - train_size
train_dataset, valid_dataset = torch.utils.data.random_split(train_dataset, [train_size,
valid_size])

# 21173621_ Test dataset
test_dataset = WasteData(test, data_transform)

# 21173621_ Data loaders
batch_size = 64
train_dataloader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
test_dataloader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False)
valid_dataloader = DataLoader(valid_dataset, batch_size=batch_size, shuffle=False)
```

Defining WasteVGG16 Model

```
# 21173621_ Model
class WasteVGG16(nn.Module):
    def __init__(self):
        super(WasteVGG16, self).__init__()
        self.vgg16 = vgg16(pretrained=True)
        num_ftrs = self.vgg16.classifier[6].in_features
        # 21173621_ Add dropout for regularization
        self.vgg16.classifier[6] = nn.Sequential(
            nn.Dropout(0.5),
            nn.Linear(num_ftrs, 2)  # Change to 2 for two classes
        )

    def forward(self, x):
        x = self.vgg16(x)
        return x
```

Setting up Loss, Optimizer, and Scheduler

```
# 21173621_ Loss and optimizer
criterion = nn.CrossEntropyLoss()

optimizer = torch.optim.Adam(net.parameters(), lr=0.0005, betas=(0.95, 0.9995)) #Adjusted values

# 21173621_ Learning rate scheduler
scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=2, gamma=0.5)
```

Training and Validation of the model

```
# 21173621_ Training loop
train_loss = []
val_loss = []
accuracy_list = []
best_val_loss = float('inf')  # Initialize with a large value
best_epoch = 0

epochs = 20

for epoch in range(epochs):
    epoch_loss = 0.0
    epoch_loss_val = 0.0
    correct = 0
    total = 0
    running_loss = 0.0
    print(f'Epoch {epoch + 1}/{epochs}')
```

```
    # 21173621_ Training
    net.train()
    for i, data in enumerate(train_dataloader, 0):
        inputs, labels = data[0].to(device), data[1].long().to(device)
        optimizer.zero_grad()
        outputs = net(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        running_loss += loss.item()
        if i % 10 == 9:
            loss = running_loss / 10
            epoch_loss += loss
            print(f'\tIteration {i + 1}, Loss: {loss:.6f}')
            running_loss = 0.0

    # 21173621_ Validation
    net.eval()
    with torch.no_grad():
        for i, data in enumerate(valid_dataloader, 0):
            inputs, labels = data[0].to(device), data[1].long().to(device)
            outputs = net(inputs)
            loss = criterion(outputs, labels)
            epoch_loss_val += loss.item()

            # 21173621_ Calculate accuracy
            _, predicted = torch.max(outputs, 1)
            correct += (predicted == labels).sum().item()
            total += labels.size(0)

    accuracy = correct / total
    accuracy_list.append(accuracy)

    # 21173621_ Step the learning rate scheduler
    scheduler.step()
```

Plotting Training and Validation loss curves and accuracy curves

```
# 21173621_ Plot losses
plt.figure(figsize=(20, 6))
sns.lineplot(x=list(range(len(train_loss))), y=train_loss)
sns.lineplot(x=list(range(len(val_loss))), y=val_loss)
plt.legend(['Training loss', 'Validation loss'])
plt.title('Training and Validation Loss Curve')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.savefig('loss_curve.pdf', format='pdf', bbox_inches='tight')
```

```
# 21173621_ Plot accuracy
plt.figure(figsize=(8, 6))
sns.lineplot(x=list(range(len(accuracy_list))), y=accuracy_list)
plt.title('Accuracy Curve')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.savefig('accuracy_curve.pdf', format='pdf', bbox_inches='tight')
```

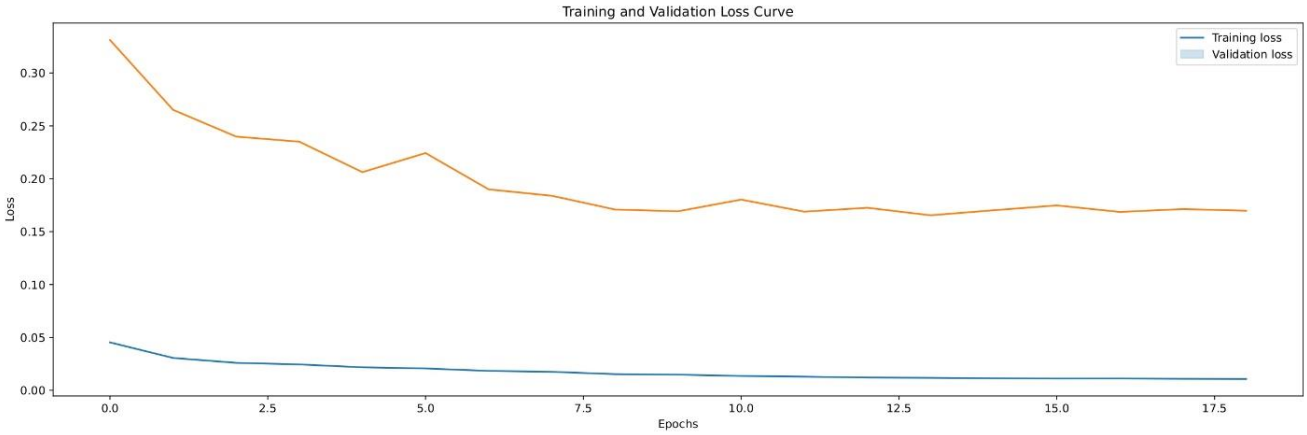

Fig 1. Training and Validation loss - Vgg16 – customized with transfer learning (proposed)

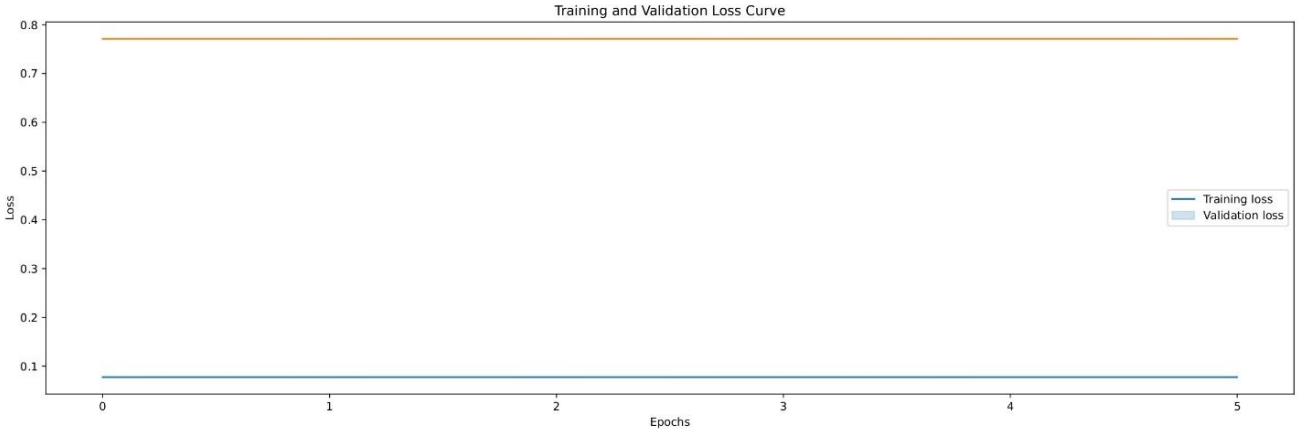

Fig 2. Training and Validation loss - Vgg16 with default values and parameters

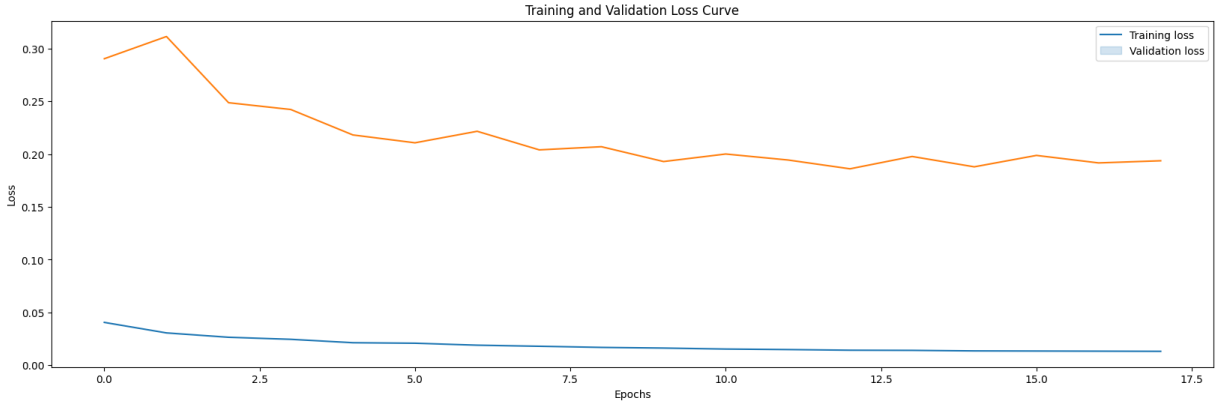

Fig3. Training and Validation loss – ResNet18

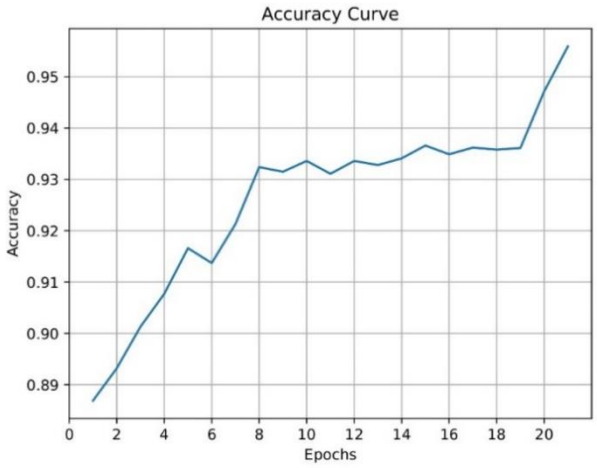

Fig 4. Accuracy curve
Vgg16 – customized with transfer learning

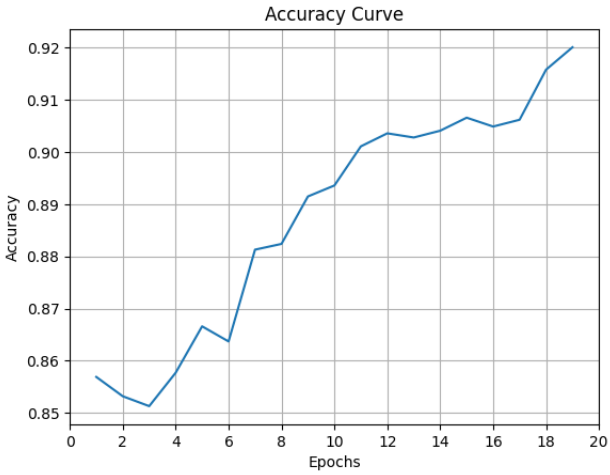

Fig 5. Accuracy curve : ResNet18

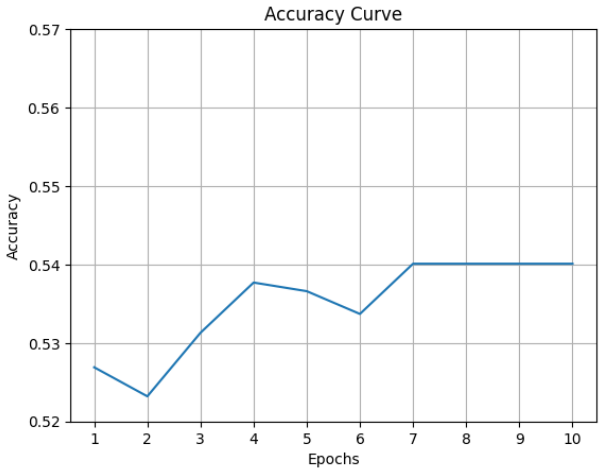

Fig 6. Accuracy curve
Vgg16 with default values and parameters

Testing the model and Metrics Calculation

```
# 21173621_ Testing accuracy
correct = 0
total = 0
net.eval()
with torch.no_grad():
    all_labels = []
    all_preds = []

    for data in test_dataloader:
        images, labels = data[0].to(device), data[1].long().to(device)
        outputs = net(images)
```

```
        _, predicted = torch.max(outputs, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

        # 21173621_ Collect data for metrics
        all_labels.extend(labels.cpu().numpy())
        all_preds.extend(predicted.cpu().numpy())

# 21173621_ Calculate metrics
test_accuracy = 100 * correct / total
test_precision = precision_score(all_labels, all_preds)
test_recall = recall_score(all_labels, all_preds)
test_f1 = f1_score(all_labels, all_preds)
conf_matrix = confusion_matrix(all_labels, all_preds)

# 21173621_ Print metrics
print(f'Test Accuracy: {test_accuracy:.4f}')
print(f'Test Precision: {test_precision:.4f}')
print(f'Test Recall: {test_recall:.4f}')
print(f'Test F1 Score: {test_f1:.4f}')

# 21173621_ Plot confusion matrix for the test set
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['Organic',
'Recyclable'], yticklabels=['Organic', 'Recyclable'])
plt.title('Test Confusion Matrix')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()  # Show the plot

# 21173621_ Save confusion matrix plot as PNG
plt.savefig('conf_matrix.png', format='png', bbox_inches='tight')

# 21173621_ Create a PDF file
with PdfPages('result_summary.pdf') as pdf:
    # 21173621_ Add confusion matrix to PDF
    plt.figure(figsize=(8, 6))
    sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['Organic',
'Recyclable'], yticklabels=['Organic', 'Recyclable'])
    plt.title('Test Confusion Matrix')
    plt.xlabel('Predicted Label')
    plt.ylabel('True Label')
    pdf.savefig()
    plt.close()

    # 21173621_ Attach metrics to the PDF
    pdf.attach_note("Metrics:")
    pdf.attach_note(f'Test Accuracy: {test_accuracy:.4f}')
    pdf.attach_note(f'Test Precision: {test_precision:.4f}')
```

```
    pdf.attach_note(f'Test Recall: {test_recall:.4f}')
    pdf.attach_note(f'Test F1 Score: {test_f1:.4f}')

# 21173621_ Download the PDF file in Colab
files.download('result_summary.pdf')
```

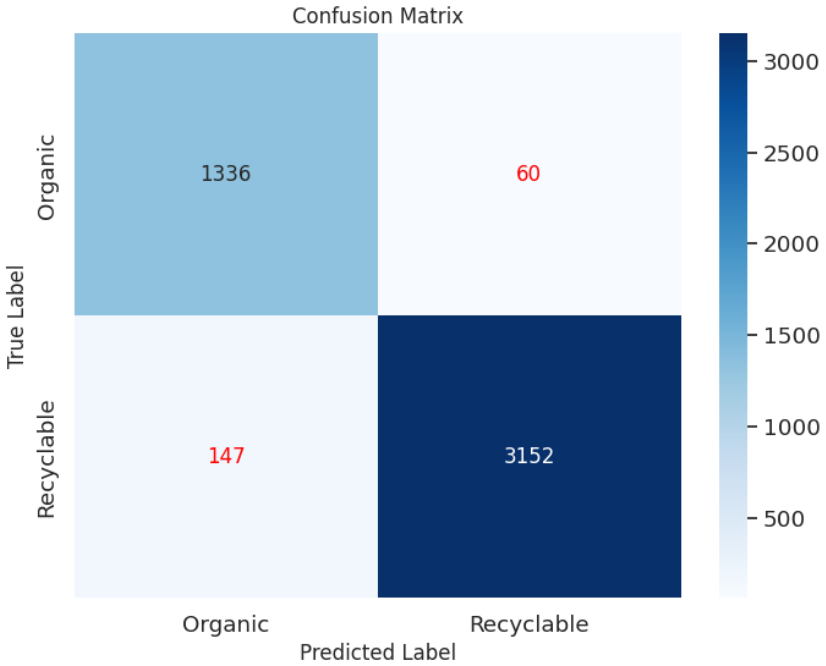

Fig 7. Confusion matrix for Vgg16 – customized with transfer learning (proposed)

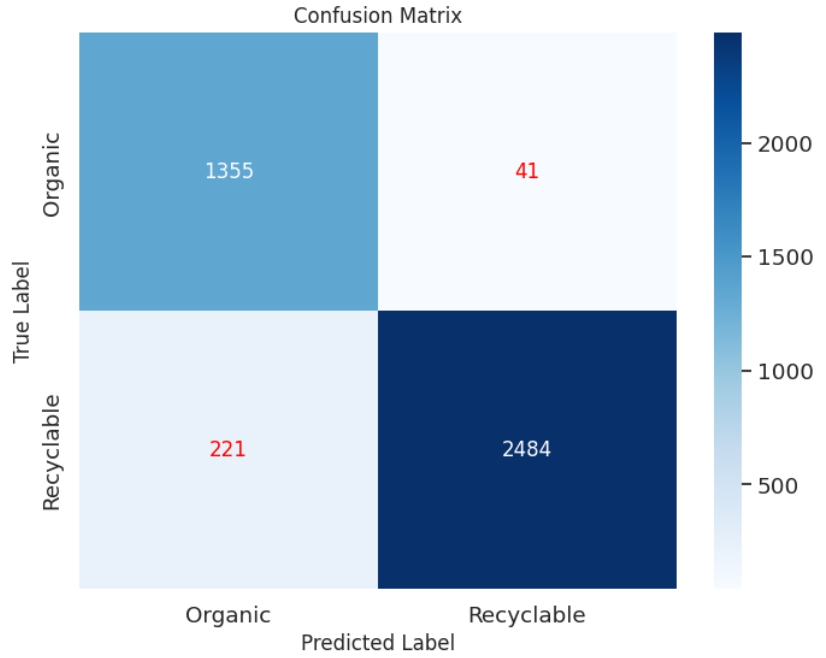

Fig. 8. Confusion matrix for ResNet18

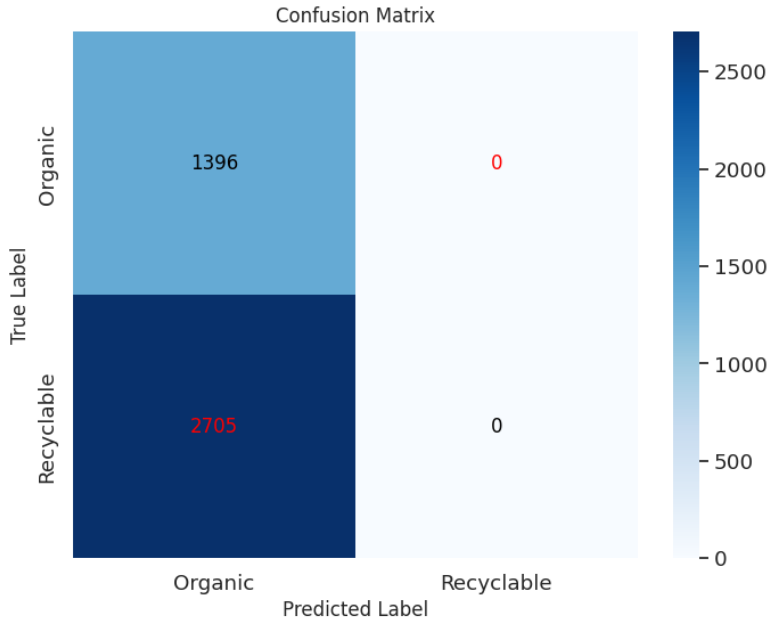

Fig. 9. Confusion matrix for Vgg16 with default values and parameters

File Upload and Prediction

```
# Function to handle file upload and prediction
def handle_upload():
    # Updated: Allow users to upload an image from their local machine
    uploaded = files.upload()

    # Check if any file is uploaded
    if len(uploaded) > 0:
        # Use the first uploaded file
        name, data = list(uploaded.items())[0]

        # Load the image
        img = Image.open(io.BytesIO(data))

        # Get predictions
        predicted_label = predict_single_image(net, img, data_transform)

        # Updated: Provide more descriptive output
        waste_type = "Recyclable" if predicted_label == 1 else "Organic"
        print(f'Predicted Waste Type for User Image: {waste_type}')

        # Display the uploaded image
        plt.imshow(img)
        plt.axis('off')
        plt.title(f'Uploaded Image - Predicted Waste Type: {waste_type}')
        plt.show()

    else:
```

```
        print("No file uploaded.")
# Trigger file upload and prediction
handle_upload()
```