CS 2302 - Data Structures
Fall 2018
Project 1 - Option A (Animal Shelter)

Overview

You decide to volunteer at a non-profit organization in your spare time. The organization runs an animal shelter where abandoned dogs and cats are rehabilitated. They decide to build a website where people can see the dogs and cats they have for adoption. Some volunteers decide to use their phones to take pictures of the dogs and cats that are ready to be adopted, so they can put them on the website. Here are some of those pictures:


The volunteers are asked to upload the pictures they took to a shared Google Drive folder, so the people that are building the website can access them. You and some other programmers volunteer to build the website; however, when you take a look at the shared Google Drive folder, you notice the following:

- There are multiple folders inside the shared Google Drive folder, and inside those folders, there are more folders! Here is a subset of the things you notice about the whole directory structure:
    - Juan uploaded some of the pictures he took to <Shared Drive Directory>/Pictures/DCIM/100ANDRO
    - Maria uploaded the pictures she took to <Shared Drive Directory>/Maria/Pics/CatsAndDogs
    - John copied and pasted the pictures he took in the root folder

- Dog and cat pictures are all mixed and you cannot use the filenames (or any other piece of metadata) to tell them apart

You and the other programmers decide to write a program to traverse the messy directory tree and determine which pictures contain dogs and which contain cats. To tackle this problem, one of your team members decides to build a deep learning model to classify images. She

implements a method called *classify_image* that receives the path to an image and returns a number between 0 and 1. The closer the number is to 1, the more likely it is to be a dog picture; the closer the number is to 0, the more likely it is to be a cat picture.

She uploads the code and folders to GitHub, and tells you that she did most of the work! You only need to implement the *process_dir* method. This method is supposed to traverse the directory tree,and return two lists: dog_list and cat_list. These two lists must contain the full paths to all dog and cat pictures (respectively). You think this is a perfect opportunity to show off your **recursion** skills!

Requirements

You need to install the following on your machine (if you haven't already):

- Anaconda (Python 3.6)
- Tensorflow (Optional. Install only if you want to actually classify the pictures)

What you need to do

Part 1 - Due Thursday, September 13, 2018

Download the starter code, implement *process_dir* (use recursion!), and implement multiple tests for your method (in your comments and report, you must justify your selection of tests). When you are done, upload the code to GitHub and send the link to the TA and instructor.

Part 2 - Due Tuesday, September 18, 2018

Add your team members as collaborators to your GitHub repo. They will add you to their projects as a collaborator as well. Read their code and give them feedback. Use *pull requests* and/or the *Issues* section to do so .

Extra Credit

The animal shelter operates in a "first in, first out" basis. People willing to adopt an animal have two options:

- Adopt the animal (of any type) that has been waiting the most time to be adopted
- Ask for a dog or a cat specifically, in which case the person will receive the animal of that type that has been waiting the longest time to be adopted

Implement the following methods:

- get_any_animal_type

    - Retrieves (and removes) the animal that has been waiting the longest to be adopted
- get_cat
    - Retrieves (and removes) the cat that has been waiting the longest to be adopted
- get_dog
    - Retrieves (and removes) the dog that has been waiting the longest to be adopted

Feel free to use any data structure(s) to solve this problem. Each picture contains a date in its file name. This is the date when the animal was admitted to the shelter.

Rubric

Criteria	**Proficient**	**Neutral**	**Unsatisfactory**
Correctness	The code compiles, runs, and solves the problem.	The code compiles, runs, but does not solve the problem (partial implementation).	The code does not compile/run, or little progress was made.
Space and Time complexity	Appropriate for the problem.	Can be greatly improved.	Space and time complexity not analyzed
Problem Decomposition	Operations are broken down into loosely coupled, highly cohesive methods	Operations are broken down into methods, but they are not loosely coupled/highly cohesive	Most of the logic is inside a couple of big methods
Style	Variables and methods have meaningful/appropriat e names	Only a subset of the variables and methods have meaningful/appropriat e names	Few or none of the variables and methods have meaningful/appropriat e names
Robustness	Program handles erroneous or unexpected input gracefully	Program handles some erroneous or unexpected input gracefully	Program does not handle erroneous or unexpected input gracefully
Documentation	Non-obvious code segments are well documented	Some non-obvious code segments are documented	Few or none non-obvious segments are

			documented
Code Review	Useful feedback was provided to team members. Feedback received from team members was used to improve the code.	Feedback was provided to team members, but it was not very useful. Feedback received from team mates was partially used to improve the code	Little to no feedback was provided to team mates. Received feedback was not used to improve the code.
Report	Covers all required material in a concise and clear way with proper grammar and spelling.	Covers a subset of the required material in a concise and clear way with proper grammar and spelling.	Does not cover enough material and/or the material is not presented in a concise and clear way with proper grammar and spelling.