Kaggle Competition

Cassava Leaf Disease Classification

경상대학교 수학과 오서영

1. Introduction

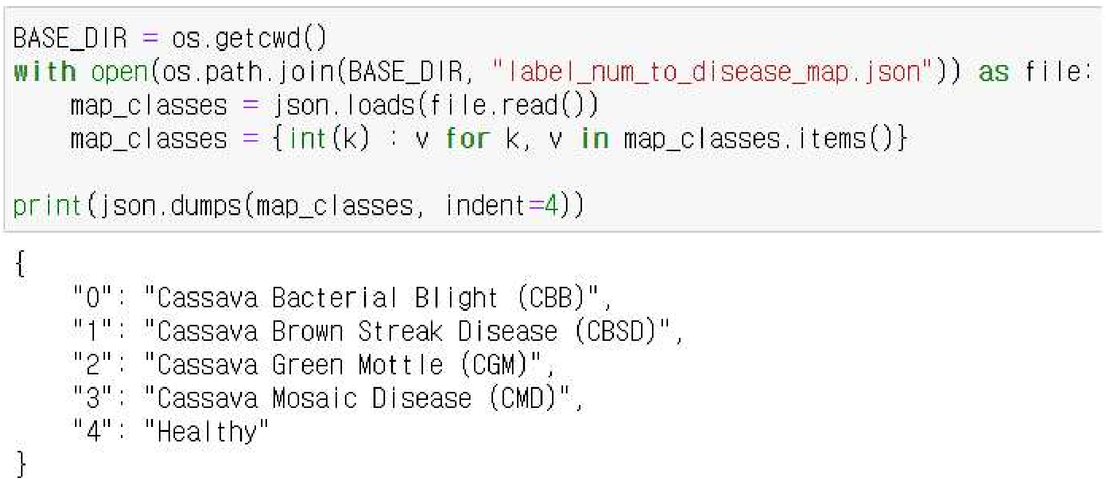
Cassava is a key food security crop in Africa because it can withstand harsh conditions.
But viral diseases are major sources of poor yields. Existing methods of disease detection require experts to visually inspect and diagnose the plants. This suffers from being labor-intensive, low-supply and costly.
So I want to solve this problem through data science. This competition aims to classify each cassava image into 5 category indicating four diseases and healthy leaf.

2. Background

In this competition, i use a dataset of 21,367 labeled images collected in Uganda. Most images were crowdsourced from farmers taking photos of their gardens, and annotated by experts at NaCRRI in collaboration with the AI lab at Makerere University.

3. Project

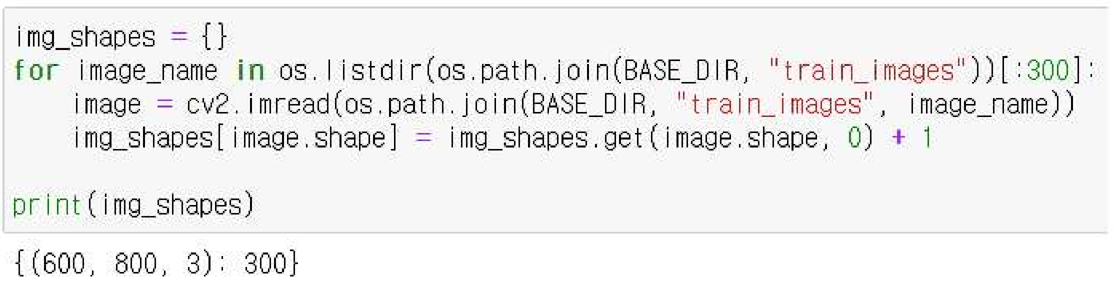
1) Data Exploration and Visualization



```
BASE_DIR = os.getcwd()
with open(os.path.join(BASE_DIR, "label_num_to_disease_map.json")) as file:
    map_classes = json.loads(file.read())
    map_classes = {int(k) : v for k, v in map_classes.items()}

print(json.dumps(map_classes, indent=4))
```

```
{
    "0": "Cassava Bacterial Blight (CBB)",
    "1": "Cassava Brown Streak Disease (CBSD)",
    "2": "Cassava Green Mottle (CGM)",
    "3": "Cassava Mosaic Disease (CMD)",
    "4": "Healthy"
}
```

There are four types of disease in dataset : Cassava Bacterial Blight (CBB), Cassava Brown Streak Disease (CBSD), Cassava Green Mottle (CGM) and Cassava Mosaic Disease (CMD).
The five labels, including healthy state, are expressed as integers between 0 and 4 in the model.



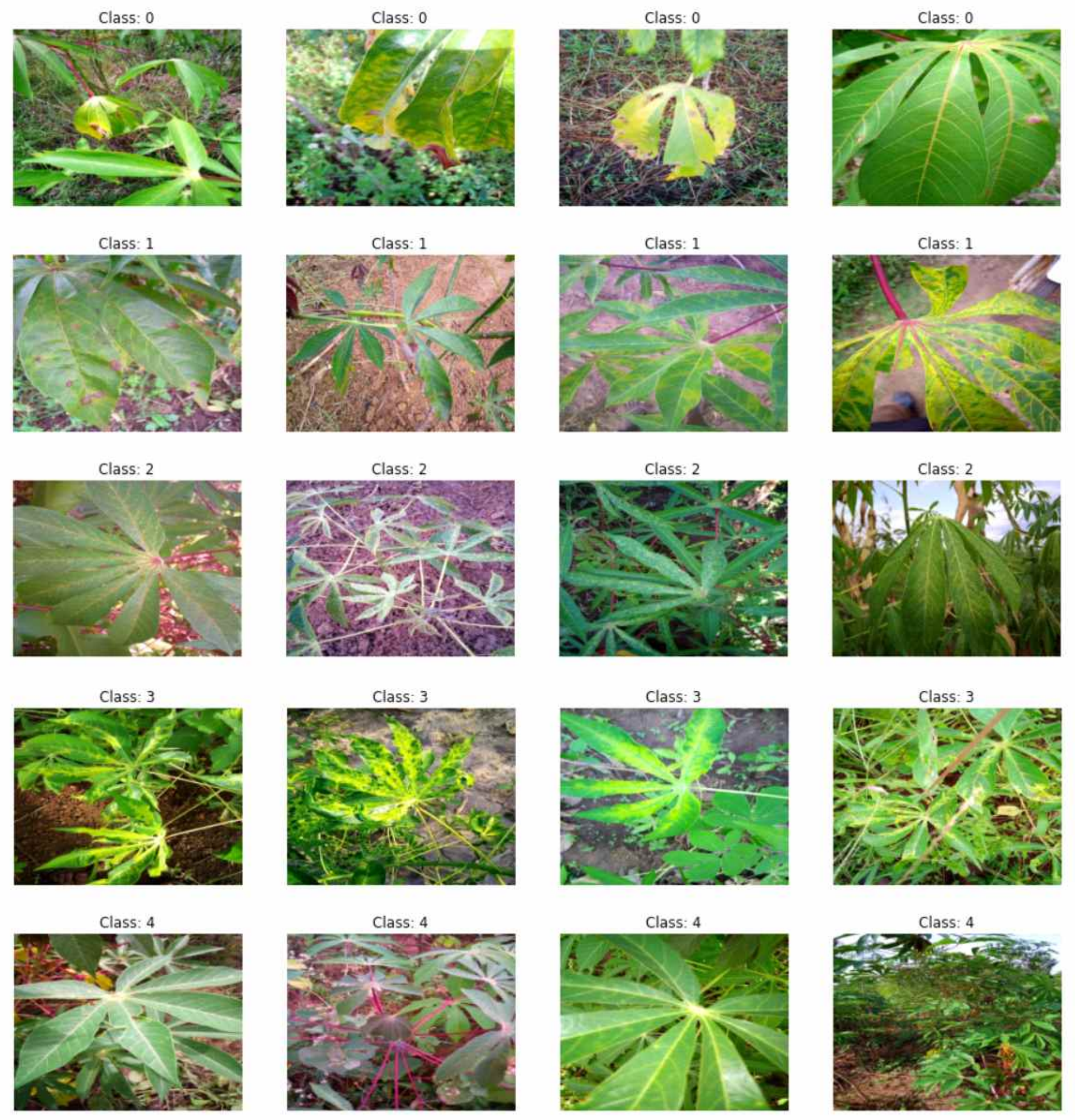
```
img_shapes = {}
for image_name in os.listdir(os.path.join(BASE_DIR, "train_images"))[:300]:
    image = cv2.imread(os.path.join(BASE_DIR, "train_images", image_name))
    img_shapes[image.shape] = img_shapes.get(image.shape, 0) + 1

print(img_shapes)
```

```
{(600, 800, 3): 300}
```

```
tmp_df0 = df_train[df_train["label"] == 0]
print(f"Total train images for class 0: {tmp_df0.shape[0]}")
tmp_df1 = df_train[df_train["label"] == 1]
print(f"Total train images for class 1: {tmp_df1.shape[0]}")
tmp_df2 = df_train[df_train["label"] == 2]
print(f"Total train images for class 2: {tmp_df2.shape[0]}")
tmp_df3 = df_train[df_train["label"] == 3]
print(f"Total train images for class 3: {tmp_df3.shape[0]}")
tmp_df4 = df_train[df_train["label"] == 4]
print(f"Total train images for class 4: {tmp_df4.shape[0]}")
```

```
Total train images for class 0: 1087
Total train images for class 1: 2189
Total train images for class 2: 2386
Total train images for class 3: 13158
Total train images for class 4: 2577
```

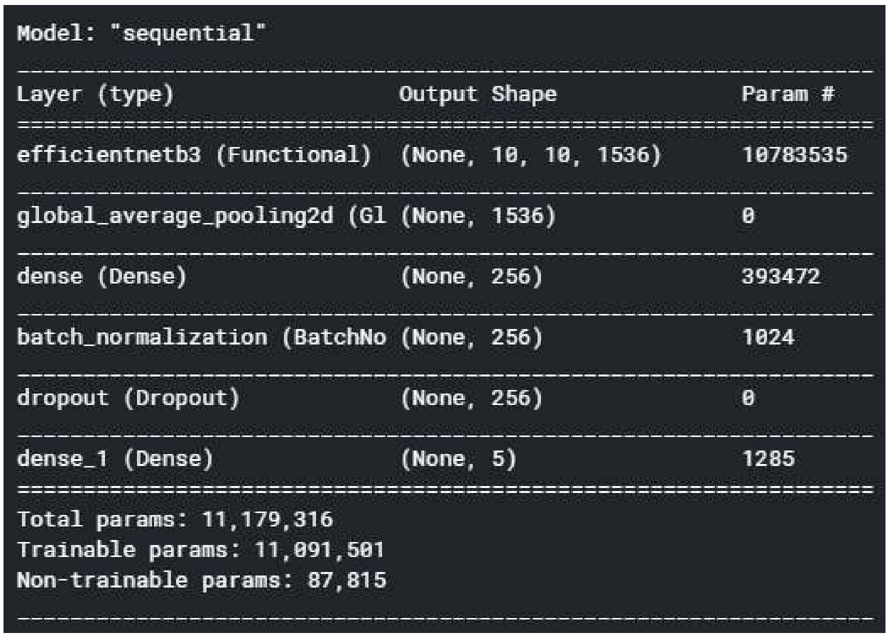
Also, all image is (600,800,3) shape. And class 3 has a lot more data than other classes.



2) Modeling

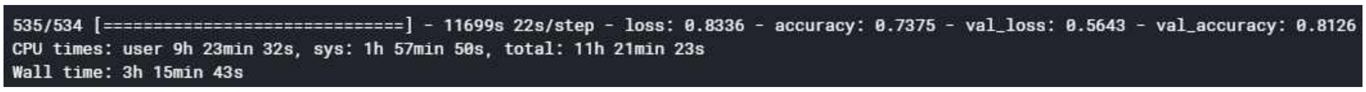
Since the purpose of this competition is to classify images, I use vanilla CNN first. Image data are resized to (64,64), and training are conducted with CNN using 64 filters of size (3,3) on epoch 10. However, The accuracy was less than 60%.

The traditional way to improve CNN performance is to increase the depth of model. However, I decide to use a model called EfficientNet that can improve accuracy and efficiency by considering not only depth but also width and resolution together. Images of size 128 x 128 have more information than images of size 64 x 64. But high-resolution input can result in a large calculation cost, and does not provide a significant benefit to model performance after certain values. The images we use have a large resolution, so it will be helpful to choose the optimal resolution through this model.



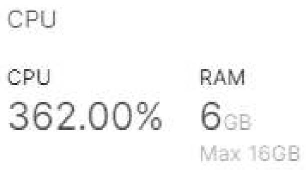
```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
efficientnetb3 (Functional)  (None, 10, 10, 1536)      10783535
_________________________________________________________________
global_average_pooling2d (Gl (None, 1536)              0
_________________________________________________________________
dense (Dense)                (None, 256)               393472
_________________________________________________________________
batch_normalization (BatchNo (None, 256)               1024
_________________________________________________________________
dropout (Dropout)            (None, 256)               0
_________________________________________________________________
dense_1 (Dense)              (None, 5)                 1285
=================================================================
Total params: 11,179,316
Trainable params: 11,091,501
Non-trainable params: 87,815
_________________________________________________________________
```

I don't use normalization. Since in EfficientNet, normalization is done within the model itself and the model expects input in the range of [0,255].



```
535/534 [==============================] - 11699s 22s/step - loss: 0.8336 - accuracy: 0.7375 - val_loss: 0.5643 - val_accuracy: 0.8126
CPU times: user 9h 23min 32s, sys: 1h 57min 50s, total: 11h 21min 23s
Wall time: 3h 15min 43s
```

The training accuracy in the training process is about 70%, but the test accuracy is 40%. And it takes 3 hours per epoch. This is too slow.



```
CPU

CPU        RAM
362.00%    6GB
           Max 16GB
```

References

[1] Cassava Leaf Disease - Exploratory Data Analysis,
https://www.kaggle.com/ihelon/cassava-leaf-disease-exploratory-data-analysis
[2] Cassava Leaf Disease: Best Keras CNN,
https://www.kaggle.com/maksymshkliarevskyi/cassava-leaf-disease-best-keras-cnn#Preparation-for-modeling