PYTHON

Course: Introduction to Python ||

Title: Document for Final Project **[Trash Classification]**

Team: No.7

Student Name: Huiyi Liu, Yuming Chen

Student Number: 320180940030, 320180939611

Date Due: Wednesday 16th, October, 2019

1. Introduction

1.1 Overview

The package is designed for classifying trash. The main capabilities include collecting images of individual pieces of trash from cameras on the mechanisms, and dividing the trashinto two categories: recyclable and organic. From the technical perspective, the first capability mainly uses OpenCV to implement while the other depends on deep learning which is a subclass of machine learning to construct the binary classification model.

1.2 Background

We choose this idea for the reason that trash shall be classified for proper handling. However, nowadays, people rarely sort rubbish voluntarily and the classify result is far from accurate. Therefore, an automatic rubbish classifier is useful and interesting in some degree.

2. Requirement

2.1 Functional requirements

Rq1: A camera class shall be implemented whose instances are cameras of different trash processing device.

Rq1.1: The cameras shall be tested.

Rq1.2: For each camera, the frames in which trash appears shall be detected

Rq1.3: The detected images with timestamps shall be stored in different directories according to the camera taken by.

Rq2: A classifier classifying the images collected by the cameras shall be implemented.

Rq2.1: A training module classifying recyclable and non-recyclable trash shall be built using deep learning.

Rq2.2: The trained model generated by the training module shall be stored as a file.

Rq2.3 The images shall be classified using the model.

Rq2.4 A thread shall be set for each camera directory

Rq2.4.1 Each thread shall classify the pictures in the corresponding camera directory.

Rq2.5 The images shall be stored in different directories according to its classifying result.

Rq2.6 The result shall be sent to the mechanical device which sends the trash of different categories to different places (maybe different rubbish bin).

Rq2.7 The image in the camera directory shall be removed after the result is sent to the mechanism.

Rq2.8 The classifying record shall be added to the log file after classifying.

2.2 Non-functional requirements

Rq3: The doctests shall be provided.

Rq4. The accuracy of the machine learning model shall be evaluated.

Rq5: Test driven development shall be used

Rq5.1: Function refactoring shall be logged

Rq5.2: Basic functional checking shall be recorded in doctests

Rq5.3: Unit testing shall be provided

Rq5.4: A demonstrator use-case (application) using the package may be provided.

Rq6: The benchmark of technologies and solutions shall be reported.

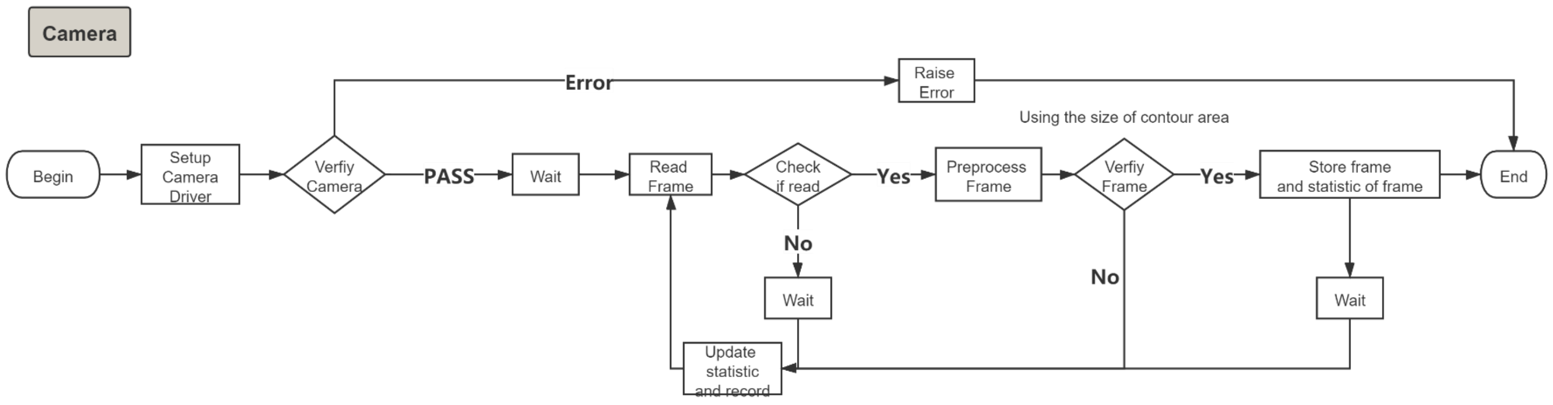
2.3 Technical requirements

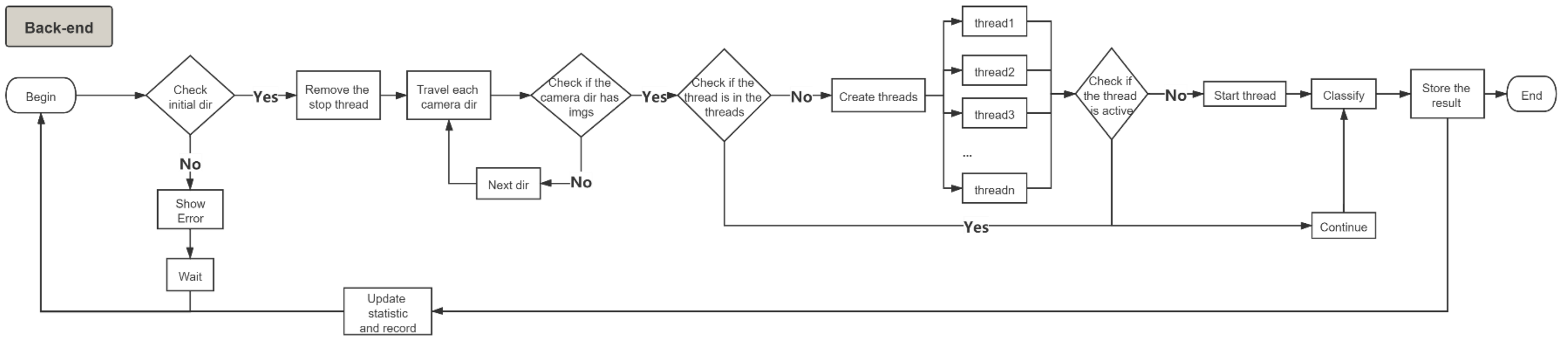
Rq6: Python3 and OpenCV2 shall be used.

Rq7: The system shall be available 99.99% of the time for any 24-hour period.

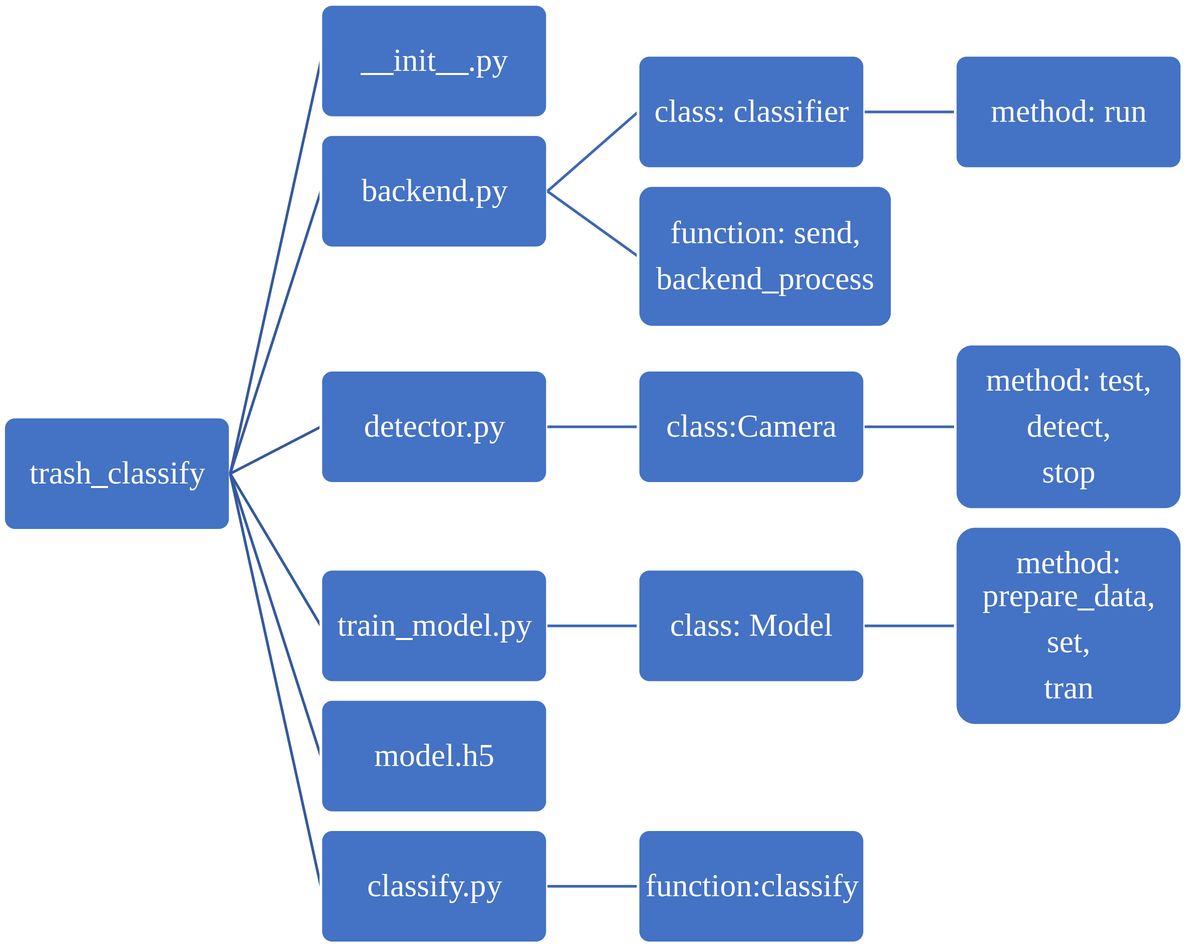
3. Design

3.1 Workflow





3.2 High-level structure



The overall structure is shown above. The classifier is implemented as a class because it uses concurrency and there is an instance of classifier for each camera. Classify is set as a function which simply uses the model to classify a single image, implementing a step of operation used in classifier class. Detection is implemented as a method of Camera class for it only applies to the instances of Camera class.

Note: "model.h5" is the binary classification model that has already been built using *train_model.py and the dataset from https://www.kaggle.com/techsash/waste-classification-data. The user can retrain the model with other datasets.*

4. Benchmark

4.1 Classifier Benchmark

When using the trained model to classify the images captured, it is necessary to read and preprocess images. There are two solutions to implement it.

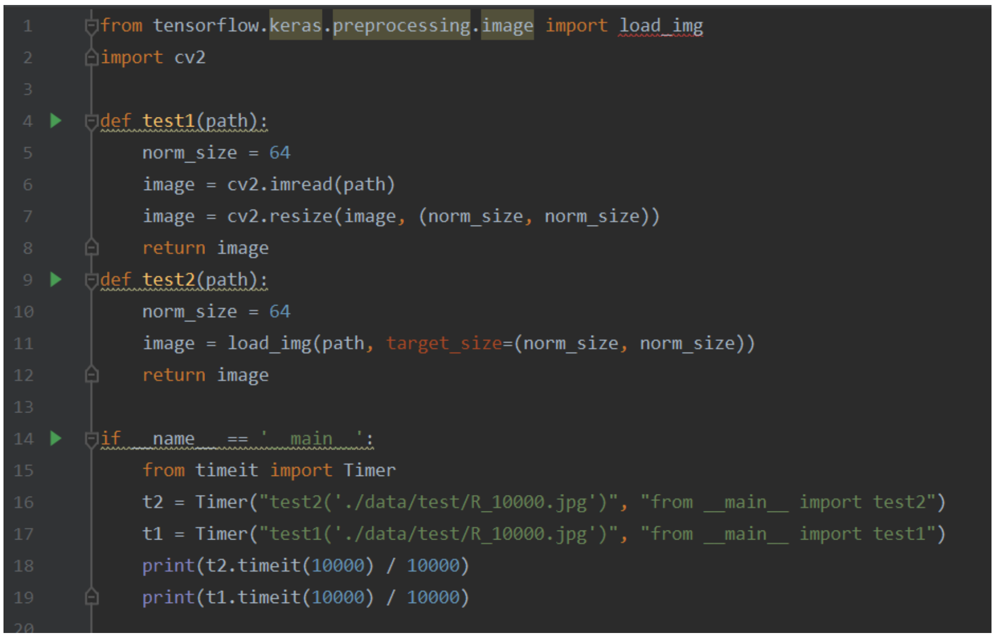
Solution1: Use '***cv2.imread(path)***' together with '***cv2.resize(image, (norm_size, norm_size))***'.

Solution2: Use '***tensorflow.keras.preprocessing.image.load_img(path, target_size=(64, 64))***'.

In order to compare the two solutions, we use two evaluation metrics to obtain a holistic understanding: speed and memory consumption. The environment of the benchmark is Python3.7 run on Intel® Core™ i7-10510U CPU.

4.1.1 Speed Benchmark

We use timeit.Timer as the method to get their execution time in order to measure the speed for the reason that another module line_profiler does not fit Python 3.7 well. The test1(path) function refers to *Solution1* while test2(path) refers to the second one. The testing code is as follows:



```python
from tensorflow.keras.preprocessing.image import load_img
import cv2

def test1(path):
    norm_size = 64
    image = cv2.imread(path)
    image = cv2.resize(image, (norm_size, norm_size))
    return image
def test2(path):
    norm_size = 64
    image = load_img(path, target_size=(norm_size, norm_size))
    return image

if __name__ == '__main__':
    from timeit import Timer
    t2 = Timer("test2('./data/test/R_10000.jpg')", "from __main__ import test2")
    t1 = Timer("test1('./data/test/R_10000.jpg')", "from __main__ import test1")
    print(t2.timeit(10000) / 10000)
    print(t1.timeit(10000) / 10000)
```

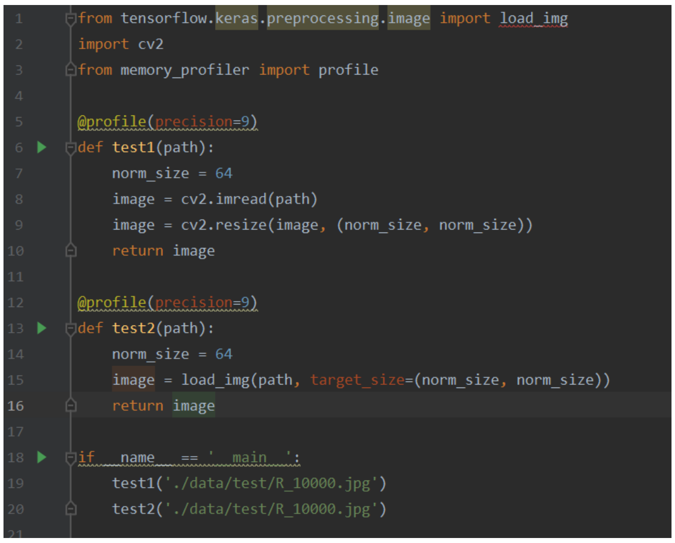
In order to get more accurate result, we run this testing file for ten times. For first five times, test2(path) runs before test1(path). For the others,test1(path) runs first. The results are as follows (keep 9 decimal places).

	Time of test1 function(s)	*Time of test2 function(s)*
1	0.001946587	0.001632469
2	0.001269862	0.001265879
3	0.001627106	0.001484421
4	0.001426762	0.001134087
5	0.001531878	0.001485469
6	0.001429907	0.001411047
7	0.001567396	0.001473186
8	0.001354674	0.001468779
9	0.001168707	0.001029274
10	0.001332155	0.001348859
Average	0.001465504	0.001373347

It can be concluded that in most cases, test2 is faster than test1. The average run time that test2 spent is 0.063 times shorter than test1. Therefore, solution2 performs better when considering run speed.

4.1.2 Memory Benchmark

We use memory_profiler.profile for line-by-line analysis of memory consumption for python programs. The memory increment of the relevant code of the solutions reflects the memory consumption. The test1(path) function refers to *Solution1* while test2(path) refers to the second one. The code is shown as follows.



```python
from tensorflow.keras.preprocessing.image import load_img
import cv2
from memory_profiler import profile

@profile(precision=9)
def test1(path):
    norm_size = 64
    image = cv2.imread(path)
    image = cv2.resize(image, (norm_size, norm_size))
    return image

@profile(precision=9)
def test2(path):
    norm_size = 64
    image = load_img(path, target_size=(norm_size, norm_size))
    return image

if __name__ == '__main__':
    test1('./data/test/R_10000.jpg')
    test2('./data/test/R_10000.jpg')
```

The process is the same as Speed Benchmark. We run this testing file for ten times to improve the reliability of the result. We compute the sum of the memory increment of the lines in a function as the increment of this function.

	Memory Increment of test1 function(MiB)	*Memory Increment of test2 function(MiB))*
1	0.593750000	0.683593750
2	0.750000000	0.742187500
3	0.726562500	0.542968750
4	0.5820312500	0.562500000
5	0.726562500	0.753906250
6	0.531250000	0.589843750
7	0.582000000	0.609400000
8	0.519531250	0.699218750
9	0.531250000	0.636718750
10	0.472656250	0.769531250
Average	0.601559375	0.658986875

The result indicates that in more than half cases, Solution 2 assumes more memory. The average increment of test2 function is about 0.0955 times larger than test1. From this perspective, Solution1 is a better choice.

4.1.3 Conclusion

According to the benchmark above, the differences the two functions are small. For the reason that in the whole package, each call to this function is to operate on a single image, there is no need to consider the influence brought by the large scale of data in one call. But the gap between the two solutions will increase when it is called for many times. For the reason that time is more essential than memory space, on condition that the memory is enough, *Solution1* would be a better choice. Therefore, we choose *Solution1* after the tradeoff.

5. Technology

5.1 Blur function

In the camera class, which blur function to c

hoose was questioned. There are four commonly used blur functions provided by OpenCV:

- **cv2.blur:** The function smoothes an image using the kernel:

$$K = \frac{1}{\texttt{ksize.width*ksize.height}} \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 & 1 \\ 1 & 1 & 1 & \cdots & 1 & 1 \\ & & \cdots\cdots\cdots & & & \\ 1 & 1 & 1 & \cdots & 1 & 1 \end{bmatrix}$$

- **cv2.GaussianBlur:** Blurs an image using a Gaussian filter. We should specify the width and height of the kernel which should be positive and odd. We also should specify the standard deviation in the X and Y directions, sigmaX and sigmaY respectively. If only sigmaX is specified, sigmaY is taken as the same as sigmaX. If both are given as zeros, they are calculated from the kernel size.
- **cv2.medianBlur:** The function smoothes an image using the median filter with the aperture. Each channel of a multi-channel image is processed independently.
- **cv2.bilateralFilter** Bilateral filtering is a simple, non-iterative scheme for edge-preserving smoothing.

BilateralFilter can reduce unwanted noise very well while keeping edges fairly sharp. However, it is very slow compared to most filters. MedianBlur does not fit images with many details for precise details takes a lot of time when using MedianBlur. For the reason that the brightness of different parts of the image varies, cv2.blur is absolute not a good choise. Gaussian blurring is highly effective in removing Gaussian noise from an image. And it has high accuracy in detector. Therefore we use GaussianBlur as the blur function.

5.2 Deep learning framework

When we build the model for the classifier using deep learning technology, we use tensorflow.keras to build and train the binary classification model. The official explanation is as follows:

tf.keras is TensorFlow's high-level API for building and training deep learning models. It's used for fast prototyping, state-of-the-art research, and production, with three key advantages: User-friendly Keras has a simple, consistent interface optimized for common use cases.

In other words, Keras is compact, easy to learn, high-level Python library run on top of TensorFlow framework. It is made with focus of understanding deep learning techniques, such as creating layers for neural networks maintaining the concepts of shapes and mathematical details. The creation of framework can be of the following two types −Sequential API and Functional API.

- **The sequential API** allows us to create models layer-by-layer for most problems. It is limited in that it does not allow to create models that share layers or have multiple inputs or outputs.
- **The functional API** allows to create models that have a lot more flexibility as we can easily define models where layers connect to more than just the previous and next layers. In fact, we can connect layers to (literally) any other layer. As a result, creating complex networks such as siamese networks and residual networks become possible.

In conclusion, for the reason that the model to be built is a binary classification model with only one input and one output, the sequential API is enough and it is simpler to implement. Therefore, we use the sequential API to create the framework.

Reference

[1]OpenCV: Image Filtering,

https://docs.opencv.org/2.4/modules/imgproc/doc/filtering.html

[2]TensorFlow-Keras: Tutorialspoint

https://www.tutorialspoint.com/tensorflow/tensorflow_keras.htm

[3] Keras Documentation

https://keras.io/