Computer Vision
Assignment 01

Transfer Learning with Cassava

Melchor Lafuente
Iker Jauregui

Contents

1 Introduction

The aim of this project is to determine whether a cassava leaf is diseased and, if so, identify the type of disease: Cassava Bacterial Blight (CBB), Cassava Brown Streak Disease (CBSD), Cassava Green Mottle (CGM) and Cassava Mosaic Disease (CMD).

Figure 1: Random samples for each class proceding from predefined train split

2 Exploratory Data Analysis (EDA)

An Exploratory Data Analysis was performed to understand the dataset characteristics and guide model development. The following key observations were made:

2.1 Dataset Characteristics

- All images have uniform dimensions of 800×600 pixels.
- The CMD class is highly imbalanced along all three predefined splits (Figure 2, Figure 3 and Figure 4).

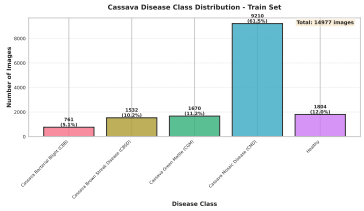


Figure 2: Class distribution on Train set.

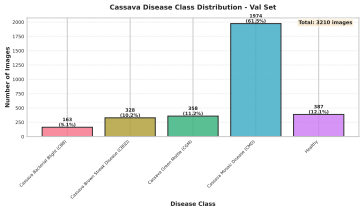


Figure 3: Class distribution on Val set.

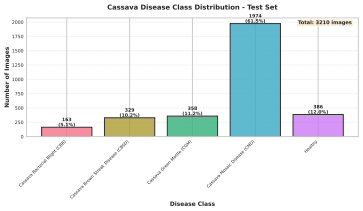


Figure 4: Class distribution on Test set.

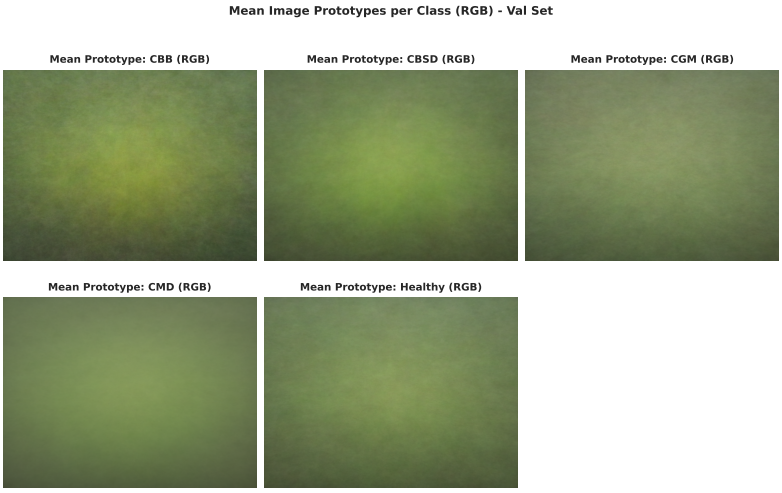
2.2 Class Prototypes Analysis

Class prototypes were computed using median aggregation method (Figure 5). At first sight, we could think that there aren't any differences between the different prototypes, but if we look closely we can observe some distinctive visual characteristics for each class:

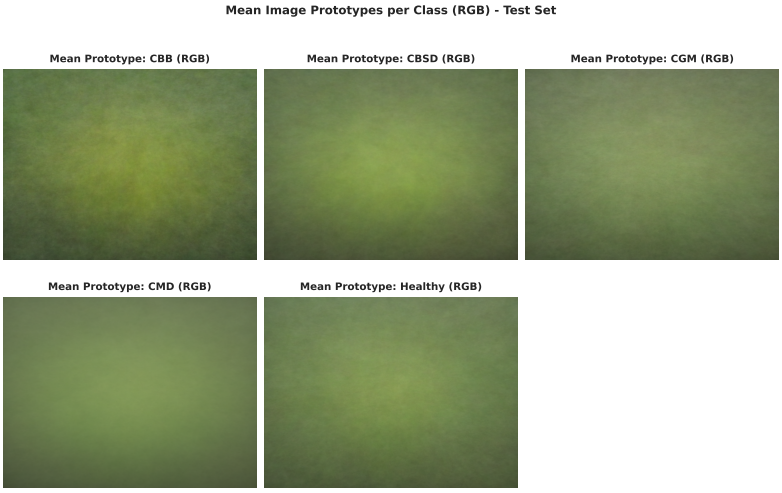
- Both **CBB** and **CBSD** classes seem to be lighter. That makes sense because those diseases color the surface of the leaves with yellow.
- The **CGM** class is a little more brownish than the others and that may be because that disease affects not only to the leaves but also to the roots of the plant.
- The **CMD** disease seems to have the smoothest texture among all classes.

There aren't notable differences between the prototypes generated over the different splits, so we can say that these observations maintain among the three sets.

(a) Mean prototypes of Train set.



(b) Mean prototypes of Val set.



(c) Mean prototypes of Test set.

Figure 5: Mean image prototypes for each class among Train, Val and Test splits.

3 Experimental Setup

3.1 Dataset Splitting

The dataset was divided into training, validation, and test sets using the predefined splits stated at the metadata file *cassava_split.csv*, but the imbalance problem was corrected by random downsampling the CMD class for Train and Val sets (Test set was kept unmodified).

Set	CBB samples	CBSD samples	CGM samples	CMD samples	Healthy samples
Train	761	1532	1670	2000	1804
Val	163	328	358	400	387
Test	163	328	358	1974	387

Table 1: Dataset splits used during the experiments.

3.2 Callbacks and LR Scheduler

During the experiments, two main callbacks were used: **EarlyStopping** and **ModelCheckpoint**. For batch size selection, an initial **BatchSizeFinder** was used and it suggested a value of 2048 samples. That size allocated almost the whole memory of the GPU (50GB) so we finally used a batch size of 1024. Finally, instead of using a LearningRateFinder, we decided to employ a learning rate scheduler. We decided to use the **ReduceLROnPlateau** scheduler, as it could lead the models to obtain the best possible validation losses.

3.3 Model Architecture and Evaluation Metric

As the main purpose of this study is to analyze the obtained results after applying different transfer learning techniques, we decided to fix the model architecture to a **ResNet18**. It's a modern and a powerful enough model, but it is one of the smallest (in terms of layer depth) architectures, so it could let us perform different experiments in a reasonable time.

For the evaluation metric, we selected the **F1 Score (macro)**, as we wanted to measure the model's performance over the unbalanced test set.

4 Experiments

4.1 Experiment 1: Model Trained from Scratch

In this first experiment we just trained the model from scratch, without applying any transfer learning technique. It achieved a F1 score (macro) of **0.4639**.

Results:

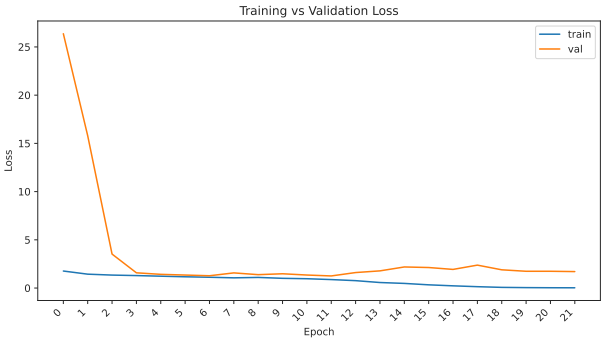


Figure 6: Loss curves for the not pretrained ResNet18 model.

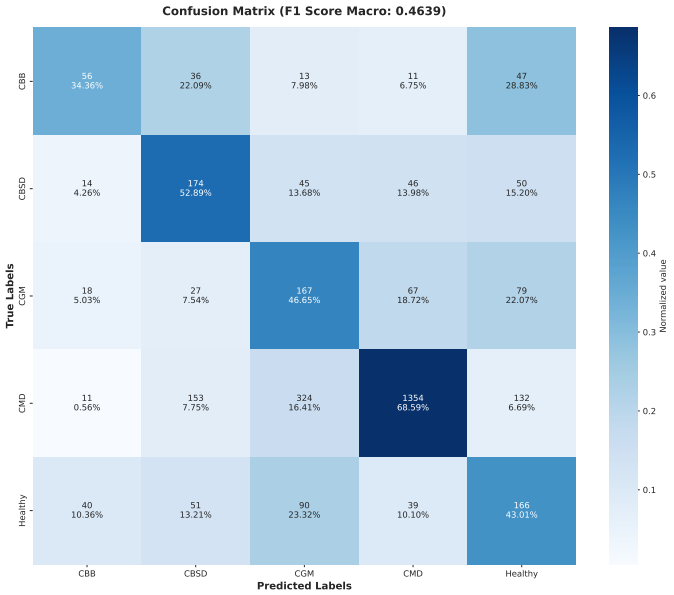


Figure 7: Confusion matrix over Test set for the not pretrained ResNet18 model.

4.2 Experiment 2: Transfer Learning with Feature Extraction

In this experiment, we used pretrained weights from ImageNet (ResNet18_Weights.IMAGENET1K_V1) and applied the *Feature Extraction* transfer learning technique. We can observe how this time the validation loss reached lower values than on the previous experiment (Figure 6 and Figure 8). Moreover, the model achieved a F1 score value of **0.5558** (0.0919 points more respect to the not pretrained model). These results state that the model took advantage from the prelearned kernels.

Results:

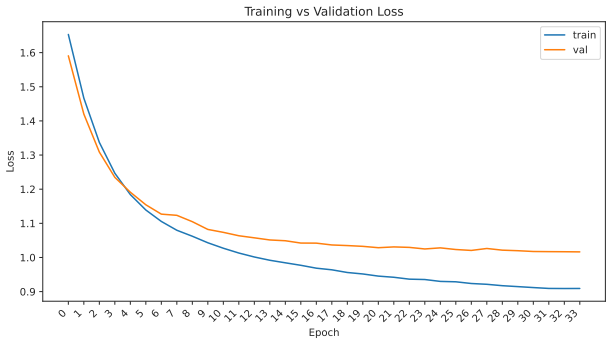


Figure 8: Loss curves for the ResNet18 model pretrained with feature extraction technique.

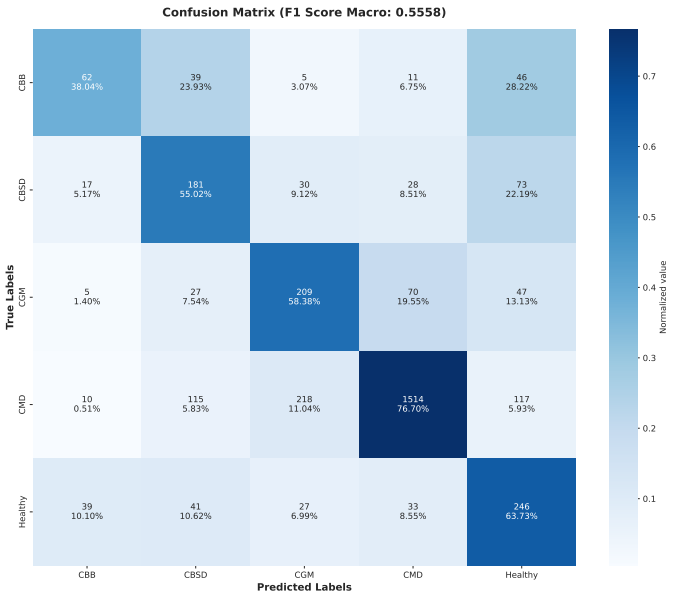


Figure 9: Confusion matrix over Test set for the ResNet18 model pretrained with feature extraction technique.

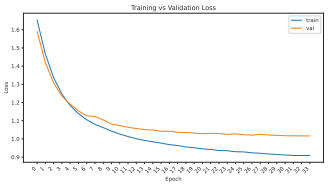
4.3 Experiment 3: Fine-Tunning with Gradual Unfreezing

In this experiment, we also used the pretrained weights from ImageNet, but this time we applied a different transfer learning technique: Fine-tunning with gradual unfreezing. This gradual unfreezing was performed per blocks, starting from the classification head and finishing at the input layer of the network (Table 2). We generated a loss curve graph each time a block was unfroze. As we can see on the results (Figure 12), the model started to strongly overfit quite soon, just when the second block (the last feature extractor block of the network) was unfroze Even so, it achieved a F1 score of **0.6423** (0.0865 points more respect to the previous experiment).

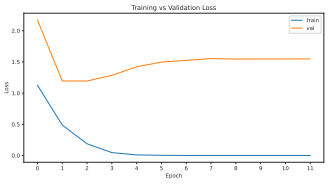
Block	**Layers**
Block #1	fc.weight, fc.bias
Block #2	layer4.0.conv1.weight, layer4.0.bn1.weight, layer4.0.bn1.bias, layer4.0.conv2.weight, layer4.0.bn2.weight, layer4.0.bn2.bias, layer4.0.downsample.0.weight, layer4.0.downsample.1.weight, layer4.0.downsample.1.bias, layer4.1.conv1.weight, layer4.1.bn1.weight, layer4.1.bn1.bias, layer4.1.conv2.weight, layer4.1.bn2.weight, layer4.1.bn2.bias
Block #3	layer3.0.conv1.weight, layer3.0.bn1.weight, layer3.0.bn1.bias, layer3.0.conv2.weight, layer3.0.bn2.weight, layer3.0.bn2.bias, layer3.0.downsample.0.weight, layer3.0.downsample.1.weight, layer3.0.downsample.1.bias, layer3.1.conv1.weight, layer3.1.bn1.weight, layer3.1.bn1.bias, layer3.1.conv2.weight, layer3.1.bn2.weight, layer3.1.bn2.bias
Block #4	layer2.0.conv1.weight, layer2.0.bn1.weight, layer2.0.bn1.bias, layer2.0.conv2.weight, layer2.0.bn2.weight, layer2.0.bn2.bias, layer2.0.downsample.0.weight, layer2.0.downsample.1.weight, layer2.0.downsample.1.bias, layer2.1.conv1.weight, layer2.1.bn1.weight, layer2.1.bn1.bias, layer2.1.conv2.weight, layer2.1.bn2.weight, layer2.1.bn2.bias
Block #5	layer1.0.conv1.weight, layer1.0.bn1.weight, layer1.0.bn1.bias, layer1.0.conv2.weight, layer1.0.bn2.weight, layer1.0.bn2.bias, layer1.1.conv1.weight, layer1.1.bn1.weight, layer1.1.bn1.bias, layer1.1.conv2.weight, layer1.1.bn2.weight, layer1.1.bn2.bias
Block #6	conv1.weight, bn1.weight, bn1.bias

Table 2: Block composition for the gradual unfreezing technique.

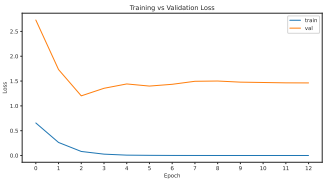
Results:



(a) Frozen until block #1

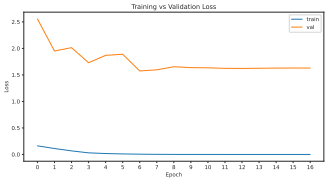


(b) Frozen until block #2



(c) Frozen until block #3

(d) Frozen until block #4



(e) Frozen until block #5

(f) Completely unfrozen

Figure 10: Loss curves for the ResNet18 model trained with the gradual unfreezing technique.

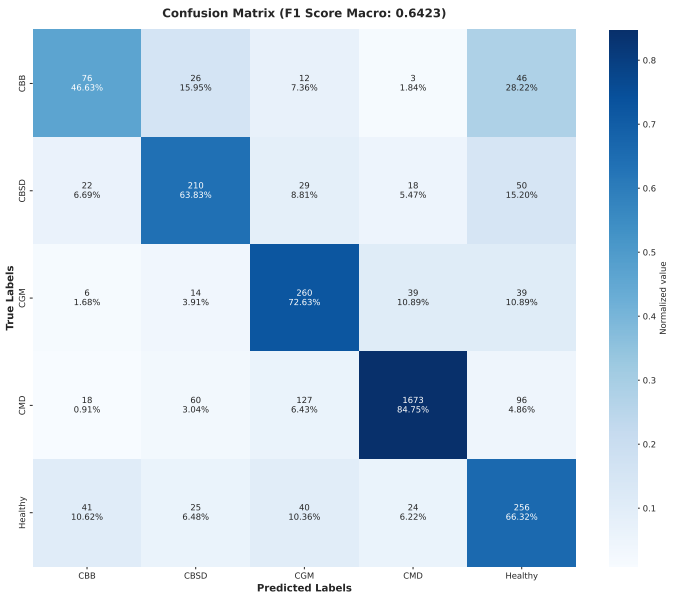
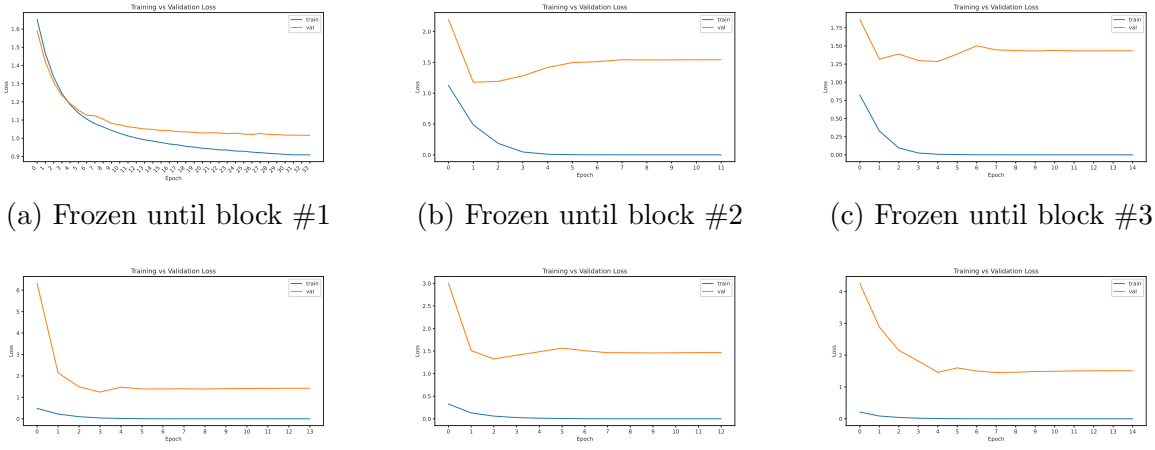


Figure 11: Confusion matrix over Test set for the ResNet18 model trained with the gradual unfreezing technique.

4.4 Experiment 4: Fine-Tunning with Gradual Unfreezing and training subsets

This experiment follows the same gradual unfreezing strategy as before, with a single difference: instead of using the entire training set at each stage, we used different subsets for every newly unfrozen stage. The goal was to introduce additional regularization and mitigate overfitting. As shown in the results (Figure 12), the model still overfits, but the loss curves appear smoother compared to the previous experiment. The final model reached an F1 score of 0.6536, which is an improvement of 0.0113 points over the previous setup.

Results:



(a) Frozen until block #1 (b) Frozen until block #2 (c) Frozen until block #3

(d) Frozen until block #4 (e) Frozen until block #5 (f) Completely unfrozen

Figure 12: Loss curves for the ResNet18 model trained with the gradual unfreezing technique.

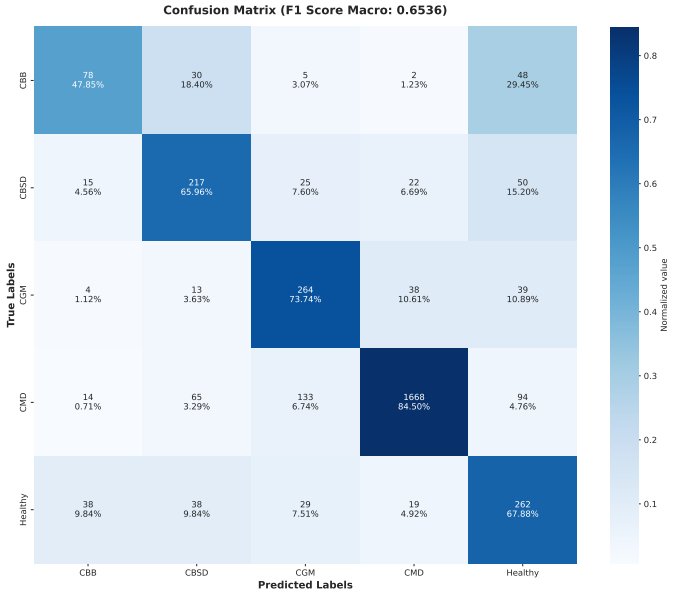


Figure 13: Confusion matrix over Test set for the ResNet18 model trained with the gradual unfreezing technique.

4.5 Experiment 5: Fine-Tunning with Layer-wise Learning Rates

In this experiment, we applied a another fine-tunning technique that uses layer-wise learning rates. Using the defined blocks of the previous experiment, we applied a different learning rate value for each block, using bigger learning rates on the tail of the network and smaller learning rates on the input side (Table 3). Similar to the previous experiment, the model started to overfit in the early training epochs (Figure 14). The obtained F1 score was of **0.6167**, lower than the previous experiment's score but higher than the second one's.

Block	**Learning rate**
Block #1	1e-3
Block #2	1e-4
Block #3	1e-5
Block #4	1e-6
Block #5	1e-7
Block #6	1e-8

Table 3: Used learning rates values among the different blocks of the RestNet18.

Results:

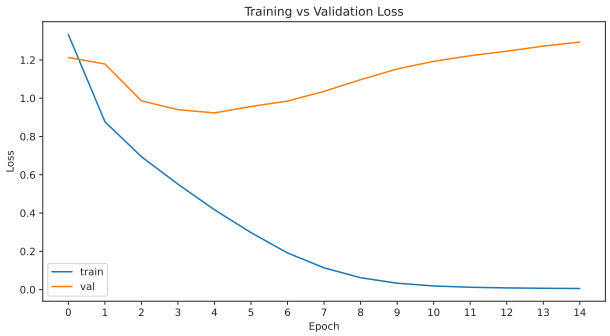


Figure 14: Loss curves for the ResNet18 model trained with layer-wise learning rates technique.

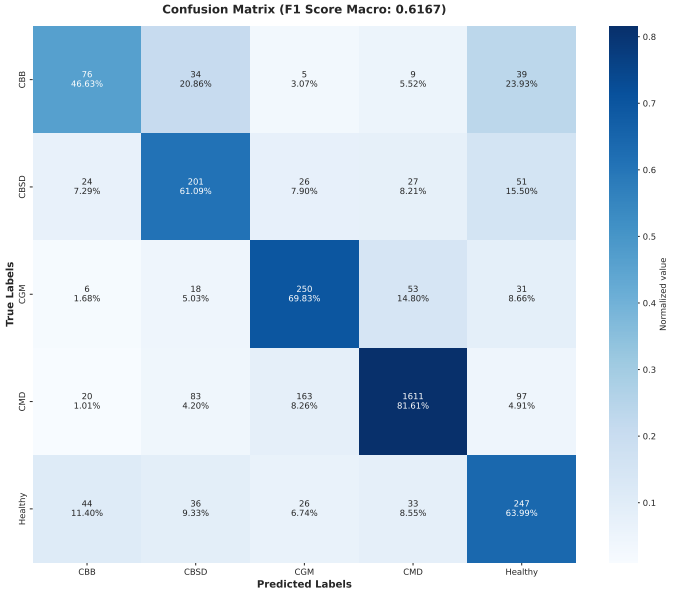


Figure 15: Confusion matrix over Test set for the ResNet18 model trained with the layer-wise learning rates technique.

5 Conclusions

From the experiments carried out, we highlight the following conclusions:

- Gradual unfreezing consistently stabilizes training and produces smoother loss curves compared to full fine-tuning from the start.
- Using subsets of the training data during unfreezing introduces mild regularization, slightly reducing overfitting.
- Despite improvements in curve stability, overfitting remains a challenge across all experiments.
- The best-performing configuration achieved an F1 score of **0.6536**, showing only incremental gains over simpler fine-tuning strategies.
- Variations in data selection during unfreezing appear to influence model robustness more than learning-rate adjustments alone.

6 Future Work

- Evaluate stronger data augmentation techniques to further reduce overfitting.
- Explore advanced regularization methods such as adding dropout layers to the model.
- Test alternative backbone architectures (e.g., EfficientNet, ConvNeXt, ViT).
- Incorporate semi-supervised or self-supervised pretraining to improve feature generalization.