Student ID: 19127592

Name: Lê Minh Trí

Classification

1. Environment:

- Compiler: Visual Studio Code
- Language: Python
- Version: 3.10.0
- Environment: kernel
- Application: Jupiter Notebook

2. Plan:

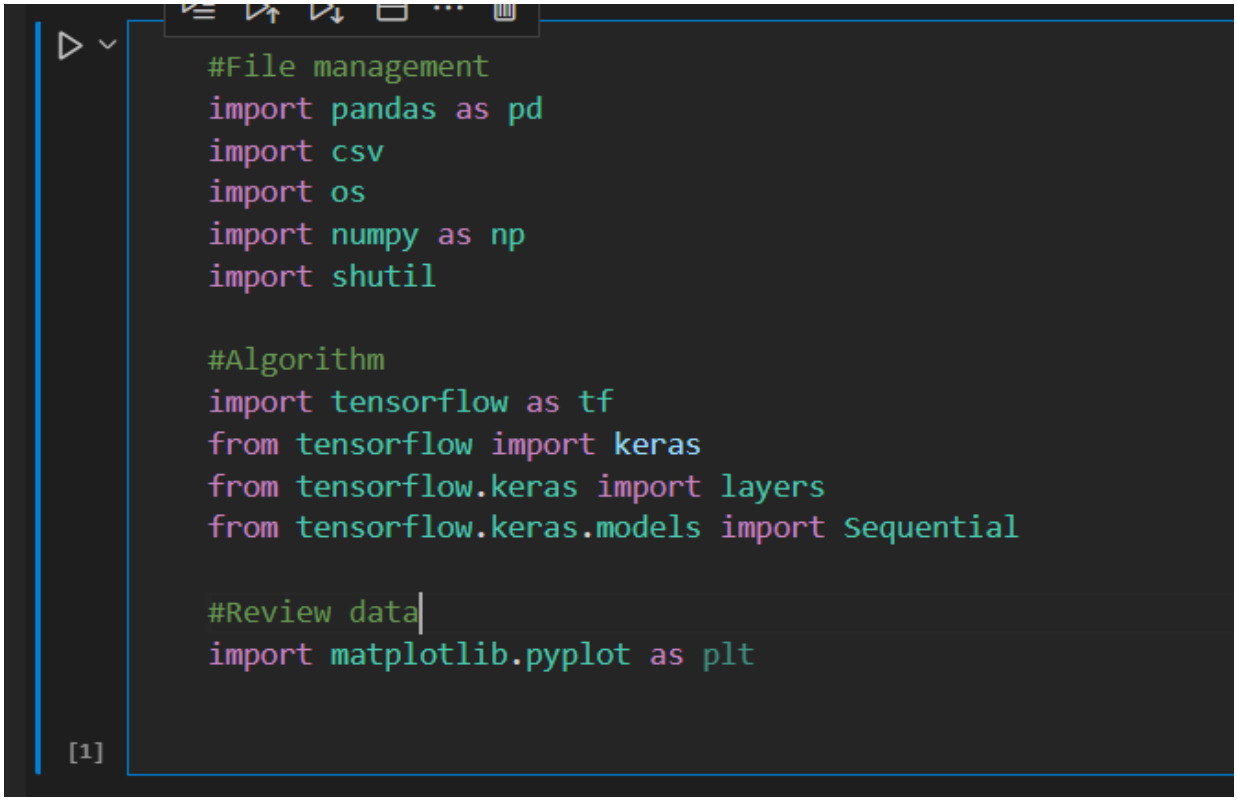
The idea is similar to detect faces from the last homework but this time, we detect leaves and classify their status.

In order to do that, we have to build a model from training the dataset many times. The training can be ideally made in aspect ratio of 8:2 or 7:3 and repeat the procedure at least 10 times to increase the accuracy.

I use Convolutional Neutral Network(CNN) with the help of TensorFlow library to do the job.

3. Implementation:

- Import libraries



```
#File management
import pandas as pd
import csv
import os
import numpy as np
import shutil

#Algorithm
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential

#Review data
import matplotlib.pyplot as plt
```

[1]

- Prepare the dataset by labeling and splitting images due to train.csv

```
train_path = 'train.csv'
parent_dir_cls = 'classification/'
df_train = pd.read_csv(train_path)

#Get class-names
header = list(df_train)
image_header = header[0]
header.pop(0)
label_list = []

#Label images from train.csv
for row in range(len(df_train)):
    for label in header:
        if(df_train.loc[row,label] == 1):
            image_id = df_train.loc[row,image_header]
            label_list.append((image_id+".jpg",label))

#Create class-names folder
for label in header:
    path = os.path.join(parent_dir_cls, label)
    if (os.path.exists(path)) == False:
        os.mkdir(path)

#Copy labeled images to suitable folders
for elem in label_list:
    #Original path
    original = "images/" + elem[0]
    #Target path
    target = parent_dir_cls + elem[1] + '/' + elem[0]
    #Copy the image
    shutil.copyfile(original, target)
```

- Create training and testing dataset with ratio 8:2

```
parent_dir_cls = 'classification/'
batch_size = 100
height = 400
width = 400

#Create training and testing dataset
train_img = tf.keras.utils.image_dataset_from_directory(
  parent_dir_cls,
  validation_split=0.2,
  subset="training",
  seed=123,
  image_size=(height, width),
  batch_size=batch_size)

val_ds = tf.keras.utils.image_dataset_from_directory(
  parent_dir_cls,
  validation_split=0.2,
  subset="validation",
  seed=123,
  image_size=(height, width),
  batch_size=batch_size)


class_names = train_img.class_names
print(class_names)
```

- Configure dataset

```
#Configure the cache for dataset
AUTOTUNE = tf.data.AUTOTUNE
train_img = train_img.cache().shuffle(1000).prefetch(buffer_size=AUTOTUNE)
val_ds = val_ds.cache().prefetch(buffer_size=AUTOTUNE)
```

- Prepare model

```
data_augmentation = keras.Sequential(
  [
    layers.RandomFlip("horizontal",input_shape=(height,width,3)),
    layers.RandomRotation(0.1),
    layers.RandomZoom(0.1),
  ]
)

model = Sequential([
  data_augmentation,
  layers.Rescaling(1./255),
  layers.Conv2D(16, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(32, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(64, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Dropout(0.2),
  layers.Flatten(),
  layers.Dense(128, activation='relu'),
  layers.Dense(5)
])

model.compile(optimizer='adam',loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),metrics=['accuracy'])
```

- Train model

```
calc_model = model.fit(
  train_img,
  validation_data=val_ds,
  epochs=15
)
```

- Save model

```
model.save('y_model.h5')
```

- Test model

```
image_dir = "images/"
new_model = tf.keras.models.load_model('y_model.h5')
height = 400
width = 400
with open('test.csv','r',newline='',encoding='utf-8') as f:
    csv_reader = csv.reader(f, delimiter=',')
    for row in csv_reader:
        fileName = row[0] + ".jpg"
        img_path = image_dir + fileName
        if os.path.exists(img_path):
            load_img = tf.keras.utils.load_img(img_path,target_size=(height,width))
            img_array = tf.keras.utils.img_to_array(load_img)
            img_array = tf.expand_dims(img_array, 0)
            predictions = new_model.predict(img_array)
            score = tf.nn.softmax(predictions[0])
            pred_class = class_names[np.argmax(score)]


            print(
                    fileName + " maybe " + pred_class
                )
```

- Result:

```
Test_0.jpg maybe rust
Test_1.jpg maybe rust
Test_2.jpg maybe scab
Test_3.jpg maybe healthy
Test_4.jpg maybe rust
Test_5.jpg maybe healthy
Test_6.jpg maybe healthy
Test_7.jpg maybe scab
Test_8.jpg maybe scab
Test_9.jpg maybe rust
Test_10.jpg maybe healthy
Test_11.jpg maybe healthy
Test_12.jpg maybe scab
Test_13.jpg maybe scab
Test_14.jpg maybe rust
Test_15.jpg maybe rust
Test_16.jpg maybe rust
Test_17.jpg maybe scab
Test_18.jpg maybe scab
```

4. Analysis

- **Training and Testing**

  Create both training and testing data with **tf.keras.utils.image_dataset_from_directory**

  Common used ratios are:
  + 70% train, 30% validation
  + 80% train, 20% validation

  I choose the second option due to some information, it is the best ratio for random and manipulating data.
  Source:https://stackoverflow.com/questions/13610074/is-there-a-rule-of-thumb-for-how-to-divide-a-dataset-into-training-and-validatio

  I also resize the image to 400x400 and define batch_size = 100 for less memory system and faster training. Batch_size is the number of samples the program takes for each training.

```
Found 1821 files belonging to 4 classes.
Using 1457 files for training.
Found 1821 files belonging to 4 classes.
Using 364 files for validation.
['healthy', 'multiple_diseases', 'rust', 'scab']
```

- **Cache**

  To optimize performance, we can use **Dataset.cache()** to save the images to memory which handle the flow of data during model training. If the dataset is too large for the memory system, use this method to create an on-disk cache.

  Remember to randomize data using shuffle() function.

  Dataset.prefetch() can be used to allow later sample to be prepared while the current sample is being processed. This improves latency and output but using more memory to store those samples.

- **Create model**
  I use Sequential model as it is suitable to image classifier in CNN.

  One problem is that overfitting given false prediction and the quality of model is not good in testing data. This happens when training data is too small compared to the complex model.

  One solution is increasing the diversity of your training set by applying several layers:
  + layers.RandomFlip
  + layers.RandomRotation
  + layers.RandomZoom

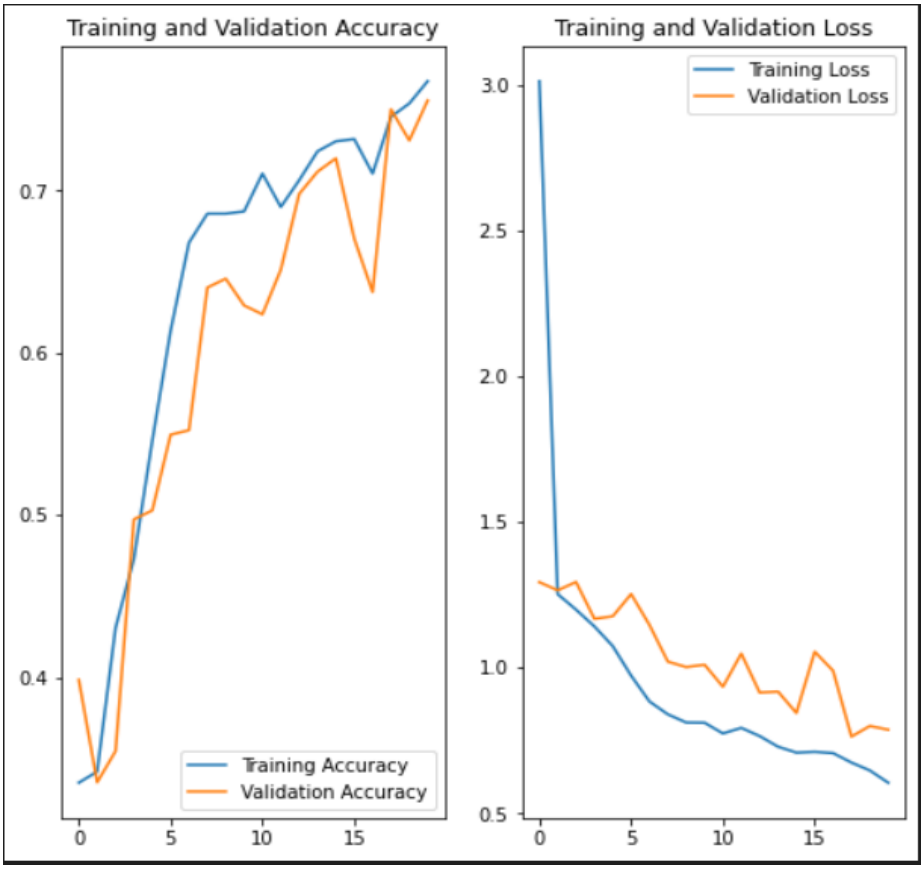
  Another solution is the dropout technique by adding dropout layer to the model. Dropout layer randomly drops out a number of outputs during the training process. Input value is fractional number meaning the percentage of random units generated by the layer.
  Source: https://developers.google.com/machine-learning/glossary#dropout_regularization

- **Compile the model**

  For optimizer, I choose Adam algorithm which is a stochastic gradient descent method that is based on adaptive estimation of first-order and second-order moments.
  Source: https://arxiv.org/abs/1412.6980

  As loss function, because there are four label classes, SparseCategoricalCrossentropy is recommended
  Source:https://www.tensorflow.org/api_docs/python/tf/keras/losses/SparseCategoricalCrossentropy
- **Train the model**
  Just use fit method with proper epochs. I choose 20 epochs to increase the accuracy as best as possible

  Graph illustrates training and validation accuracy after 20 times



As you can see from the graph, multiple training times are able to improve the accuracy and reduce loss.

- **Save and test model**

  Because my model takes 1h30p to complete, it is necessary to save the model with .h5 extension

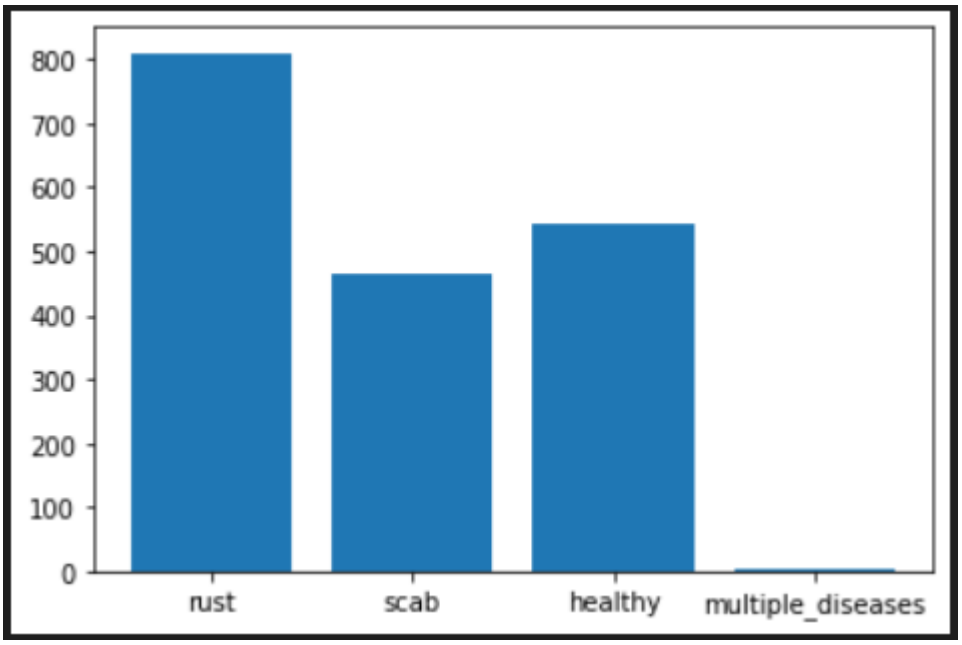
  Pip install: pip install pyyaml h5py

  Run predict function to every test images and test the model to predict the illness of the leaves.

```
Test_0.jpg maybe rust
Test_1.jpg maybe rust
Test_2.jpg maybe scab
Test_3.jpg maybe healthy
Test_4.jpg maybe rust
Test_5.jpg maybe healthy
Test_6.jpg maybe healthy
Test_7.jpg maybe scab
Test_8.jpg maybe scab
Test_9.jpg maybe rust
Test_10.jpg maybe healthy
Test_11.jpg maybe healthy
Test_12.jpg maybe scab
Test_13.jpg maybe scab
Test_14.jpg maybe rust
Test_15.jpg maybe rust
Test_16.jpg maybe rust
Test_17.jpg maybe scab
```

5. Final result:

Visualize the occurrences of four diseases of the given testing leaves by a bar diagram:



As I don’t have the exact output data, I cannot guarantee my output correct.

One thing needed to consider is that the number of leaves suffered from multiple_diseases is only a small amount compared to others. In my opinion, the label multiple_diseases is a bit more complex than others because it contains both scab and rust phenomenon so overfitting or some other problems may be negative to this situation.