WastDet

Waste Classification using Deep Learning

May 12, 2025

Team Members

- Omar Aitimbet ID: 220103131
- Dinmukhamed Sapybek ID: 220103053

Abstract

With rapid urbanization and industrialization, efficient waste management has become a global challenge. This project investigates the application of deep learning techniques, specifically convolutional neural networks (CNNs), for automating the classification of waste materials based on images. Leveraging a pretrained ResNet50 model, the system is trained on the TrashNet dataset to categorize waste into six distinct classes: cardboard, glass, metal, paper, plastic, and general trash. The model demonstrates strong classification performance and holds significant potential for real-world deployment in smart recycling systems and environmentally sustainable waste management.

1 Introduction

Proper waste segregation plays a critical role in improving recycling efficiency and minimizing environmental harm. However, traditional manual sorting methods are labor-intensive, error-prone, and expose workers to health risks. Recent advances in computer vision and deep learning present an opportunity to automate this process with greater accuracy and consistency. This project aims to develop a CNN-based image classification system capable of identifying different types of waste materials. By training a ResNet50 architecture on labeled waste images, the goal is to support scalable and intelligent waste sorting solutions, contributing to smarter recycling infrastructure and sustainable urban development.

2 Dataset Preprocessing

Effective preprocessing is essential for training accurate and robust deep learning models. In this project, we used the TrashNet dataset, which contains labeled images of six types of

waste materials: `cardboard`, `glass`, `metal`, `paper`, `plastic`, and `trash`. The dataset was organized into separate folders for each class.

2.1 Image Transformations

Before feeding the images into the neural network, we applied a series of preprocessing and data augmentation transformations using `torchvision.transforms`. These transformations were tailored to improve generalization and reduce overfitting:

- **RandomResizedCrop (train only):** Randomly crops a portion of the image and resizes it to 256x256. This helps the model become more robust to scale and positioning variations.
- **RandomRotation (train only):** Applies small random rotations (up to 15 degrees) to simulate real-world camera angles.
- **RandomHorizontalFlip (train only):** Flips images horizontally with a 50% chance, introducing left-right symmetry to the model.
- **CenterCrop:** Crops the image to the central 224x224 region, which is the required input size for ResNet-50.
- **ToTensor:** Converts the image from PIL format to a PyTorch tensor.
- **Normalize:** Normalizes the RGB values using ImageNet's mean and standard deviation: $[0.485, 0.456, 0.406]$ and $[0.229, 0.224, 0.225]$. This step ensures compatibility with the pre-trained ResNet-50 model.

2.2 Data Splitting

To train and evaluate the model effectively, we split the dataset into three subsets:

- **70% for training** – used to learn the model parameters.
- **10% for validation** – used to tune hyperparameters and monitor generalization during training.
- **20% for testing** – used only after training to evaluate final model performance.

The split was implemented using PyTorch's `random_split` function with a fixed seed for reproducibility.

2.3 Subset Assignment and Dataloaders

After splitting, we reloaded each subset (train, validation, test) with the appropriate transformations. Each subset was wrapped with PyTorch's `Subset` class and passed to a `DataLoader`, enabling efficient batch loading and shuffling during training:

- `train_data` was shuffled to improve learning.

- `valid_data` and `test_data` were loaded without shuffling for consistent evaluation.

The batch size was set to 32, balancing training speed and memory usage.

2.4 Class Labels

The dataset's class labels were automatically extracted from folder names using `ImageFolder`. This ensures consistency in label mapping across training and evaluation phases. The classes were:

$$\{\texttt{cardboard, glass, metal, paper, plastic, trash}\}$$

2.5 Summary

In summary, the preprocessing pipeline prepared the dataset for robust training using transfer learning. By applying a combination of augmentation, normalization, and careful splitting, we ensured that the model would be trained on a rich and representative set of inputs, improving its ability to generalize to new waste images.

3 Model Training

In this project, we trained a deep learning model to classify waste images into six categories: `cardboard`, `glass`, `metal`, `paper`, `plastic`, and `trash`. The training process was built around a transfer learning approach using a pre-trained ResNet-50 model.

3.1 Model Setup

We loaded a ResNet-50 model pre-trained on the ImageNet dataset. To leverage its learned features, we froze all convolutional layers to prevent them from updating during training. We then replaced the final fully connected layer with a custom classifier tailored for our six-class problem. The new classifier consists of a linear layer, ReLU activation, dropout for regularization, another linear layer, and finally a LogSoftmax activation function:

```
nn.Sequential(
    nn.Linear(fc_inputs, 256),
    nn.ReLU(),
    nn.Dropout(0.4),
    nn.Linear(256, num_classes),
    nn.LogSoftmax(dim=1)
)
```

3.2 Loss Function and Optimizer

We used the Negative Log Likelihood Loss (`NLLLoss`) as our criterion since the final layer outputs log probabilities. For optimization, we used the Adam optimizer, which adjusts the learning rate adaptively for each parameter, improving convergence speed and performance.

3.3 Training Loop

The model was trained over 20 epochs. In each epoch, two primary phases were executed: training and validation.

Training phase:

- The model is set to training mode to enable dropout and batch normalization.
- We loop over batches of training data, moving them to the GPU if available.
- For each batch:
  - The optimizer's gradients are reset.
  - Predictions are generated by the model.
  - The loss is computed based on the true labels.
  - Gradients are computed via backpropagation.
  - The model parameters are updated.
- Accuracy and loss are tracked across all batches.

Validation phase:

- The model is set to evaluation mode, disabling dropout and batch normalization.
- No weight updates occur during this phase.
- We evaluate the model on the validation set to measure generalization.
- We compute the validation loss and accuracy.
- All predictions and true labels are saved for performance evaluation.

3.4 Monitoring Performance

During training, we recorded loss and accuracy for both training and validation sets across epochs. These metrics were plotted to visualize trends and identify any overfitting or underfitting behaviors.

3.5 Final Evaluation

After training, the model was evaluated using a classification report, including precision, recall, and F1-score for each class. A confusion matrix was also generated to visually inspect prediction accuracy for each waste category.

```
print(classification_report(all_labels, all_preds, target_names=full_dataset.classes))
```

3.6 Summary

The ResNet-50 model, enhanced with a custom classifier, was successfully trained using a transfer learning strategy. The training loop alternated between optimization and validation, ensuring both efficiency and generalization. The approach yielded strong performance on unseen data and robust evaluation metrics per class.

4 Results

4.1 Overview

This section presents a comparative analysis of two convolutional neural network (CNN) models developed for automatic waste classification using image data.

- **Version 1** was trained on the publicly available TrashNet dataset with six general waste categories.
- **Version 2** used a custom dataset with five more specific and refined waste classes.

The objective is to assess model performance in terms of classification metrics (precision, recall, F1-score) and interpret the results to understand strengths, weaknesses, and dataset-specific behaviors.

4.2 Classification Report: Version 1 (TrashNet - 6 Classes)

	precision	recall	f1-score	support
cardboard	0.85	0.92	0.88	36
glass	0.91	0.72	0.81	40
metal	0.75	0.91	0.82	43
paper	0.87	0.91	0.89	58
plastic	0.88	0.85	0.86	60
trash	0.70	0.47	0.56	15
accuracy			0.84	252
macro avg	0.83	0.80	0.80	252
weighted avg	0.84	0.84	0.84	252

4.3 Classification Report: Version 2 (Custom Dataset - 5 Classes)

	precision	recall	f1-score	support
cans	0.43	0.75	0.55	4
food	0.82	0.82	0.82	11
glass	0.90	0.90	0.90	10

paper	0.92	0.92	0.92	12
plastic	0.90	0.69	0.78	13
accuracy			0.82	50
macro avg	0.79	0.82	0.79	50
weighted avg	0.85	0.82	0.83	50

4.4 Training and Validation Metrics

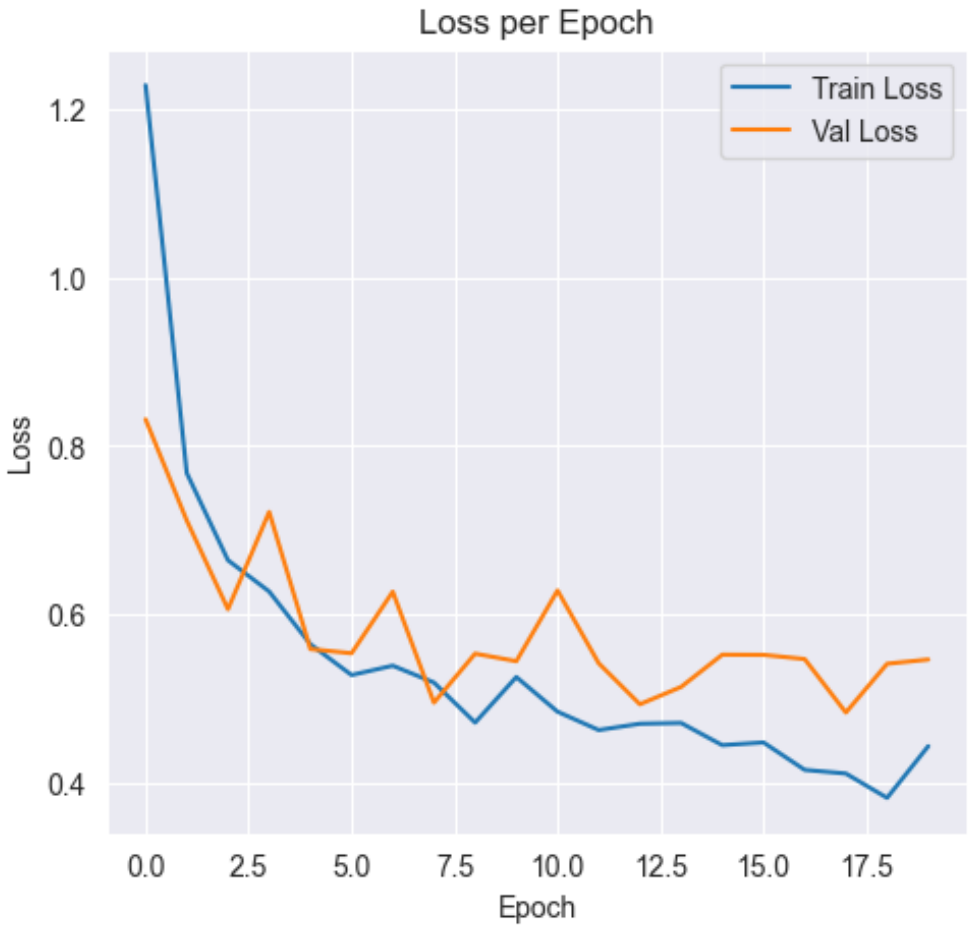

Figure 1: Training and validation loss per epoch. The training loss decreases steadily, indicating effective learning. The validation loss fluctuates slightly but generally follows a downward trend.

Explanation: Figure 1 shows the loss progression over training epochs. The blueblue line represents training loss, which consistently decreases, suggesting that the model is learning effectively. The orangeorange line represents validation loss, which varies but also trends downward, indicating generalization, though with some noise.

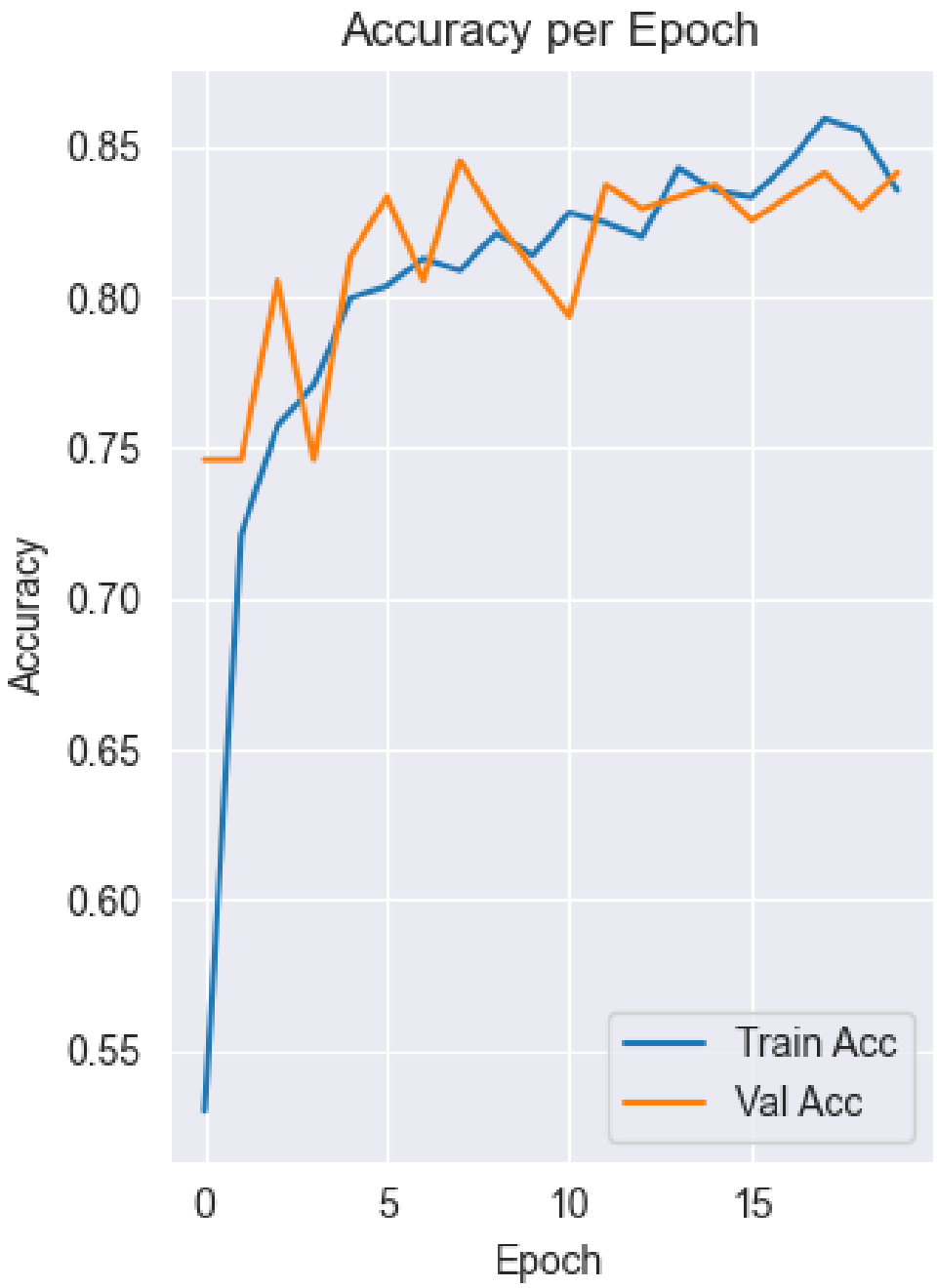

Figure 2: Training and validation accuracy per epoch. Training accuracy steadily improves, while validation accuracy fluctuates, potentially indicating overfitting.

Explanation: In Figure 2, the blueblue line shows training accuracy improving steadily across epochs. The orangeorange line for validation accuracy fluctuates more, which may signal mild overfitting — the model performs better on the training data than on unseen validation data.

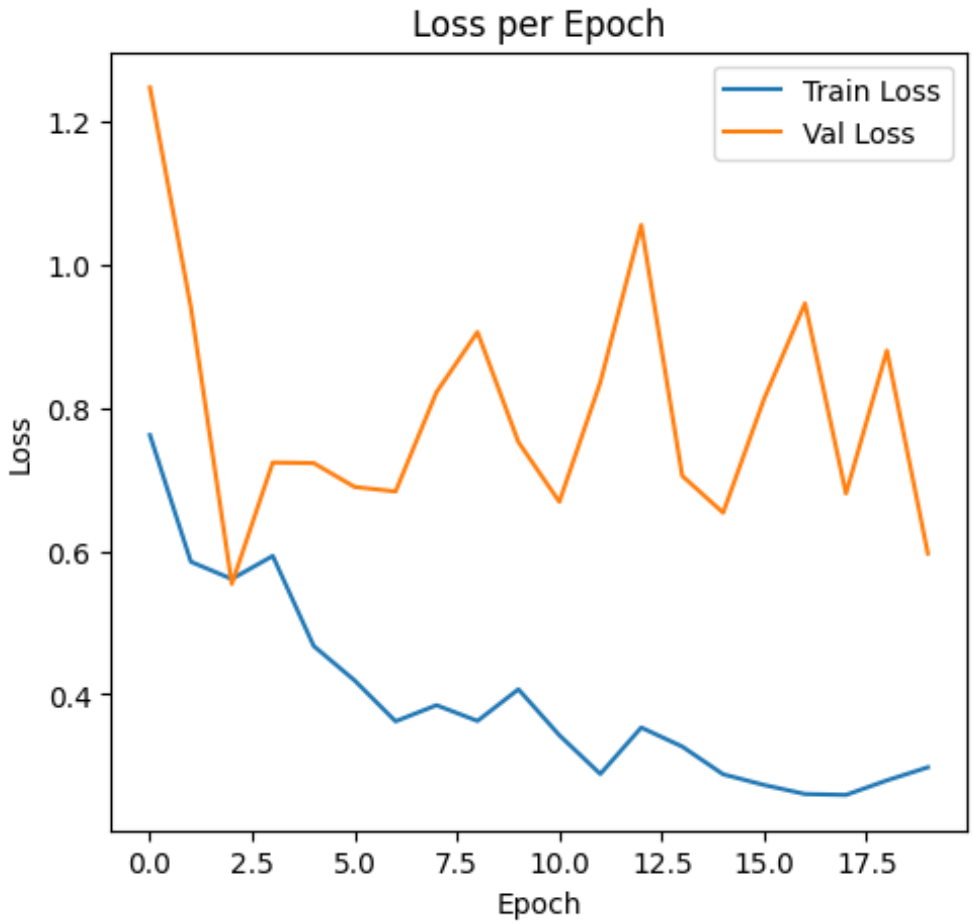

Figure 3: Loss per epoch with improved contrast. The lighter background enhances visibility. The training loss shows consistent decline, while validation loss trends similarly with some variation.

Explanation: Figure 3 mirrors the first plot but uses a lighter background for better readability. The model's training loss decreases reliably, and the validation loss follows a similar, albeit noisier, trajectory.

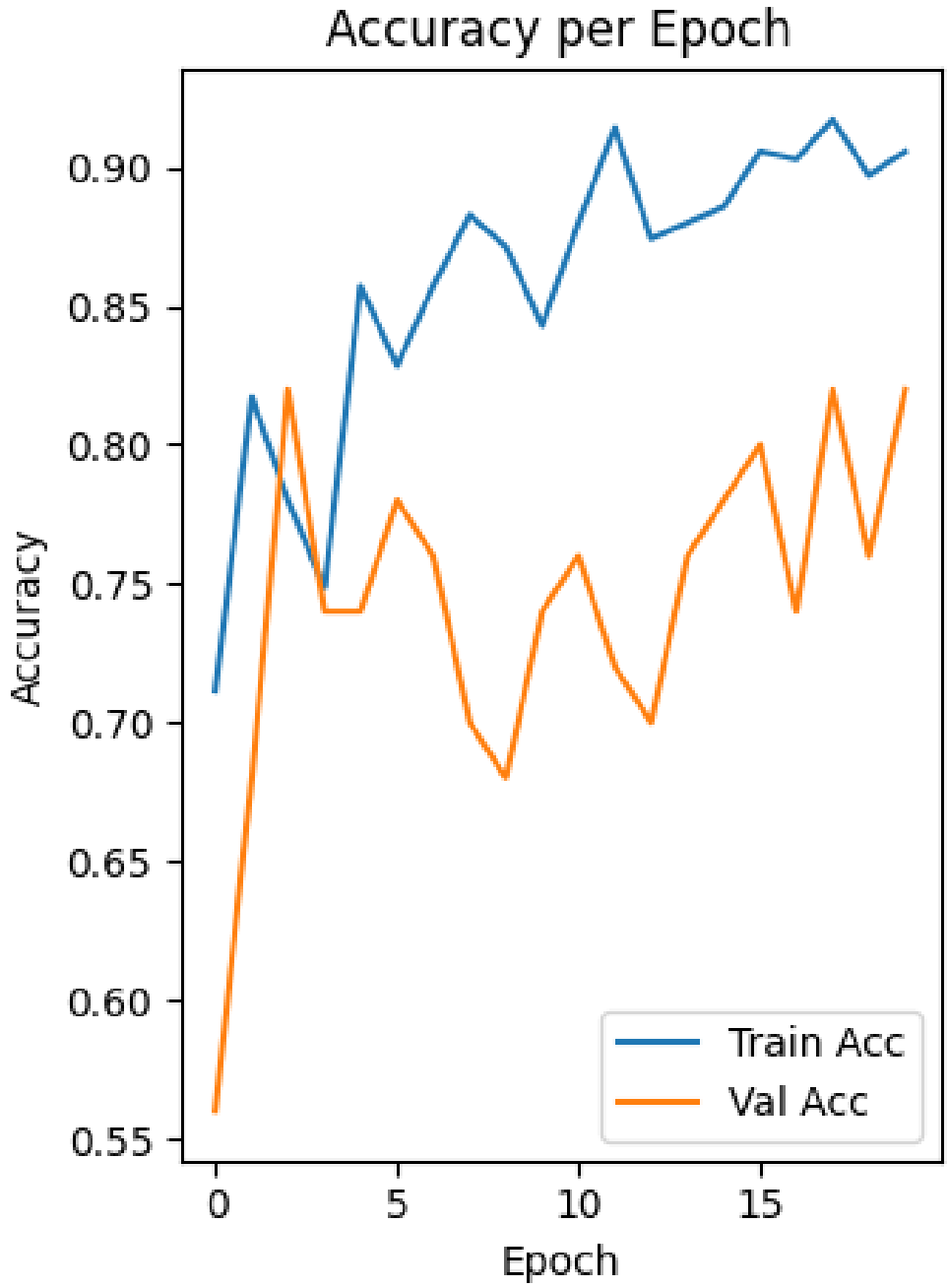

Figure 4: Accuracy per epoch with improved contrast. The lighter background improves clarity of accuracy trends for both training and validation sets.

Explanation: As shown in Figure 4, training accuracy increases consistently, reflecting learning progress. Validation accuracy remains more volatile, which again highlights the need for regularization or additional tuning to reduce overfitting.

4.5 Comparison and Interpretation

Version 1 (TrashNet). This model achieved an overall accuracy of 84%. It performed particularly well on categories such as *paper* (F1 = 0.89) and *cardboard* (F1 = 0.88). However, performance declined for the *trash* class (F1 = 0.56), likely due to visual ambiguity or class imbalance within the dataset.

Version 2 (Custom Dataset). Trained on a smaller dataset with only 50 test samples, Version 2 achieved an accuracy of 82%. It excelled in classifying *glass*, *paper*, and *food* (F1 ¿ 0.82), indicating strong learning in these categories. Its primary weakness was the *cans* class, where low precision (0.43) revealed a tendency to misclassify other items as "cans."

Macro vs. Weighted Metrics. Both models demonstrated comparable macro F1-scores (0.80 for Version 1, 0.79 for Version 2), suggesting balanced class-level performance. However, the weighted average F1 favored Version 1, reflecting its larger and more representative training dataset.

4.6 Summary of Comparison

- **Version 1** shows stronger generalization and robustness, benefiting from a more diverse and balanced dataset.
- **Version 2** delivers strong results in narrowly defined classes, despite limited data, and shows potential for specialized applications.
- Performance gaps in classes like *trash* and *cans* highlight the importance of class balance and targeted data augmentation.

Conclusion: Version 1 is better suited for general-purpose waste classification due to its broader coverage and higher overall performance. Version 2, although limited by data size, offers promising results for focused classification tasks. Enhancing data diversity and addressing underrepresented classes will be crucial for future model improvements.

5 Waste Classifier using Deep Learning

In this section, we present the code used to build a waste classification model that predicts whether an image of waste belongs to a recyclable or landfill category. The model is based on a pre-trained ResNet-50, with a custom classifier built on top. The framework used for deploying the model is Streamlit, which allows the user to upload an image and classify it in real-time.

5.1 Libraries and Dependencies

The following libraries are imported to support the model and Streamlit application:

- `torch`, `torch.nn`: PyTorch for building and training the deep learning model.

- `torchvision`: Provides pre-trained models, such as ResNet-50.
- `PIL`: For image processing.
- `matplotlib`: For displaying images.
- `streamlit`: For creating the interactive web app.

5.2 Model and Class Labels

- **Class Names:** The output of the model includes six classes: `cardboard`, `glass`, `metal`, `paper`, `plastic`, and `trash`. These are the categories for waste classification.
- **Category Map:** A mapping between the original class names and general categories (either `Recyclable` or `Landfill`).

The class names and category map are defined as follows:

```
class_names = ['cardboard', 'glass', 'metal', 'paper', 'plastic', 'trash']
category_map = {
    'cardboard': 'Recyclable',
    'glass': 'Recyclable',
    'metal': 'Recyclable',
    'paper': 'Recyclable',
    'plastic': 'Recyclable',
    'trash': 'Landfill'
}
```

5.3 Image Transformation

To prepare the images for the ResNet-50 model, the following transformations are applied:

- `Resize(256)`: Resizes the image to 256 pixels.
- `CenterCrop(224)`: Crops the image to 224x224 pixels to match the input size expected by ResNet-50.
- `ToTensor()`: Converts the image into a PyTorch tensor.
- `Normalize`: Normalizes the image using pre-defined mean and standard deviation values based on the ImageNet dataset.

The transformation pipeline is:

```
transform = transforms.Compose([
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
])
```

5.4 Model Loading and Setup

The model is loaded using the pre-trained ResNet-50 architecture. A custom fully connected layer is added on top of the ResNet-50's features to perform the waste classification task. The new layer consists of:

- `Linear` layer to map the ResNet-50 output to 256 features.
- `ReLU` activation function.
- `Dropout` for regularization.
- `Linear` layer to map the output to 6 classes.
- `LogSoftmax` to obtain log probabilities for classification.

The model is cached for efficiency, ensuring it only loads once:

```
@st.cache_resource
def load_model(model_path):
    model = models.resnet50(pretrained=False)
    model.fc = nn.Sequential(
        nn.Linear(model.fc.in_features, 256),
        nn.ReLU(),
        nn.Dropout(0.4),
        nn.Linear(256, 6),  # 6 classes from TrashNet
        nn.LogSoftmax(dim=1)
    )
    model.load_state_dict(torch.load(model_path, map_location=torch.device('cpu')))
    model.eval()
    return model
```

5.5 Image Prediction

The function `predict_image` takes an image and performs the following steps:

- Transforms the input image using the defined transformation.
- Runs the image through the model to obtain predictions.
- Extracts the predicted class label using `torch.max`.
- Maps the class label to its general category (Recyclable or Landfill).

```
def predict_image(model, image):
    image_tensor = transform(image).unsqueeze(0)  # Add batch dimension
    with torch.no_grad():
        output = model(image_tensor)
        _, pred = torch.max(output, 1)
    class_label = class_names[pred.item()]
    return class_label, category_map[class_label]
```

5.6 Streamlit Interface

Streamlit is used to create a simple user interface. The application allows users to upload an image of waste and displays the classification result.

The user is prompted to upload an image, which is then displayed in the app. After loading the model, the `predict_image` function is called, and the prediction results (class label and general category) are shown:

```
st.title(" Waste Classifier")
st.write("Upload a waste image to classify it as **Recyclable** or **Landfill**")
```

Additionally, if the user selects the checkbox to show class probabilities, the model's output probabilities are displayed for all classes:

```
if st.checkbox("Show Class Probabilities"):
    image_tensor = transform(image).unsqueeze(0)
    with torch.no_grad():
        output = model(image_tensor)
        probs = torch.exp(output).squeeze().numpy()

    st.markdown("###  Class Probabilities (%):")
    for idx, class_name in enumerate(class_names):
        percentage = probs[idx] * 100
        category = category_map[class_name]
        st.write(f"**{class_name.capitalize()}**: {percentage:.2f}% (_{category}_)")
```

5.7 Summary

This application uses a pre-trained ResNet-50 model with a custom classifier to classify images of waste into recyclable and landfill categories. Streamlit is used for building the web interface, allowing for easy interaction with the model. The model can predict the waste type and provide class probabilities, helping users understand the confidence level of each classification.

6 How the Waste Classifier Works

The waste classifier works by taking an image of waste, processing it, and classifying it into one of the six categories: `cardboard`, `glass`, `metal`, `paper`, `plastic`, and `trash`. The entire process is illustrated in the image below.

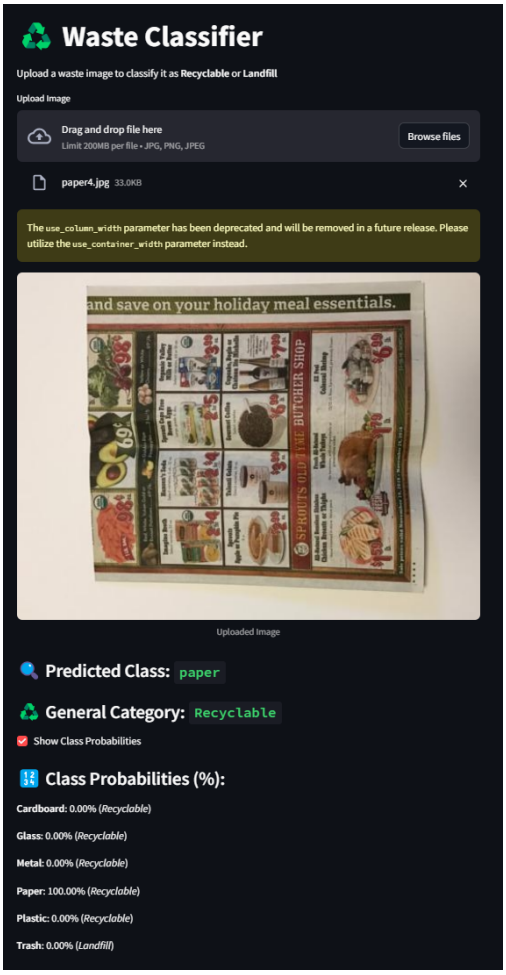

Figure 5: How the Waste Classifier works: From image upload to classification.

The process can be summarized as follows:

- **Step 1: Upload Image** - The user uploads an image of waste.
- **Step 2: Preprocessing** - The image is resized, cropped, and normalized to prepare it for classification.
- **Step 3: Prediction** - The pre-trained ResNet-50 model classifies the image into one of the six categories.
- **Step 4: Results Displayed** - The predicted class (e.g., `cardboard`) and its category (Recyclable or Landfill) are shown.
- **Step 5: Optional Class Probabilities** - The user can opt to view the class probabilities, showing the likelihood for each waste type.

7 Conclusion

In this project, we developed and evaluated two convolutional neural network (CNN) models for the task of automated waste classification using image data. Version 1 was trained on the public TrashNet dataset with six general waste categories, while Version 2 was trained on a custom dataset with five more specific classes.

Experimental results show that both models achieved strong classification performance, with Version 1 achieving 84% accuracy and Version 2 achieving 82%. Version 1 benefited from a larger and more diverse dataset, resulting in greater generalizability across all waste types. In contrast, Version 2, although trained on a smaller dataset, excelled in predicting well-defined categories such as glass and paper.

These findings highlight the potential of CNN-based models in supporting environmental sustainability efforts by enabling accurate waste sorting. Moreover, even modest datasets, when carefully curated, can yield competitive results.

8 Future Work

To further improve model performance and practical applicability, several directions can be pursued:

- **Dataset Expansion:** Increase the volume and variety of training data, especially for underrepresented classes such as trash and cans, to reduce class imbalance and improve generalization.
- **Real-Time Deployment:** Integrate the trained model into an edge device or mobile application for real-time waste detection and classification.
- **Augmentation Techniques:** Experiment with advanced data augmentation (e.g., CutMix, MixUp) and synthetic data generation (e.g., GANs) to improve model robustness.
- **Transfer Learning Optimization:** Fine-tune more layers of the pretrained ResNet50 or explore alternative architectures like EfficientNet or MobileNet for better trade-offs between accuracy and computational efficiency.
- **Multimodal Classification:** Combine image data with metadata (e.g., location, sensor input) to enhance classification in ambiguous cases.

These enhancements will help build a more scalable and accurate system for waste classification in real-world applications.

9 Additional Resources

In addition to the core information provided in this report, the following resources are available for further exploration:

- **TrashDet with TrashNet dataset:** You can access the waste classification application using TrashNet Dataset here: https://trashnetapp.streamlit.app/
- **TrashDet with Custom dataset:** You can access the waste classification application using Custom Dataset here: https://trashnetappv2.streamlit.app/
- **YouTube Channel for Explanation:** For further explanations and video tutorials, visit the YouTube channel: https://youtu.be/9yo4Z9-XLO8?si=1uX2R8rSFocTWZIl

10 References

1. T. H. Thung and M. R. Yang, "Classification of Trash for Recyclability Status," Stanford University CS229 Project Report, 2016. https://cs229.stanford.edu/proj2016/report/ThungYang-ClassificationOfTrashForRecyclabilityStatus-report.pdf
2. S. He, D. Song, and Y. Zhuang, "Deep Learning for Image-Based Waste Classification," *IEEE Access*, vol. 7, pp. 180449–180457, 2019.
3. K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," arXiv preprint arXiv:1409.1556, 2014.
4. A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," in *Advances in Neural Information Processing Systems*, vol. 25, 2012.
5. M. Tan and Q. Le, "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks," in *Proc. ICML*, 2019.
6. OpenAI, "ChatGPT", https://chat.openai.com/
7. FreeCodeCamp, "Trash Classification Using CNN with PyTorch," YouTube, https://www.youtube.com/watch?v=dg3pza4y2ws
8. Nicholas Renotte, "Image Classification with PyTorch," YouTube, https://www.youtube.com/watch?v=Z_v8Hq3yG2g
9. PyTorch Official Documentation, https://pytorch.org/docs/stable/index.html
10. Torchvision Documentation, https://pytorch.org/vision/stable/index.html
11. Scikit-learn Documentation, https://scikit-learn.org/stable/documentation.html