**Machine Learning Project Documentation**

**Model Refinement**

**1. Overview**

The model refinement phase plays a crucial role in enhancing the performance of our machine learning model. This step involves identifying areas where the model underperforms and applying techniques such as hyperparameter tuning, algorithm adjustments, and feature selection to improve its accuracy and generalization capabilities.

**2. Model Evaluation**

Initially, we used ResNet50, a pretrained deep learning model, for this project. However, the model achieved an accuracy of only 28%, which was below acceptable levels. We identified that ResNet50 struggled to generalize due to the complexity of the dataset and the mismatch between the problem and the pretrained model's architecture. This prompted a shift towards a custom Convolutional Neural Network (CNN), which improved performance significantly.

Key Metrics from Initial Evaluation:

* ResNet50 Accuracy: 28%
* CNN Accuracy (pre-tuning): 62%

Visualizations such as confusion matrices and training-validation loss curves provided insights into where the model was underperforming.

**3. Refinement Techniques**

To refine the CNN model, we applied the following techniques:

* **Hyperparameter Tuning:** Adjusted filters, kernel size, learning rate, dropout rate, and dense units using SearchCV and Keras Tuner.
* **Data Augmentation:** Enhanced the training dataset by applying transformations such as rotation, zoom, and horizontal flips to prevent overfitting.
* **Custom Architecture:** Designed a CNN model tailored to the dataset, including layers optimized for extracting features from waste classification images.

**4. Hyperparameter Tuning**

We performed hyperparameter tuning using Keras Tuner to explore combinations of:

* Filters: [32, 64, 128]
* Kernel Sizes: [3, 5]
* Dropout Rates: [0.2, 0.3, 0.5]
* Dense Units: [64, 128, 192,256]
* Learning Rates: [0.01, 0.001, 0.0001]

Best Hyperparameters:

* Filters: 128
* Kernel Size: 5
* Dropout Rate: 0.3
* Dense Units: 192
* Learning Rate: 0.01

Impact of Tuning:

* Validation Accuracy decreased from 62% to %62.50.
* Training time was optimized by reducing unnecessary computations.

**5. Cross-Validation**

Initially, we used an 80-20 train-validation split. After refinement, We're using **image augmentation** (via ImageDataGenerator), which effectively creates more variability in the dataset, reducing the need for cross-validation.

**6. Feature Selection**

While this project primarily relied on image data, we ensured that all features (pixel values) were normalized to a range of [0, 1]. Feature extraction was handled by the convolutional layers, and no manual feature selection was performed.

**Test Submission**

**1. Overview**

The test submission phase involved evaluating the refined model on a held-out test dataset to assess its generalization capability. The refined CNN model was prepared for deployment and evaluated on unseen data.

**2. Data Preparation for Testing**

The test dataset was prepared by:

* Rescaling pixel values to [0, 1].
* Ensuring no overlap with training and validation datasets.
* Maintaining class distribution similar to the training set.

**3. Model Application**

The trained CNN model was applied to the test dataset using the following code:

# Evaluate the model on the test set

test\_loss, test\_accuracy = model.evaluate(test\_generator)

print(f"Test Accuracy: {test\_accuracy:.4f}")

**4. Test Metrics**

* Test Accuracy: 59%
* Test Loss: 1.2

Comparison of Metrics:

* Training Accuracy: 62%
* Validation Accuracy: 59%

The performance on the test dataset was consistent with training and validation metrics, indicating good generalization.

**5. Model Deployment**

While deployment was not fully implemented, the model is prepared for integration into a real-world system. Future deployment steps include:

* Building a user interface for end-user interaction.

**6. Code Implementation**

Below is a snippet for deploying the trained model:

import tensorflow as tf

# Save the model

model.save('garbage\_classification\_model.h5')

# Load the model for deployment

deployed\_model = tf.keras.models.load\_model('garbage\_classification\_model.h5')

**Conclusion**

The CNN model outperformed the initial ResNet50 approach, achieving an accuracy of 62% on the training set and 59% on the test set after hyperparameter tuning and refinement. While the results are promising, further improvements can be made by:

* Exploring ensemble methods.
* Increasing the dataset size.
* Incorporating more advanced architectures like EfficientNet.

This project demonstrates the importance of model refinement and provides a solid foundation for future work.

**References**

1. TensorFlow Documentation: [https://www.tensorflow.org](https://www.tensorflow.org/)
2. Keras Tuner Documentation: <https://keras.io/keras_tuner/>
3. ResNet50 Paper: <https://arxiv.org/abs/1512.03385>