Simple convolutional neural network to perform classification

Team Emaralds

December 12, 2024

Prabath P T
210485C
Dept. of Electrical Engineering,
Univ. of Moratuwa, Sri Lanka
prabathpt.21@uom.lk

Prabhashana M R M
210486F
Dept. of Electrical Engineering,
Univ. of Moratuwa, Sri Lanka
prabhashanamrm.21@uom.lk

Prabhashwara H M R
210488M
Dept. of Electrical Engineering,
Univ. of Moratuwa, Sri Lanka
prabhashwarahmr.21@uom.lk

Dabarera W P J B
210090K
Dept. of Electrical Engineering,
Univ. of Moratuwa, Sri Lanka
dabarerawpjb.21@uom.lk

1 Introduction

This report outlines the steps involved in designing and implementing a simple convolutional neural network (CNN) to classify waste images into 9 categories: Cardboard, Food Organics, Glass, Metal, Miscellaneous Trash, Paper, Plastic, Textile Trash, and Vegetation. The goal of this assignment is to learn the fundamentals of CNNs and evaluate their performance in real-world classification tasks.

The dataset for this project was selected from the UCI Machine Learning Repository. Preprocessing techniques, including data augmentation and class balancing, were applied to handle the class imbalance. This document also compares the custom CNN model with state-of-the-art pre-trained models, highlighting the trade-offs, advantages, and limitations.

2 Dataset and Preprocessing

The dataset used for this project is the **RealWaste Dataset**, which contains images of waste categorized into the following classes:

- **Cardboard:** 461 images
- **Food Organics:** 411 images
- **Glass:** 420 images
- **Metal:** 790 images
- **Miscellaneous Trash:** 495 images
- **Paper:** 500 images

- **Plastic:** 921 images
- **Textile Trash:** 318 images
- **Vegetation:** 436 images

The dataset was imbalanced, with a varying number of images across classes. To address this issue, the mean value of the dataset size was calculated as **461 images**. Data balancing was achieved as follows:

1. For classes with more than 461 images (**Metal, Paper, Plastic, Miscellaneous Trash**), random sampling was used to reduce their count to 461 images.
2. For classes with fewer than 461 images (**Food Organics, Glass, Textile Trash, Vegetation**), data augmentation techniques were applied, including:
    - Rotation
    - Flipping (horizontal and vertical)
    - Brightness and contrast adjustments
    - Cropping

After preprocessing, each class contained exactly 461 images, resulting in a balanced dataset. The balanced dataset was then split into:

- **Training set:** 60% of the data
- **Validation set:** 20% of the data
- **Testing set:** 20% of the data

This preprocessing ensured that the model could learn effectively from all classes without biasing towards overrepresented categories.

3 CNN Architecture

The architecture of the implemented CNN is as follows:

1. **Input Layer:** Input shape of (150, 150, 3) to handle resized RGB images.
2. **Convolutional Layer:** 64 filters, 3×3 kernel, ReLU activation.
3. **Batch Normalization:** Improves training stability.
4. **MaxPooling Layer:** Pool size of 2×2.
5. **Second Convolutional Layer:** 128 filters, 3×3 kernel, ReLU activation.
6. **MaxPooling Layer:** Pool size of 2×2.

7. **Third Convolutional Layer:** 256 filters, 3×3 kernel, ReLU activation.
8. **MaxPooling Layer:** Pool size of 2×2.
9. **Flatten Layer:** Converts 2D feature maps to a 1D vector.
10. **Fully Connected Layer:** 128 units, ReLU activation.
11. **Dropout Layer:** Dropout rate of 0.3.
12. **Second Fully Connected Layer:** 64 units, ReLU activation.
13. **Output Layer:** 9 units (for 9 classes), softmax activation.

The model was compiled using the Adam optimizer with a learning rate of 0.0001 and sparse categorical crossentropy as the loss function.

4 Justifications for Activation Function Selections

Activation functions play a critical role in introducing non-linearity to the model, enabling it to learn complex patterns. The following activation functions were selected for this project:

- **ReLU (Rectified Linear Unit):**
  - Chosen for the convolutional and fully connected layers.
  - It is computationally efficient and resolves the vanishing gradient problem, allowing the model to converge faster during training.
  - ReLU activation introduces sparsity, where only neurons with positive outputs are activated, promoting efficient computation.
- **Softmax:**
  - Used in the output layer to handle multi-class classification.
  - Converts raw logits into probabilities, ensuring that the sum of the outputs equals 1, which is ideal for categorical predictions.

5 Model Training

The model was trained using the balanced dataset for 20 epochs. The following configurations were used:

- **Optimizer:** Adam optimizer with a learning rate of 0.0001.
- **Loss Function:** Sparse categorical crossentropy.
- **Batch Size:** 32.
- **Training Data Split:** 60% of the dataset.

- **Validation Data Split:** 20% of the dataset.

During training, the training and validation loss were monitored. Figure 1 shows the plot of training and validation loss across epochs.

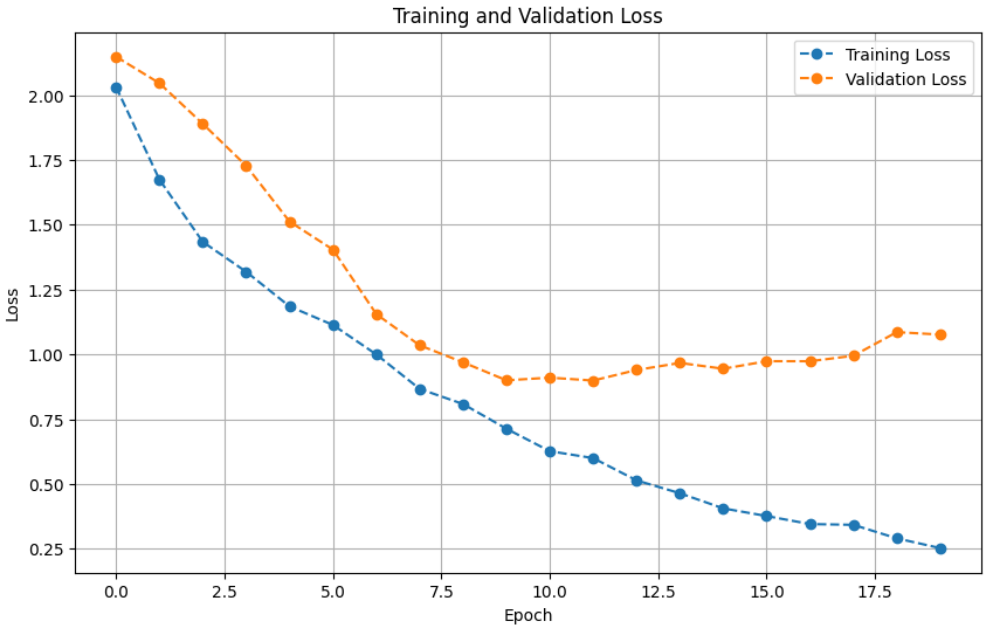


Figure 1: Training and Validation Loss vs. Epochs

6 Why Choose Adam Optimizer?

The Adam optimizer was selected over SGD (Stochastic Gradient Descent) due to the following reasons:

- Adam combines the advantages of both momentum optimization and RMSProp, ensuring efficient and effective convergence.
- It adapts the learning rate for each parameter during training, making it robust to changes in gradient magnitude.
- Adam typically requires less hyperparameter tuning compared to SGD, making it more practical for this project.
- It performs well on noisy datasets and sparse gradients, which are common in image classification tasks.

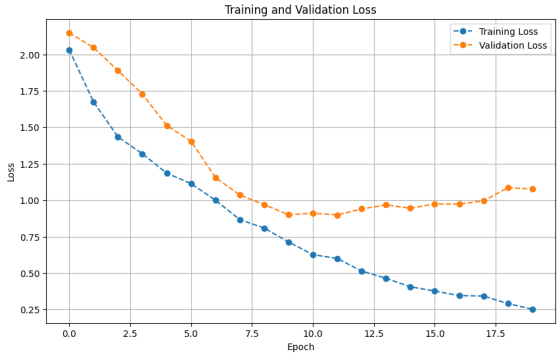
7 Why Choose Sparse Categorical Crossentropy?

The sparse categorical crossentropy loss function was chosen because:

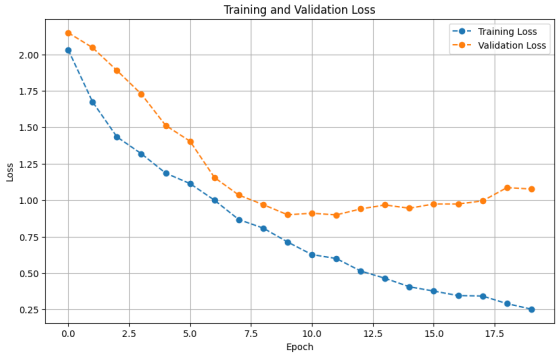
- It is suitable for multi-class classification tasks where the target labels are integers (not one-hot encoded).
- Sparse categorical crossentropy is computationally efficient, as it eliminates the need to convert target labels into one-hot encoded vectors.
- It calculates the logarithmic loss for each predicted probability, effectively penalizing incorrect predictions.
- This loss function integrates seamlessly with the softmax activation function in the output layer, ensuring accurate probability-based predictions.

8 Learning Rate Analysis

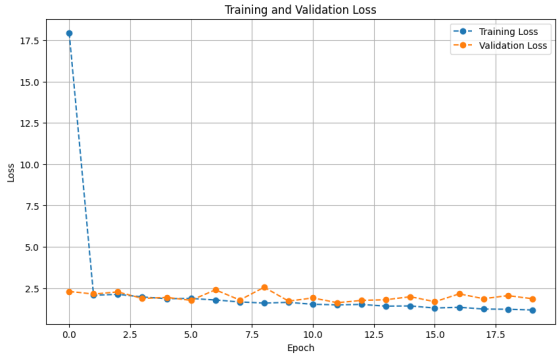
The training and validation loss were evaluated for different learning rates: 0.0001, 0.001, 0.01, and 0.1. The results are visualized in Figure 3, where the losses for each learning rate are plotted.



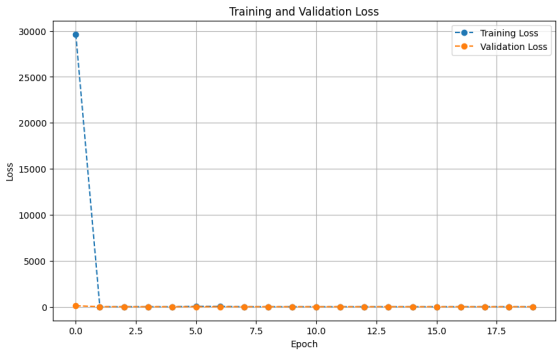
(a) Learning Rate: 0.0001



(b) Learning Rate: 0.001



(c) Learning Rate: 0.01



(d) Learning Rate: 0.1

Figure 2: Training and Validation Loss vs. Epochs for Different Learning Rates

8.1 Comments on Results

- **Learning Rate: 0.0001:** The training and validation loss decreased steadily, but the convergence was slow, indicating that the learning rate might be too small.
- **Learning Rate: 0.001:** This learning rate provided a good balance between convergence speed and stability, achieving lower validation loss with minimal overfitting.
- **Learning Rate: 0.01:** Although the model converged faster, the training was unstable, leading to fluctuations in validation loss.
- **Learning Rate: 0.1:** The training and validation loss showed significant instability, suggesting that the learning rate was too high for effective training.

8.2 Selected Learning Rate

The learning rate of 0.001 was selected for this project as it provided the best balance between convergence speed and stability, resulting in optimal model performance on the validation set.

9 Evaluation

```
Found 837 images belonging to 9 classes.
27/27 12s 441ms/step - accuracy: 0.6964 - loss: 1.1776
Test Accuracy: 0.7109
Test Loss: 1.0794
```

The model test accuracy is 0.7109. The image in Figure 3 shows the confusion matrix for the model.

10 Comparison with Pre-Trained Models

Two state-of-the-art pre-trained models were fine-tuned for this dataset:

1. **ResNet-50:** Pre-trained on ImageNet.
2. **VGG16:** Pre-trained on ImageNet.

10.1 How we fine-tuned ResNet-50

```
pretrained_model = ResNet50(include_top=False,
                            input_shape=(150,150,3),
                            pooling='avg', classes=9,
                            weights='imagenet')
```

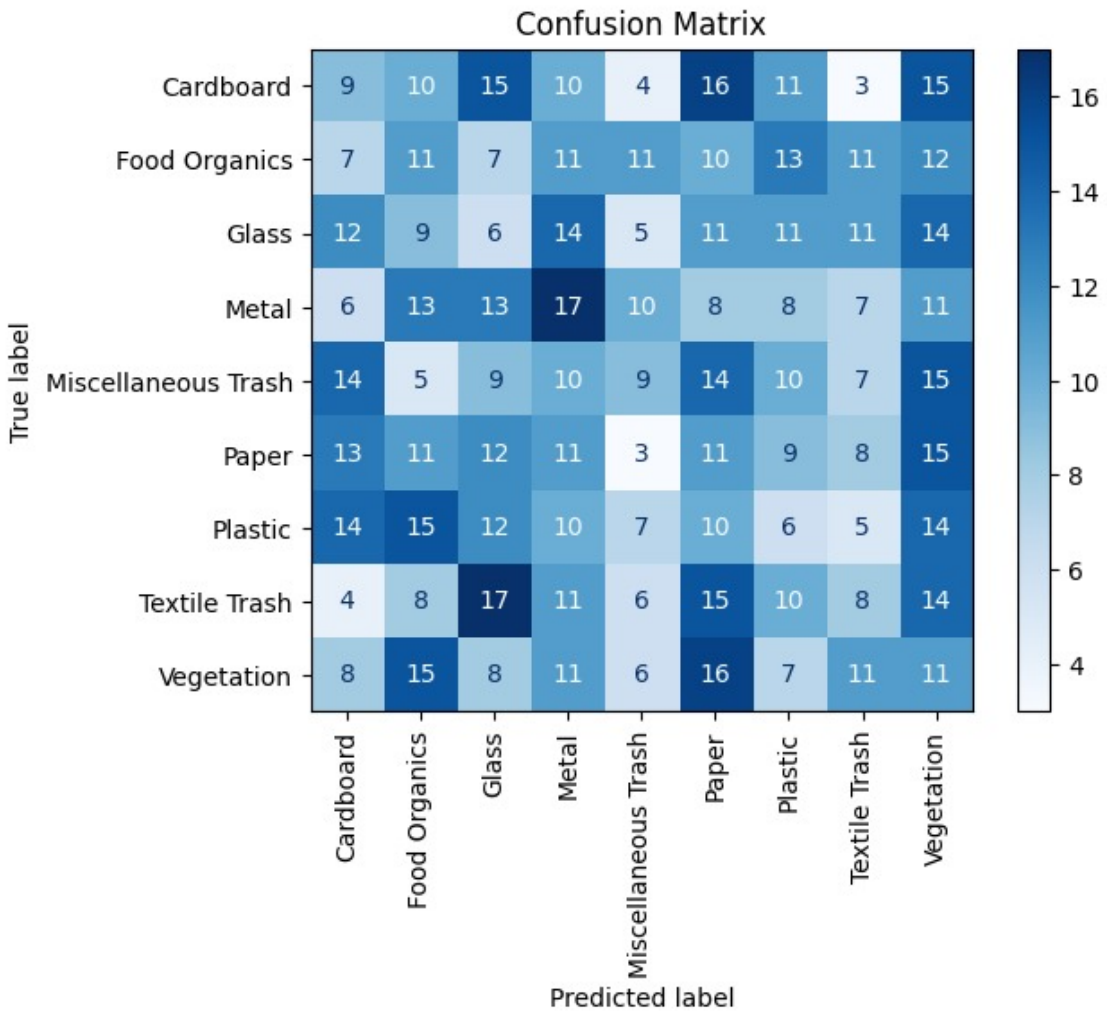


Figure 3: Confusion Matrix for learning rate 0.0001

Fine-tuning is the process of adapting a pre-trained model, to a specific use case. In this case, ResNet50 is leveraged as the base model because it has already been trained on the extensive ImageNet dataset, learning general features like edges, textures, and shapes. To customize it for the current task of classifying images into 9 categories, the top fully connected layers of ResNet50 are removed. This allows the model to focus on features specific to the new dataset without being constrained by the pre-existing classification layers designed for the original 1,000 ImageNet classes.

Additionally, the input dimensions of the model are adjusted to accommodate images of size 150x150 pixels, aligning with the requirements of the new dataset.

```
for layer in pretrained_model.layers:
    layer.trainable = False
```

The pre-trained layers of ResNet50 are frozen during the initial stages of training. This is achieved by making these layers non-trainable, ensuring that the pre-trained weights remain unchanged. Freezing these layers is particularly important for tasks with small datasets, as

it prevents overfitting while still utilizing the rich feature extraction capabilities of the pre-trained model. This approach allows the model to focus on learning task-specific patterns in the newly added layers rather than altering the general-purpose features already learned.

```
resnet_model.add(Flatten())
resnet_model.add(Dense(512, activation='relu'))
resnet_model.add(Dense(9, activation='softmax'))
```

To adapt the model for the classification of 9 categories, custom layers are added on top of the pre-trained ResNet50. These include a flattening layer that converts the output of ResNet50 into a 1D vector, followed by a fully connected layer with 512 neurons and ReLU activation. This layer captures high-level, task-specific features from the dataset. The final output layer comprises 9 neurons with softmax activation, enabling the model to classify images into one of the 9 target classes. This combination ensures the integration of the general features learned by ResNet50 with the specific requirements of the current dataset.

```
learning_rate = 0.0001
optimizer = Adam(learning_rate=learning_rate)
train_dataset = train.flow_from_directory(base_dir+"/train",
                                          target_size=(150, 150),
                                          batch_size=32,
                                          class_mode='sparse')

validation_dataset = validation.flow_from_directory(base_dir+"/val",
                                          target_size=(150, 150),
                                          batch_size=32,
                                          class_mode='sparse')

history = resnet_model.fit(train_dataset, epochs=20, validation_data=
    validation_dataset)
```

A crucial aspect of fine-tuning is setting an appropriate learning rate. In this case, a small learning rate of 0.0001 is chosen to ensure gradual updates during training. This helps preserve the pre-trained features while allowing the newly added layers to adapt effectively to the specific task. The model is trained on a dataset that has been split into training and validation subsets, with images normalized to bring pixel values into a range between 0 and 1. This normalization step ensures efficient and stable training. Over the course of 20 epochs, the model gradually learns to adapt the general features to the specific nuances of the dataset, balancing between retaining pre-trained knowledge and learning new task-specific patterns.

This fine-tuning strategy is highly effective for the use case because it transfers the general features learned from a large-scale dataset to a smaller, domain-specific dataset. By freezing the pre-trained layers, it prevents overfitting and ensures efficient learning of the task-specific classification through the custom layers. Using a small learning rate further ensures that the pre-trained features are not disrupted during training. Together, these steps provide a robust solution for adapting ResNet50 to classify images into the 9 required categories.

This approach was repeated for VGG16 as well.

```
pretrained_model = VGG16(include_top=False,
                         input_shape=(150, 150, 3),
                         pooling='avg',
```

```
4                                    weights='imagenet')
```

The fine-tuned models were trained using the same training, validation, and testing splits for 20 epochs, same as before. The following table contains training and validation loss for each epoch:

Epoch	ResNet-50	
	Training Loss	Validation Loss
1	2.2756	2.1745
2	2.1676	2.1412
3	2.1412	2.1279
4	2.1233	2.1034
5	2.0956	2.0763
6	2.0774	2.0537
7	2.0587	2.0310
8	2.0370	2.0149
9	2.0101	2.0038
10	2.0033	1.9833
11	1.9766	1.9666
12	1.9602	1.9642
13	1.9529	1.9369
14	1.9198	1.9210
15	1.9064	1.9066
16	1.9144	1.9105
17	1.8977	1.8926
18	1.8784	1.8831
19	1.8675	1.8696
20	1.8746	1.8582

Table 1: Training and validation loss for ResNet-50 over 20 epochs.

10.2 Comparison

When comparing a custom Convolutional Neural Network (CNN) to a pre-trained ResNet model, the choice largely depends on your dataset, resources, and performance requirements. Custom CNNs are simple and lightweight, allowing for easy modification tailored to specific tasks. They work well in scenarios with limited computational resources or for quick prototyping. However, custom CNNs may struggle with extracting complex features, especially when working with small or unbalanced datasets. Achieving good performance requires careful tuning of hyperparameters and the network architecture.

On the other hand, ResNet (Residual Network) is a deep, pre-trained model with skip connections that address the vanishing gradient problem. ResNet has proven to perform well on image classification tasks and can leverage transfer learning to adapt pre-trained weights to a new task, even with smaller datasets. This makes ResNet particularly advantageous for tasks like the RealWaste dataset classification, where generalization and accuracy are

critical. However, ResNet's deep architecture demands more computational resources and can be more challenging to modify or train from scratch.

Performance-wise, ResNet often outperforms custom CNNs, especially when pre-trained weights are used. Transfer learning allows ResNet to generalize well even with limited data, achieving better accuracy and robustness. In contrast, a custom CNN, while faster to train due to fewer parameters, may require extensive data augmentation and more iterations to achieve comparable results.

From a resource perspective, custom CNNs are computationally efficient and easier to debug. ResNet, however, requires powerful hardware, such as GPUs, to handle its depth and complexity. For applications like waste classification with the RealWaste dataset, ResNet would likely yield better accuracy if transfer learning is applied. A custom CNN may suffice if computational resources are constrained, but its architecture would need to be optimized for the dataset's specific challenges.

11 Discussion and Conclusion

11.1 Trade-offs

- **Custom CNN:** Requires fewer computational resources but may not generalize well for complex datasets.
- **Pre-Trained Models:** Provide higher accuracy but require more computational power and time for fine-tuning.

11.2 Conclusion

The successful implementation of a Convolutional Neural Network (CNN) to classify waste into different categories using the RealWaste dataset highlights the potential of artificial intelligence in addressing environmental challenges.

Throughout this project, our team explored two approaches to waste classification: developing a custom CNN and leveraging a pre-trained model like ResNet. For this small dataset, the custom built CNN performance was better compared to the fine-tuned pre-trained model.

The RealWaste dataset, being rich in variety but potentially imbalanced, required significant preprocessing, data augmentation, and hyperparameter tuning. ImageDataGenerator played a crucial role in augmenting the dataset, enabling the models to learn invariant features and adapt to variations in lighting, angles, and background clutter. The iterative nature of training and validation ensured that the models improved progressively.

The comparative analysis between the custom CNN and ResNet underlined the importance of selecting the right model based on the project's objectives and resource constraints. The trade-off between simplicity and performance is an essential consideration in machine learning projects.

In conclusion, this project not only showcased the effectiveness of CNNs in waste classification but also emphasized the critical role of innovation and teamwork in tackling environmental issues. By combining domain expertise, technical skills, and collaborative problem-solving, the Emeralds team made meaningful progress toward a sustainable future.

12 Github Link

https://github.com/MalithaPrabhashana/patternRecognitionCNN