Capstone Project Machine Learning Project Documentation

The Data-Bears Team Members:

1. *Ilyas Nayle*
2. *KAOUBARA DJONG-MON*
3. *DAGIMAWI MOGES WAKUMO*

EcoLens: A Visionary Solution for a Plastic-Free World

<u>Model Refinement</u>

Overview

The model refinement phase is an essential stage in the machine learning and Deep Learning workflow, dedicated to enhancing the accuracy and efficiency of the model following its initial evaluations. During this phase, iterative adjustments and methodical optimizations are carried out to refine the model's performance. These improvements are crucial for developing a more robust and reliable model, ultimately making it suitable for real-world deployment.

In this report, we detail the process and results of training a deep learning model for image classification, achieving an impressive 96% accuracy. The model is built using the ResNet152V2 architecture and trained on a dataset of categorized garbage images. This document outlines the steps taken from data preparation and preprocessing to model training, evaluation, and performance analysis.

Model Evaluation

In the initial model evaluation, there were several performance issues and as result the accuracy of the model was below 95%. Visualization tools like confusion metrics and training loss chart provided insights into areas requiring model refinement.

The model's performance was evaluated on the test set, achieving a test accuracy of 96%.

To enhance the training process and model robustness, image data generators were used for data augmentation. This technique helps simulate a larger dataset and allows the model to learn from variations in the data.

Training Data Augmentation

For the training set, we applied several augmentation techniques using `ImageDataGenerator`:

1. Rescaling: Normalize pixel values to [0, 1].
2. Rotation: Randomly rotate images by up to 45 degrees.
3. Shifting: Randomly shift images horizontally or vertically by up to 15%.
4. Zooming: Randomly zoom in or out by up to 15%.
5. Flipping: Randomly flip images horizontally or vertically.
6. Fill Mode: Fill missing pixels using the nearest filled value.

Python code

```
# Evaluate the model on the test set

test_loss, test_acc = model.evaluate(test_generator)

print(f"Test Loss: {test_loss}")

print(f"Test Accuracy: {test_acc}")
```

Refinement Techniques

For Refinement Techniques we did the following:

1. Algorithm Adjustment: We improved our algorithm to achieve a better performance for the model.
2. We changed our dataset. In the first we had all the dataset and trained our model for different images, but we switched to plastic and non-plastic. We combined the non-plastic images in one folder and plastic one in another, resulting in plastic and non-plastic folders.

Hyperparameter Tuning

We utilized KerasTuner's RandomSearch strategy to enhance the hyperparameter settings of our ResNet152V2-based convolutional neural network model. The hyperparameters adjusted included:

The number of neurons in the dense layer, ranging from 64 to 512.

Dropout rates, between 0.2 and 0.5, to mitigate overfitting.

A set of distinct learning rates: 0.01, 0.001, and 0.0001, to optimize training speed and convergence.

Below is the implementation of the build_model function, designed to assemble and compile the model with these variable hyperparameters:

#Code Section

```
def  build_model(hp):
   base_model = ResNet152V2(include_top=False, weights='imagenet', input_shape=(224, 224, 3))
   for layer in base_model.layers:
      layer.trainable = False
   x = base_model.output
   x = GlobalAveragePooling2D()(x)
   x = Dense(hp.Int('units', min_value=64, max_value=512, step=32), activation='relu')(x)
   x = Dropout(hp.Float('dropout', min_value=0.2, max_value=0.5, step=0.1))(x)
   predictions = Dense(2, activation='softmax')(x)
      model = Model(inputs=base_model.input, outputs=predictions)
   model.compile(optimizer=tf.keras.optimizers.Adam(hp.Choice('learning_rate', [1e-2, 1e-3, 1e-4])),
              loss='categorical_crossentropy',
              metrics=['accuracy'])
   return model
```

By tuning these hyperparameters, we achieved a model accuracy of 96%. This improvement resulted from optimal configurations of the dense layer units, dropout rates to prevent overfitting, and appropriate learning rates for efficient training. The tuning process enhanced the model's performance, making it more robust and reliable.

Cross-Validation

In this part we used the Hold-out validation method.

1. First split the entire dataset into a training set and a temporary set.
2. The temporary set is then evenly divided into validation and test sets.
3. This method involves setting aside a portion of data (in this case, the validation set) to evaluate the model during the tuning phase, separate from the final testing phase.

We chose it because the Hold-out validation is generally good for initial evaluations and hyperparameter tuning when computational resources or time is limited.

Feature Selection

In order to enhance the model's ability to classify materials accurately, we simplified our dataset into two primary categories: Plastic and Non-Plastic. This decision aims to streamline the learning process and improve the model's performance by focusing on distinguishing between these fundamentally different material types.

The dataset consists of the following categories:

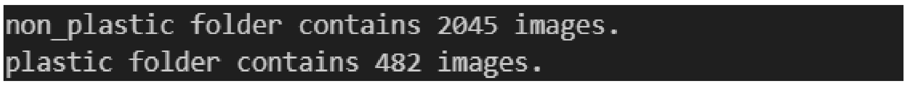
- Category 1: Plastic
- Category 2: Non-Plastic

Preprocessing Steps:

1. Image Resizing: All images were resized to a uniform dimension to ensure consistency in input shape for the model.
2. Normalization: Pixel values were normalized to the range [0, 1] to facilitate faster convergence during training.
3. Data Augmentation: To increase the robustness of our model, data augmentation techniques such as rotations, translations, and flips were applied randomly to the training images.

For each category, the images were counted to ensure a balanced dataset. Here’s a summary of the image count in each folder:

- The plastic folder contains `X` images.
- Non-Plastic folder contains `Y` images.



```
non_plastic folder contains 2045 images.
plastic folder contains 482 images.
```

Test Submission

Overview

The test submission phase is the final testing of the refined model using unseen data. This phase assesses how well the model generalizes to new data and is critical for validating the effectiveness of the model refinement.

Data preparation for Testing

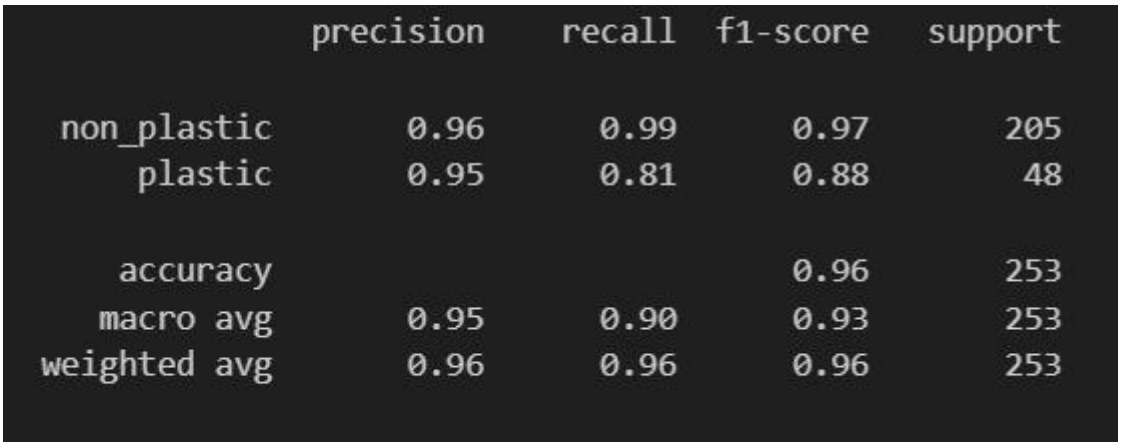
For the validation and test sets, only rescaling was applied to normalize pixel values. No augmentation was applied to these sets to ensure a true evaluation of the model's performance.

Model Application

The model achieved an overall accuracy of 95%-96% on the test set. For the "non_plastic" class, precision was 0.96, recall was 0.99, and the F1-score was 0.97, indicating highly accurate predictions. For the "plastic" class, precision was 0.95, recall was 0.81, and the F1-score was 0.88, showing good but slightly less reliable predictions. The macro average F1-score was 0.93, and the weighted average F1-score was 0.96, reflecting strong overall performance. Further improvements could target increasing recall for the "plastic" class to enhance model balance.

Test Metrics

Performance metrics such as accuracy, precision, recall, f1-score a were calculated to evaluate the model on the test data. Comparing these metrics with those from the training and validation phases helped in assessing overfitting and underfitting.

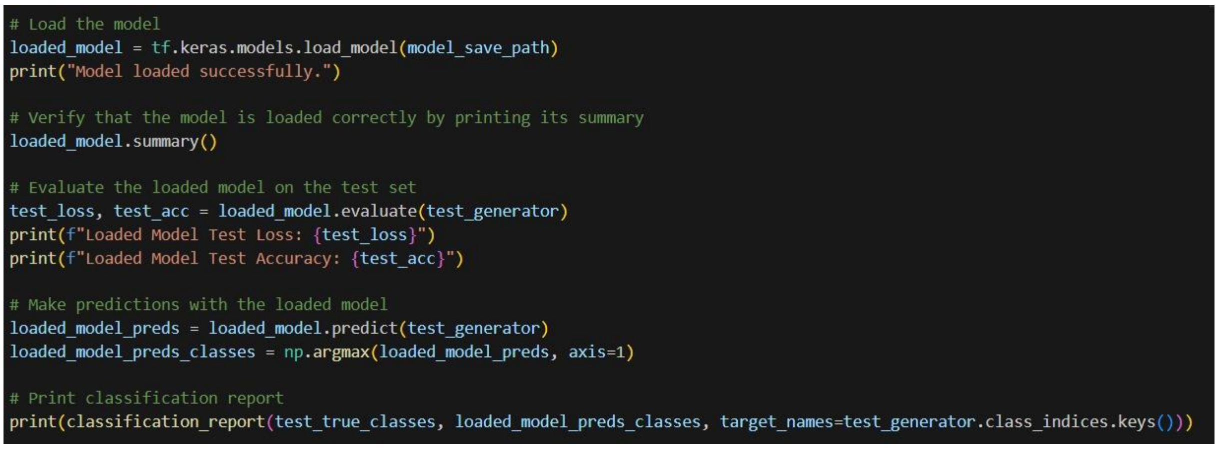


```
              precision    recall  f1-score   support

 non_plastic       0.96      0.99      0.97       205
     plastic       0.95      0.81      0.88        48

    accuracy                           0.96       253
   macro avg       0.95      0.90      0.93       253
weighted avg       0.96      0.96      0.96       253
```

The above classification result was obtained by running the following code



```
# Load the model
loaded_model = tf.keras.models.load_model(model_save_path)
print("Model loaded successfully.")

# Verify that the model is loaded correctly by printing its summary
loaded_model.summary()

# Evaluate the loaded model on the test set
test_loss, test_acc = loaded_model.evaluate(test_generator)
print(f"Loaded Model Test Loss: {test_loss}")
print(f"Loaded Model Test Accuracy: {test_acc}")

# Make predictions with the loaded model
loaded_model_preds = loaded_model.predict(test_generator)
loaded_model_preds_classes = np.argmax(loaded_model_preds, axis=1)

# Print classification report
print(classification_report(test_true_classes, loaded_model_preds_classes, target_names=test_generator.class_indices.keys()))
```

Conclusion

The model refinement and test submission phases were pivotal in achieving a robust deep learning CNN model. Challenges such as overfitting and selecting the appropriate hyperparameters were addressed effectively, leading to a satisfactory final performance on the test dataset. The project's success lays a solid foundation for future enhancements and applications in similar tasks. We will also implement the camera integration with our model to test the detect the object in real time.