# Kaggle Report on Cassava Disease Classification, May 2020.

**Team**: MTM (Mbaye Babou,Tunde Ajayi, Madiou Diene)

## 1.0 Introduction

The Kaggle competition is an in-house event of AMMI Students, enabling them to put into use what they learnt in computer vision classes. The Students were meant to develop an algorithm that would detect and separate healthy cassava plants from infected ones.

## 1.1 Problem statement/Motivation

Traditional method requires experts (who are limited in numbers given the task) to go to farms to physically diagnose (which is subjective) diseased plants. Hence, the need for a robust algorithm that can overcome the challenges arising from such a task.

## 2.0 Methodology

### 2.1 Architecture and Preprocessing

The data consisted of labelled (4 diseased and 1 healthy class) and unlabelled data. In order to provide good results we tried some pre-trained convolutional neural network models i.e. Resnet101, se_resnet152, se_resnext50_32x4d from Fastai. The final model was the se_resnext50_32x4d which is a network obtained by repeating a building block that aggregates a set of transformations with the same topology. Its architecture is composed of 50 layers with 4 bottlenecks and each layer has cardinality of 32.

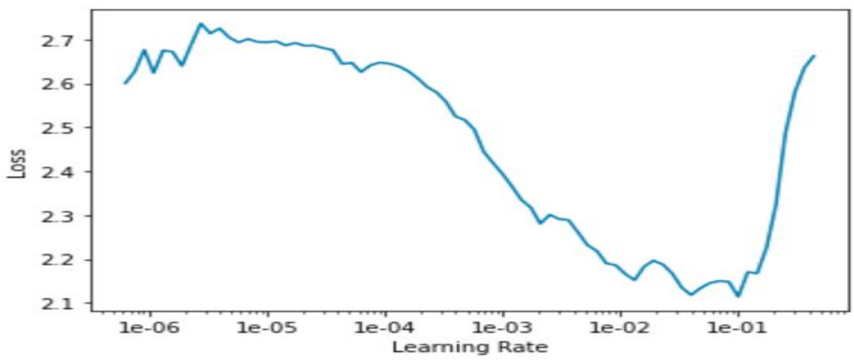
```
def se_resnext50_32x4d(pretrained=False):
    pretrained = 'imagenet' if pretrained else None
    model = pretrainedmodels.se_resnext50_32x4d(pretrained=pretrained)
    return model
```

+ Code + Markdown

```
learner = cnn_learner(data, se_resnext50_32x4d, pretrained=True,
                      cut=-2, split_on=lambda m: (m[0][3], m[1]),metrics = [accuracy])
```

The main libraries we used were pytorch and Fastai (a deep learning framework built on top of pytorch that facilitates ML modeling). Before training the model, we performed data augmentation using functions such as get_transforms and ImageDataBunch we were able to perform the data augmentation. We defined our convolutional model as:

Figure 1.0: Graph vs Learning Rate




**learner = cnn_learner(data, se_resnext50_32x4d, pretrained=True, cut=-2, split_on=lambda m: (m[0][3], m[1]),metrics = [accuracy])**

Then we used the function lr_find() in order to get the best learning rate for our model. lr_find() trained the model with a single batch and gave losses with different learning rates. The best learning rate that minimised the loss was selected.

## 2.2 Training

We use the fit_one_cycle method to train the model. The learner.fit_one_cycle(n, max_lr) method allowed us to train the model with n epochs and different learning rates less than or equal to the max_lr. The following figures 2 and 3 shows the evolution of the train phase:

```
lr=1e-2
learner.fit_one_cycle(20, lr)
```

| epoch | train_loss | valid_loss | accuracy | time |
|---|---|---|---|---|
| 0 | 0.999494 | 0.591107 | 0.814324 | 12:03 |
| 1 | 0.742390 | 0.603771 | 0.801061 | 03:13 |
| 2 | 0.612953 | 0.602696 | 0.825818 | 03:09 |
| 3 | 0.547954 | 0.670500 | 0.805482 | 03:09 |
| 4 | 0.553128 | 0.564566 | 0.854996 | 03:09 |
| 5 | 0.471272 | 0.489575 | 0.867374 | 03:09 |
| 6 | 0.490710 | 0.450932 | 0.862953 | 03:09 |
| 7 | 0.443653 | 0.397223 | 0.871795 | 03:09 |
| 8 | 0.395901 | 0.414141 | 0.871795 | 03:09 |
| 9 | 0.435617 | 0.342107 | 0.890363 | 03:09 |
| 10 | 0.384880 | 0.374383 | 0.885057 | 03:09 |
| 11 | 0.324579 | 0.324239 | 0.890363 | 03:08 |
| 12 | 0.312827 | 0.327431 | 0.895668 | 03:08 |
| 13 | 0.285917 | 0.325776 | 0.895668 | 03:10 |
| 14 | 0.221545 | 0.418395 | 0.903625 | 03:10 |

Figure 2: Training losses

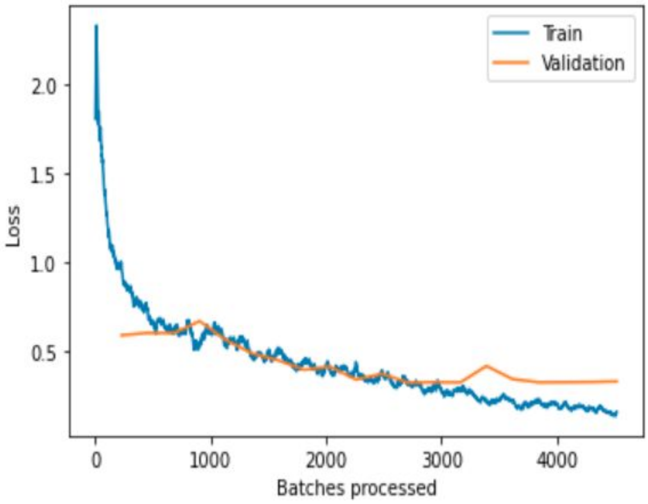



Figure 3: Loss vs Batches processed

## 2.3 Inference

preds,y = learner.TTA(ds_type=DatasetType.Test)

Test-time augmentation (TTA) is the technique we adopted for testing. It is an application of data augmentation to the test dataset.Specifically, it involves creating multiple augmented copies of each image in the test set, having the model make a prediction for each, then returning an ensemble of those predictions.

## 3.0 Results and discussions

After training our model several times, the highest accuracy we got was 91.1258 as a public score on the kaggle leaderboard with a validation of 0.3 in our notebook. By displaying the confusion matrix we were able to see the misclassifications of each category.

```
[('cbb', 'cbsd', 19),('cgm', 'cmd', 16),
 ('cbsd', 'cmd', 13),('cbsd', 'cbb', 12),
 ('cbb', 'cmd', 8),('cgm', 'cbsd', 8),
 ('cmd', 'cgm', 6),('cbsd', 'cgm', 5),
 ('cmd', 'cbsd', 5),('cgm', 'cbb', 4),('cbb', 'cgm', 2)]
```

## 4.0 Conclusion

We noticed that the healthy leaves were well predicted and most of the misclassified leaves were the cbb and cbsd classes. One remarkable result we observed was that there was no diseased leaf predicted as healthy. That means that our model is robust in classifying strictly healthy or diseased plants (if the class of the diseased plant is not considered).

## 5.0 Reference

Ernest Mwebaze et al., 2019, *"iCassava 2019 Fine-Grained Visual Categorization Challenge."*

[https://docs.fast.ai/](https://docs.fast.ai/)