Deep Learning Applications: Sustainability

Waste Classification

Problem Statement

The population has been significantly growing which has intensified waste production. Therefore, waste management is a critical global challenge needed to be strictly considered. **Automating waste sorting** serves as a key step in garbage segregation, enabling advanced solutions that promotes the sustainability.

Vision: Recognition & Classification

Idea

Develop an AI model to recognize and classify waste by using a camera with a flash built in every trash bin.

Key Benefit

Increase efficiency in filtering wastes, remove the need for multiple trash bin types and resorting for people's incorrect behaviours.

Why It Matters

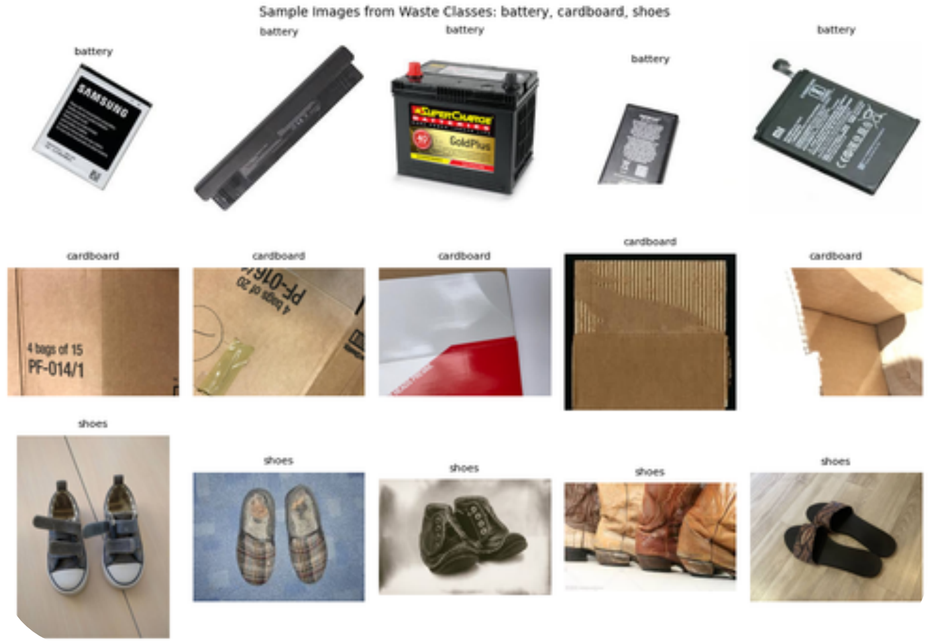
- Multiple trash bins are not practical and not optimal for every area.
- Automate trash filtering for a higher accuracy of processing and recycling.
- Technology is significantly developed.

Import Datasets & Libraries

This dataset contains images of garbage items categorized into 10 classes, designed for machine learning and computer vision projects focusing on recycling and waste management. It is ideal for building classification or object detection models or developing AI-powered solutions for sustainable waste disposal.

Dataset Summary:

- Metal: 1020
- Glaass: 3061
- Biological: 997
- Paper: 1680
- Battery: 944
- Cardboard: 1825
- Shoes: 1977
- Clothes: 5327
- Plastic: 1984
- Trash: 947



Exploratory Data Analysis

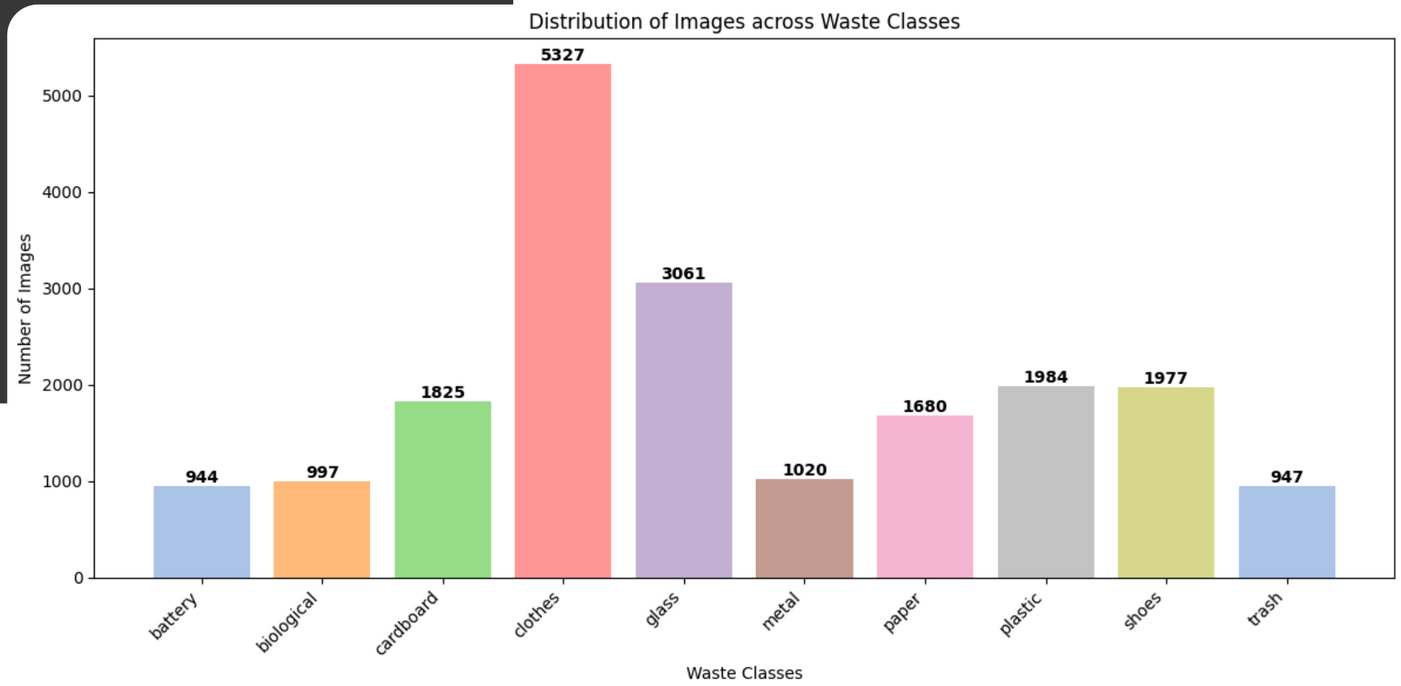
```python
# Function to get image dimensions
def get_image_dimensions(image_path):
    img = cv2.imread(image_path)
    return img.shape[:2]  # Returns (height, width)

df['Dimension'] = df['Image_Path'].apply(get_image_dimensions)
df['Dimension'].value_counts()
```

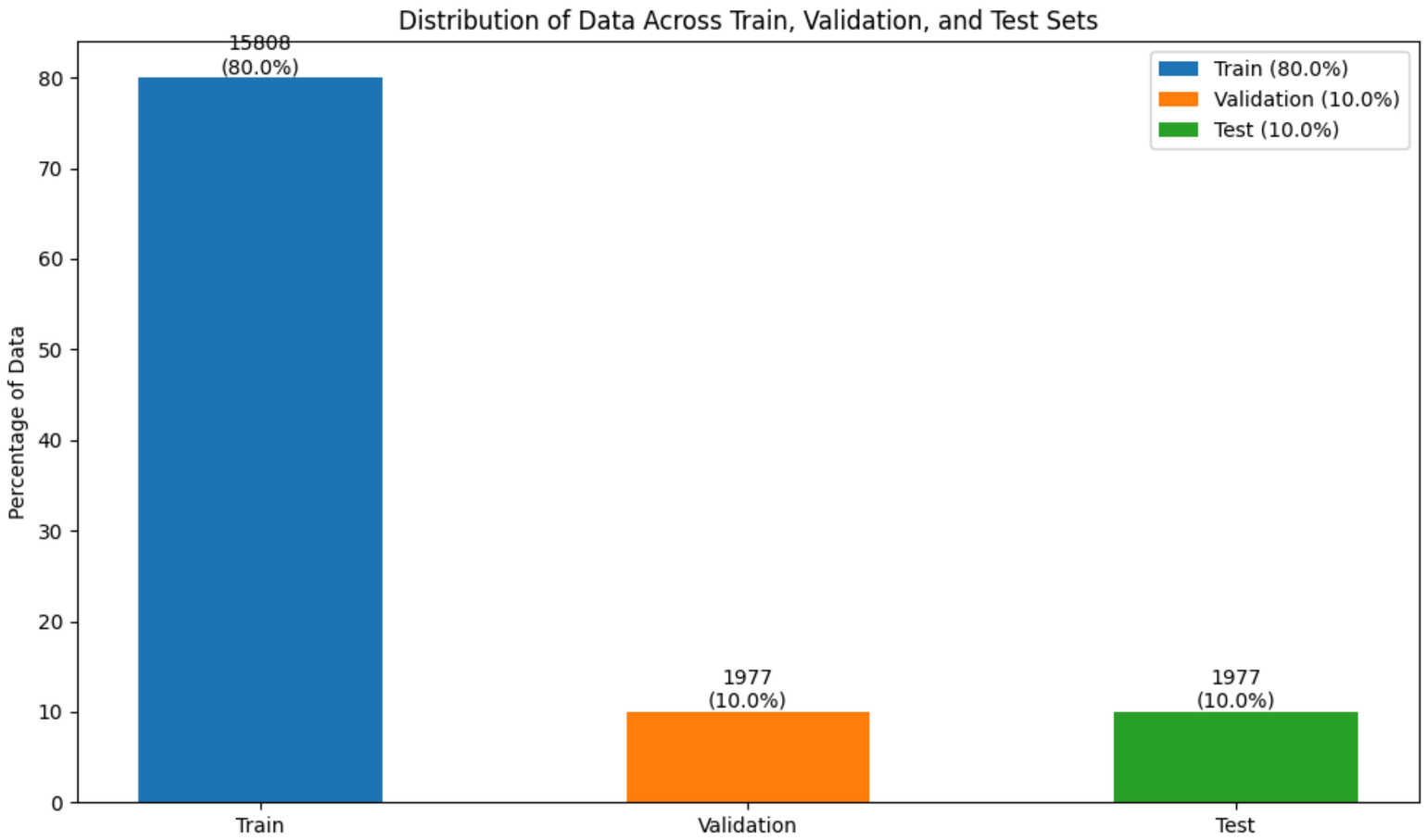
Dimension	count
(533, 400)	2591
(384, 512)	2524
(225, 225)	1924
(534, 400)	1077
(183, 275)	671
...	...
(375, 474)	1
(377, 565)	1
(240, 475)	1
(273, 346)	1
(196, 500)	1

2731 rows × 1 columns

Through various visualization techniques and statistical analyses, we'll explore aspects such as class distribution, image properties among waste types. This process will not only enhance our understanding of the dataset but also inform our decisions in subsequent stages of model development to better prepare our data and tailor our deep learning approach to achieve optimal classification performance.



Feature Engineering



Training Set (80%): It will expose the model to a wide variety of waste images, allowing it to learn the distinctive features of each waste category. The model will adjust its parameters based on this data to minimize classification errors.

Validation Set (10%): During the training process, we'll use this set to evaluate the model's performance on data it hasn't directly learned from. This helps us adjust hyperparameters, prevent overfitting, and make informed decisions about model architecture or training strategies.

Testing Set (10%): This final portion of our data remains completely unseen by the model during the entire training and validation process. We'll use it to assess the model's true performance and its ability to generalize to new, unseen waste images. This set provides the most realistic measure of how well our model might perform in real-world waste classification scenarios.

Model Training

ANN Training

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 299, 299, 32)	896
batch_normalization (BatchNormalization)	(None, 299, 299, 32)	128
re_lu (ReLU)	(None, 299, 299, 32)	0
max_pooling2d (MaxPooling2D)	(None, 149, 149, 32)	0
conv2d_1 (Conv2D)	(None, 149, 149, 64)	18,496
batch_normalization_1 (BatchNormalization)	(None, 149, 149, 64)	256
re_lu_1 (ReLU)	(None, 149, 149, 64)	0
max_pooling2d_1 (MaxPooling2D)	(None, 74, 74, 64)	0
conv2d_2 (Conv2D)	(None, 74, 74, 128)	73,856
batch_normalization_2 (BatchNormalization)	(None, 74, 74, 128)	512
re_lu_2 (ReLU)	(None, 74, 74, 128)	0
max_pooling2d_2 (MaxPooling2D)	(None, 37, 37, 128)	0
global_average_pooling2d (GlobalAveragePooling2D)	(None, 128)	0
dense (Dense)	(None, 256)	33,024
dropout (Dropout)	(None, 256)	0
dense_1 (Dense)	(None, 10)	2,570

Total params: 129,738 (506.79 KB)
Trainable params: 129,290 (505.04 KB)
Non-trainable params: 448 (1.75 KB)

Model structure:

- Three Convolutional Blocks, followed by batch normalization, ReLU activation, and max pooling
- Global Average Pooling to reduces the number of parameters helps the model generalize better
- Fully Connected Layer
- Drop out to prevent overfitting
- Output Layer

ANN Improvement

Layer (type)	Output Shape	Param #
inception_v3 (Functional)	(None, 8, 8, 2048)	21,802,784
global_average_pooling2d (GlobalAveragePooling2D)	(None, 2048)	0
dense (Dense)	(None, 256)	524,544
dropout (Dropout)	(None, 256)	0
dense_1 (Dense)	(None, 10)	2,570

Total params: 22,329,898 (85.18 MB)
Trainable params: 527,114 (2.01 MB)
Non-trainable params: 21,802,784 (83.17 MB)

Calculate balanced weights based on the frequency of each class in the training data.

Model Training

ANN Training

Maximum training accuracy: 0.6768

We utilized 50 epochs for our training. Also, we will implement what we're written before, that is callbacks and class_weight.
However, The decision to use `50 epochs` because of the time limit. Our aim is using 70+ epochs derived from trial and error experiments, where it shown that turns out there is minimal improvement in the model's performance beyond this point. Using a higher number of epochs allows us to better understand the performance of our model.

```
494/494 ━━━━━━━━━━━━━━━━━━━━ 415s 839ms/step - accuracy: 0.6320 - loss: 1.0169 - val_accuracy: 0.5584 - val_loss: 1.3460
Epoch 35/50
494/494 ━━━━━━━━━━━━━━━━━━━━ 401s 812ms/step - accuracy: 0.6418 - loss: 1.0065 - val_accuracy: 0.6166 - val_loss: 1.1742
Epoch 36/50
494/494 ━━━━━━━━━━━━━━━━━━━━ 396s 802ms/step - accuracy: 0.6455 - loss: 1.0002 - val_accuracy: 0.5367 - val_loss: 1.4555
Epoch 37/50
494/494 ━━━━━━━━━━━━━━━━━━━━ 358s 724ms/step - accuracy: 0.6463 - loss: 1.0119 - val_accuracy: 0.5331 - val_loss: 1.4832
Epoch 38/50
494/494 ━━━━━━━━━━━━━━━━━━━━ 408s 826ms/step - accuracy: 0.6507 - loss: 0.9768 - val_accuracy: 0.5427 - val_loss: 1.4104
Epoch 39/50
494/494 ━━━━━━━━━━━━━━━━━━━━ 480s 971ms/step - accuracy: 0.6595 - loss: 0.9630 - val_accuracy: 0.5878 - val_loss: 1.3234
Epoch 40/50
494/494 ━━━━━━━━━━━━━━━━━━━━ 469s 950ms/step - accuracy: 0.6578 - loss: 0.9508 - val_accuracy: 0.4871 - val_loss: 1.9928
Epoch 41/50
494/494 ━━━━━━━━━━━━━━━━━━━━ 470s 952ms/step - accuracy: 0.6620 - loss: 0.9493 - val_accuracy: 0.5817 - val_loss: 1.2808
Epoch 42/50
494/494 ━━━━━━━━━━━━━━━━━━━━ 414s 837ms/step - accuracy: 0.6623 - loss: 0.9442 - val_accuracy: 0.6065 - val_loss: 1.1938
Epoch 43/50
494/494 ━━━━━━━━━━━━━━━━━━━━ 387s 783ms/step - accuracy: 0.6686 - loss: 0.9167 - val_accuracy: 0.6515 - val_loss: 1.1000
Epoch 44/50
494/494 ━━━━━━━━━━━━━━━━━━━━ 400s 808ms/step - accuracy: 0.6677 - loss: 0.9317 - val_accuracy: 0.6004 - val_loss: 1.2525
Epoch 45/50
494/494 ━━━━━━━━━━━━━━━━━━━━ 381s 771ms/step - accuracy: 0.6650 - loss: 0.9413 - val_accuracy: 0.6378 - val_loss: 1.1230
Epoch 46/50
494/494 ━━━━━━━━━━━━━━━━━━━━ 298s 603ms/step - accuracy: 0.6659 - loss: 0.9319 - val_accuracy: 0.5746 - val_loss: 1.3868
Epoch 47/50
494/494 ━━━━━━━━━━━━━━━━━━━━ 335s 678ms/step - accuracy: 0.6625 - loss: 0.9200 - val_accuracy: 0.5594 - val_loss: 1.4808
Epoch 48/50
494/494 ━━━━━━━━━━━━━━━━━━━━ 348s 703ms/step - accuracy: 0.6768 - loss: 0.8802 - val_accuracy: 0.4638 - val_loss: 2.2152
Epoch 49/50
494/494 ━━━━━━━━━━━━━━━━━━━━ 358s 723ms/step - accuracy: 0.6777 - loss: 0.9081 - val_accuracy: 0.4901 - val_loss: 1.8280
Epoch 50/50
494/494 ━━━━━━━━━━━━━━━━━━━━ 375s 758ms/step - accuracy: 0.6805 - loss: 0.8929 - val_accuracy: 0.6211 - val_loss: 1.1614
Restoring model weights from the end of the best epoch: 43.
```

```
Epoch 30/50
494/494 ━━━━━━━━━━━━━━━━━━━━ 242s 489ms/step - accuracy: 0.8804 - loss: 0.3706 - val_accuracy: 0.9292 - val_loss: 0.2313
Epoch 31/50
494/494 ━━━━━━━━━━━━━━━━━━━━ 289s 584ms/step - accuracy: 0.8780 - loss: 0.3700 - val_accuracy: 0.9342 - val_loss: 0.2076
Epoch 32/50
494/494 ━━━━━━━━━━━━━━━━━━━━ 284s 574ms/step - accuracy: 0.8768 - loss: 0.3757 - val_accuracy: 0.9353 - val_loss: 0.2086
Epoch 33/50
494/494 ━━━━━━━━━━━━━━━━━━━━ 279s 564ms/step - accuracy: 0.8794 - loss: 0.3807 - val_accuracy: 0.9363 - val_loss: 0.2095
Epoch 34/50
494/494 ━━━━━━━━━━━━━━━━━━━━ 286s 580ms/step - accuracy: 0.8781 - loss: 0.3730 - val_accuracy: 0.9393 - val_loss: 0.2015
Epoch 35/50
494/494 ━━━━━━━━━━━━━━━━━━━━ 294s 594ms/step - accuracy: 0.8854 - loss: 0.3559 - val_accuracy: 0.9353 - val_loss: 0.2155
Epoch 36/50
494/494 ━━━━━━━━━━━━━━━━━━━━ 313s 632ms/step - accuracy: 0.8854 - loss: 0.3407 - val_accuracy: 0.9373 - val_loss: 0.2144
Epoch 37/50
494/494 ━━━━━━━━━━━━━━━━━━━━ 331s 668ms/step - accuracy: 0.8841 - loss: 0.3646 - val_accuracy: 0.9358 - val_loss: 0.2064
Epoch 38/50
494/494 ━━━━━━━━━━━━━━━━━━━━ 291s 589ms/step - accuracy: 0.8845 - loss: 0.3461 - val_accuracy: 0.9206 - val_loss: 0.2472
Epoch 39/50
494/494 ━━━━━━━━━━━━━━━━━━━━ 284s 574ms/step - accuracy: 0.8794 - loss: 0.3609 - val_accuracy: 0.9297 - val_loss: 0.2295
Epoch 40/50
494/494 ━━━━━━━━━━━━━━━━━━━━ 270s 546ms/step - accuracy: 0.8842 - loss: 0.3481 - val_accuracy: 0.9378 - val_loss: 0.2001
Epoch 41/50
494/494 ━━━━━━━━━━━━━━━━━━━━ 286s 578ms/step - accuracy: 0.8896 - loss: 0.3441 - val_accuracy: 0.9373 - val_loss: 0.2157
Epoch 42/50
494/494 ━━━━━━━━━━━━━━━━━━━━ 291s 588ms/step - accuracy: 0.8821 - loss: 0.3542 - val_accuracy: 0.9363 - val_loss: 0.2063
Epoch 43/50
494/494 ━━━━━━━━━━━━━━━━━━━━ 270s 545ms/step - accuracy: 0.8845 - loss: 0.3498 - val_accuracy: 0.9251 - val_loss: 0.2291
Epoch 44/50
494/494 ━━━━━━━━━━━━━━━━━━━━ 275s 556ms/step - accuracy: 0.8875 - loss: 0.3393 - val_accuracy: 0.9363 - val_loss: 0.2096
Epoch 44: early stopping
Restoring model weights from the end of the best epoch: 29.
```

ANN Improvement (Transfer Training)

Maximum training accuracy: 0.8896

This approach reduces the need for extensive training data while providing a strong starting point for our specific task. We also reduce training time which speeds up the process and leads to better generalization, enhancing the model's ability to classify challenging waste categories, as seen in previous results. Therefore, this model only needs 44 eporchs to achieve a stable performance.

Model Evaluation

The model's performance is evaluated through a three steps approach:
1. Analysis of training and validation metrics (loss and accuracy)
2. Generation of comprehensive classification reports
3. Construction of a confusion matrix using the test dataset
These methods provide a general view of the model's effectiveness and generalization capabilities.

- The difference between the training and test accuracy has narrowed, with both datasets showing strong performance. The model now generalizes well, with less overfitting or underfitting (GOOD-FIT).
- Categories benefit from distinct visual patterns that the model captures effectively. On the other hand, others still show some confusion due to their visual variability but perform better than before.
- Overall, the confusion between classes has decreased, especially in some categories that were previously challenging showing that the model has greatly improved its ability to handle diverse waste categories.

494/494 — 136s 275ms/step
Classification Report for Training Set:

	precision	recall	f1-score	support
battery	0.59	0.80	0.68	756
biological	0.73	0.87	0.79	797
cardboard	0.79	0.72	0.75	1460
clothes	0.84	0.84	0.84	4261
glass	0.65	0.66	0.66	2449
metal	0.49	0.64	0.56	816
paper	0.65	0.56	0.60	1344
plastic	0.66	0.43	0.52	1587
shoes	0.53	0.54	0.54	1581
trash	0.57	0.66	0.61	757
accuracy			0.69	15808
macro avg	0.65	0.67	0.65	15808
weighted avg	0.69	0.69	0.69	15808

Training Set

494/494 — 222s 450ms/step

	precision	recall	f1-score	support
battery	0.91	0.88	0.90	756
biological	0.76	0.99	0.86	797
cardboard	0.82	0.90	0.86	1460
clothes	0.99	0.94	0.96	4261
glass	0.88	0.82	0.85	2449
metal	0.78	0.75	0.76	816
paper	0.87	0.78	0.82	1344
plastic	0.83	0.73	0.78	1587
shoes	0.90	0.91	0.90	1581
trash	0.59	0.85	0.70	757
accuracy			0.87	15808
macro avg	0.83	0.86	0.84	15808
weighted avg	0.87	0.87	0.87	15808

ANN Training vs ANN Improvement

62/62 — 5s 85ms/step
Classification Report for Test Set:

	precision	recall	f1-score	support
battery	0.50	0.74	0.60	94
biological	0.74	0.75	0.75	100
cardboard	0.61	0.73	0.66	183
clothes	0.79	0.82	0.81	533
glass	0.69	0.63	0.66	306
metal	0.37	0.46	0.41	102
paper	0.62	0.62	0.62	168
plastic	0.65	0.45	0.53	198
shoes	0.55	0.43	0.49	198
trash	0.51	0.53	0.52	95
accuracy			0.65	1977
macro avg	0.60	0.62	0.60	1977
weighted avg	0.65	0.65	0.65	1977

Test Set

62/62 — 16s 250ms/step

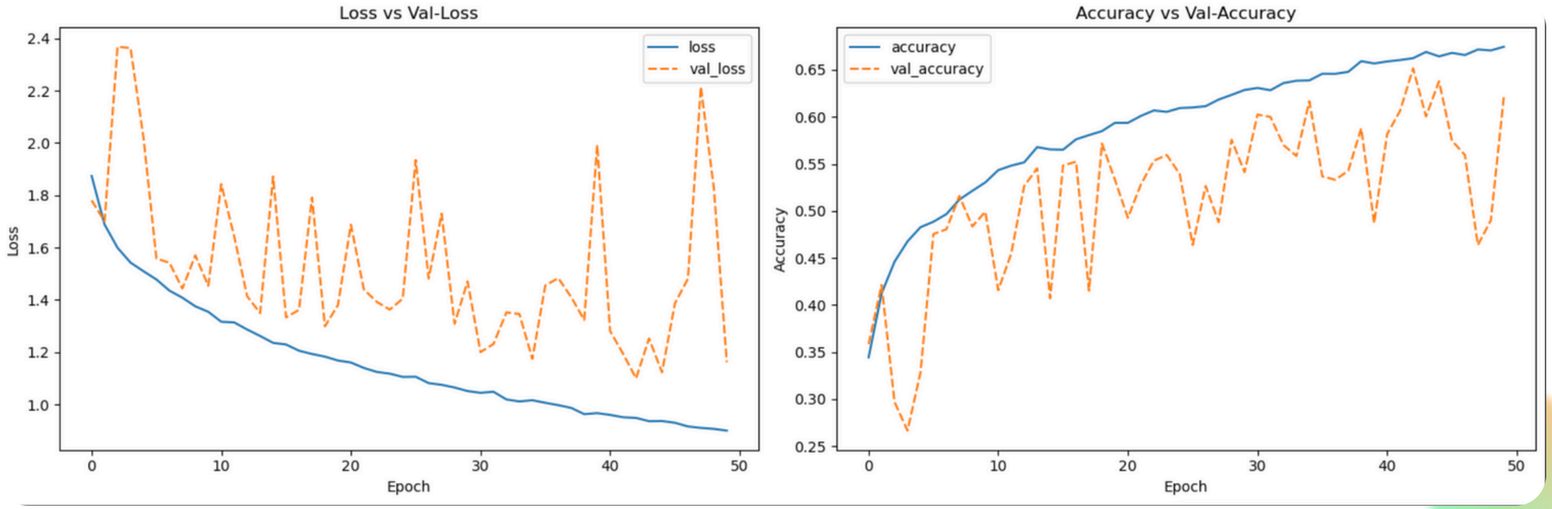
	precision	recall	f1-score	support
battery	0.98	0.84	0.90	94
biological	0.82	0.97	0.89	100
cardboard	0.85	0.84	0.84	183
clothes	0.98	0.98	0.98	533
glass	0.90	0.86	0.88	306
metal	0.72	0.82	0.77	102
paper	0.85	0.74	0.79	168
plastic	0.84	0.82	0.83	198
shoes	0.97	0.98	0.98	198
trash	0.66	0.80	0.72	95
accuracy			0.89	1977
macro avg	0.86	0.87	0.86	1977
weighted avg	0.89	0.89	0.89	1977

In general, there is a huge improvement between ANN Training and ANN Improvement in terms of Training Set and Test Set.

ANN Training vs ANN Improvement

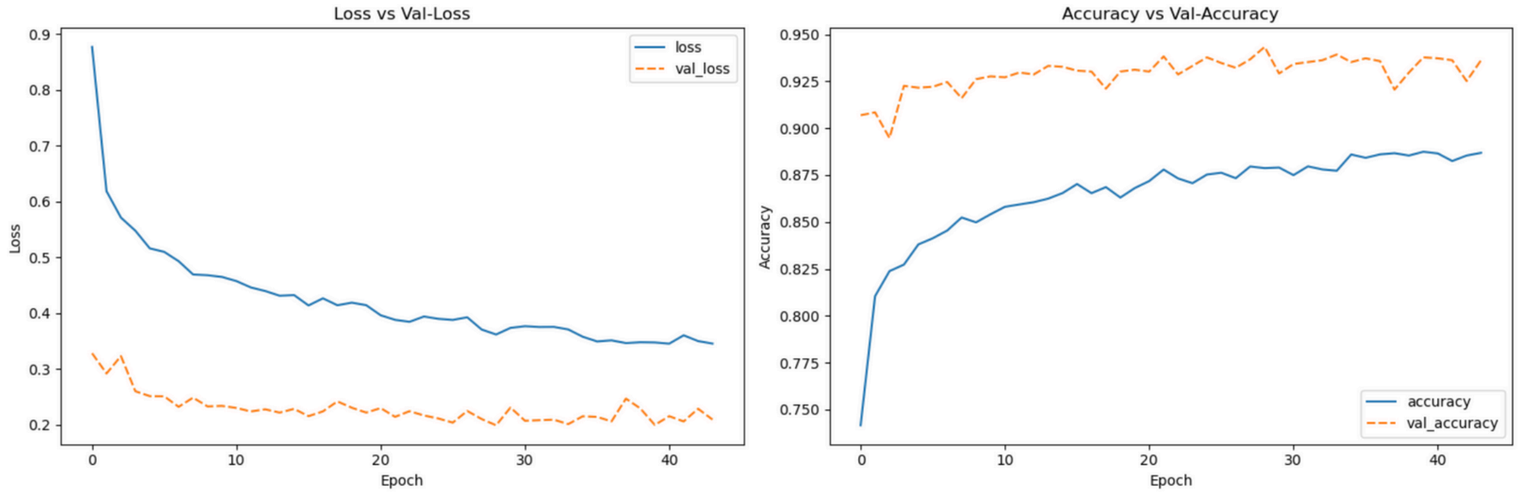
Model Evaluation

ANN Training



The close alignment between training and validation metrics indicates that the model is neither underfitting nor overfitting (a GOOD-FIT model), making it suitable for the waste classification task.

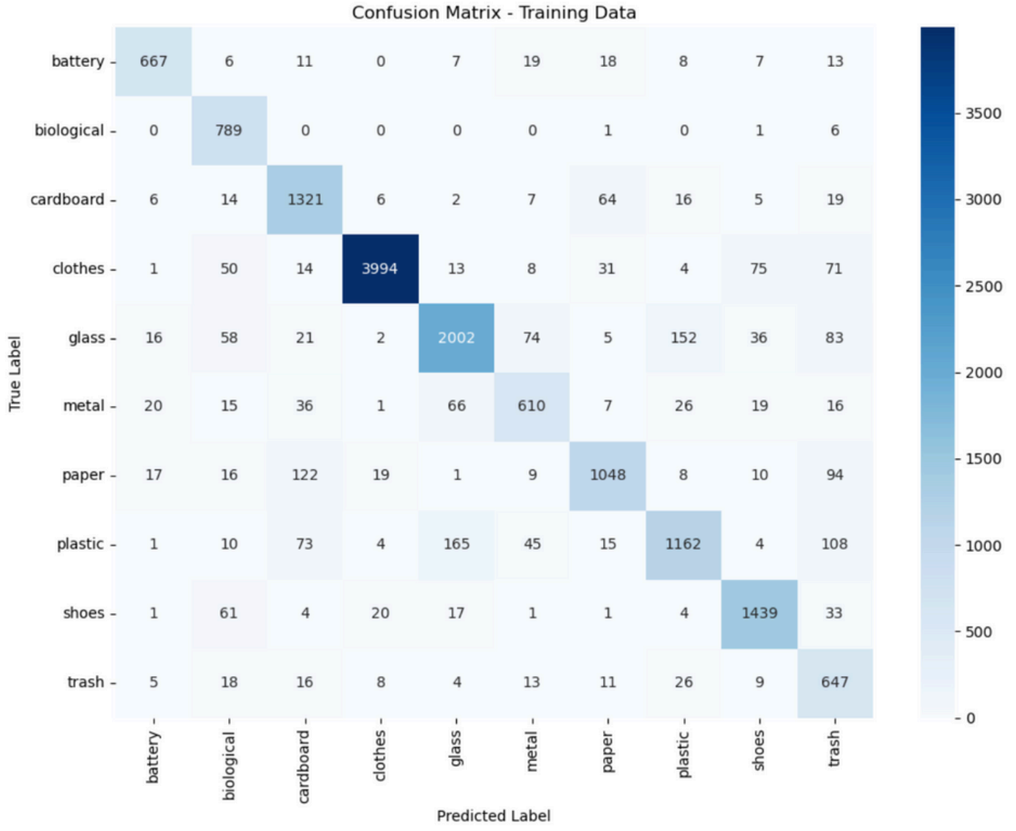
ANN Improvement

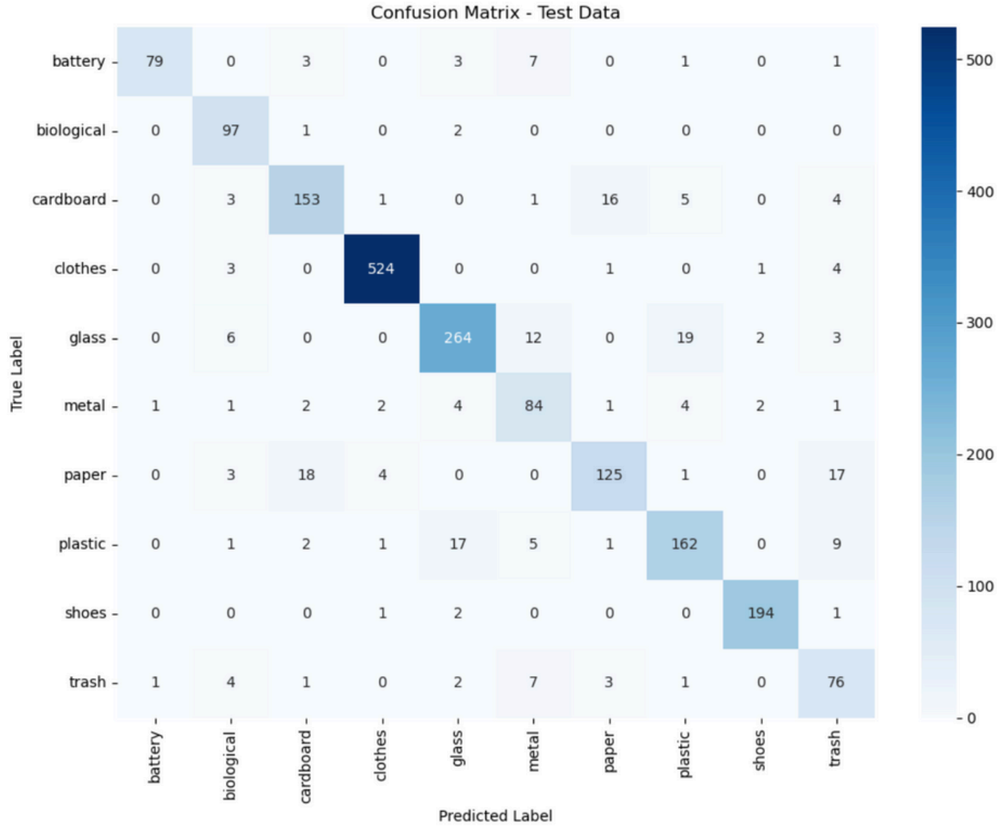


In general, there is a huge improvement between ANN Training and ANN Improvement in terms of Loss vs Val - Loss figure and Accuracy vs Val - Accuracy.

Model Evaluation

ANN Improvement





Future improvements

Improve software:

- A bigger dataset with a higher resolution.
- Using more advanced ANN Transfer Learning to have a higher stable performance and a well - optimized model.

Improve hardware:

- A camera with a higher resolution or infrared camera.
- A lidar to scan 3D of the object.
- A high - end processor.

Feasibility:

- Reduce the cost of this technology to make it more pratical.
- Plug - and - play design.
- Water resistant and dust resistant according to IP rating.
- Easy and cheap to repair.

References

- **Dataset for training:** https://www.kaggle.com/datasets/sumn2u/garbage-classification-v2
- **Reference code:** https://github.com/ayudhaamari/real-waste-classification-cnn/blob/main/realwaste-image-classification.ipynb