Image Classification for a City Dog Show

Description Problem statement:

The city is hosting a citywide dog show and you have volunteered to help the organizing committee with contestant registration. Every participant that registers must submit an image of their dog along with biographical information about their dog. The registration system tags the images based upon the biographical information.

Some people are planning on registering pets that aren't actual dogs.
You need to use an already developed Python classifier to make sure the participants are dogs.

Principal Objectives

1. Correctly identify which pet images are of dogs (even if the breed is misclassified) and which pet images aren't of dogs.
2. Correctly classify the breed of dog, for the images that are of dogs.
3. Determine which CNN model architecture (ResNet, AlexNet, or VGG), "best" achieve objectives 1 and 2.
4. Consider the time resources required to best achieve objectives 1 and 2, and determine if an alternative solution would have given a "good enough" result, given the amount of time each of the algorithms takes to run.

Tasks:

Using your Python, I will determine which image classification algorithm works the "best" on classifying images as "dogs" or "not dogs".
I determine how well the "best" classification algorithm works on correctly identifying a dog's breed. Time how long each algorithm takes to solve the classification problem. With computational tasks, minding a trade-off between accuracy and runtime.

Approach in solving the Problem:

Introduction

In this project, I will use an image classifier to identify dog breeds. Using Python and not on the actual classifier. The focus is on using my Python skills to complete these tasks using the classifier function.

Program Outline

1. Time the program
   In the ***check_images.py*** we start by timing the program using Time Module to compute program runtime.This code will calculate the runtime of the program. Specifically, measure how long each of the algorithms will take to classify all the images in the pet_images folder.
2. Get program Inputs from the user
   Use command line arguments to get user inputs. I code within ***get_input_args.py***. The function, argparse retrieves three command line arguments from the user. It includes an argument parser object using **argparse.ArgumentParser** and then we use the **add_argument method** to allow the users to enter in these three external inputs from above.When completed, this code will input the three command line arguments from the user. Our program will be more flexible in allowing external inputs from the user.

3. Create Pet Images Labels
   Use the pet images filenames to create labels. We code the function **get_pet_labels** within ***get_pet_labels.py***.The function will need to read the filenames from a folder. To achieve this task I will import the **listdir method** from the **os python module**. The **listdir** method retrieves all filenames from the files within a folder. These filenames are returned from **listdir** as a list. We create an empty dictionary. Iterate through the file with the image directory. We start cleaning up, any file starting with (.) and all hidden files need to be out. All files need to be of we need to lower() alll our file names then split the names them then extract files that may start isAlpha values get knocked out , example (7588_dog.jpg output name dog not 7588 dog). Next section is adding the results in the data structure (dictionary) where we strip(). We return the result dictionary in our function.
4. Create Classifier Labels and Compare Labels
   We Code the function **classify_images** within ***classify_images.py***. With this function, we create the labels for images using the classifier function. Additionally, we compare these classifier's labels to the pet image labels. Finally, we store the classifier generated labels, and the comparison of labels in the results dictionary that was returned by **get_pet_labels** function. Through, the classifier function in *classifier.py* which determines matches between the pet image labels and the labels the classifier function returns are tested through test_classifier.py and imported to use in classify_images.py.
   Compare Classifier Labels to Pet Image Labels.
   In our function we have (imagedir, resultdic and model.)

For this function you will be inputing the *results_dic* dictionary that contains:

- The *filenames* as *keys*

- A *list* whose only item is the *pet image label*.

We:

- Iterate through this dictionary (*results_dic*) processing each pet image (filename) with the **classifier function** to get the classifier label.
- Compare the pet image and classifier labels to determine if they match.
- Add the results to the results dictionary (*results_dic*).

(Results dictionary is a mutable data type that doesn't need to return it from the **classify_images** function).

To prevent issues with **classify_images**:

- With the **classifier function**, we concatenate the *images_dir* with the *filename* to represent the *full* path to each pet image file.
- Put classifier labels in *all* lower case letters, stripping leading and trailing whitespace from the label.
- ***results_dic*** will have the following format:
    - *key* = pet image filename (ex: Beagle_01141.jpg)
    - *value* = List with:
        - index 0 = Pet Image Label (ex: beagle)
        - index 1 = Classifier Label (ex: english foxhound)
        - index 2 = 0/1 where 1 = labels match , 0 = labels don't match (ex: 0)
    - example_dictionary = {'Beagle_01141.jpg': ['beagle', 'english foxhound', 0]}
- We *initialize* a key in *results_dic*, we use *assignment* operator (=) to assign the value of a list.
- We add a single item to the list of an **existing** *key* in *results_dic*, append to the list using either += operator or the *append* function.
- We add multiple items simultaneously to the list of an **existing** *key* in *results_dic*, use the **extend** list function.

5. Classifying Labels as "Dogs" or "Not Dogs" (Abit technical but understandable)
   code the *undefined* function **adjust_results4_isadog** within ***adjust_results4_isadog.py***. We create a dictionary . open file dogfilename and close automatically process the file till end of new line
   Classify all Labels as "Dogs" or "Not Dogs" using dognames.txt file
   Store new classifications in the complex data structure (e.g. dictionary of lists). # Classifier Label IS image of Dog (e.g. found in dognames_dict then appends (1, 1) because both labels are dogs. Classifier Label IS NOT an image of a dog (e.g. NOT in dognames_dic) appends (1, 0) because the only pet label is a dog. Pet Image Label IS NOT a Dog image (e.g. NOT found in dognames_dic). Classifier Label IS image of Dog (e.g. found in dognames_dic) appends (0, 1) because only Classifier label is a dog. Classifier Label IS NOT an image of a dog (e.g. NOT in dognames_dic) appends (0, 0) because both labels aren't dogs

6. Calculate the Results
   We code the *undefined* function **calculates_results_stats** within ***calculates_results_stats.py***. With this function, we will be inputting the results dictionary to create a dictionary of results statistics. This results statistics dictionary will contain the *statistic's name* as the *key* and the *value* will simply be the *statistic's numeric* value. We will be storing these calculations (counts & percentages) in the results statistics dictionary. We then have Counts Computed from the *Results* dictionary for input into the *Results Statistics* dictionary and Compute a Summary of the Percentages from the *Results Statistics* dictionary counts.
   Note:

- We initialize all the counts to a value of zero before iterating through the results dictionary incrementing these counters by 1.
- The percentages (and the total number of images) will be generated from the counts (see percentage & count calculations above); therefore, these values should be calculated *after* counts have been calculated by iterating through the *results* dictionary.
- When calculating the percentage of correctly classified Non-Dog Images, we use a conditional statement to check that **D**, the number of "not-a-dog" images, is greater than zero. To avoid division by zero error, only if **D** is greater than zero should **C**/**D** be computed; otherwise, this should be set to 0.
- Because the *Results Statistics* dictionary is created inside of the function and is a mutable object, we will need to *return* its value at the end of the function (see section *Mutable Data Types and Functions*).

7. Print the Results
   We code the *undefined* function **print_results** within ***print_results.py***. The function will be inputting the *results* dictionary and the *results statistics* dictionary to print a summary of the results. Because this function allows one to print a list of incorrectly classified dogs and incorrectly classified breeds of dogs, one needs to include the *results* dictionary.
   The structure is to print model results used and result statistics of how correctly the batch made everything and then we run the program
   Lastly for Batch Processing we have **run_models_batch.sh .** For printing all model results.

Images source

There is a pet_images file where we have 40 images of different animals categorized.
To **test *test set*** the classifier there is an uploaded file with different sample images to test the classifier

Be mindful of the fact that image classifiers do not always categorize the images correctly.

Test set results

We had an inclusive dataset of a rotated image of a dog image and the rest were different animals and an image that was not of an animal Below we have a summary for the uploaded CNN Model Architecture.

Number of Images	4
Number of Dog Images	3
Number of Not-a-Dog Images	1

CNN Model Architecture	pct_match	pct_correct_dogs	pct_correct_breed	pct_correct_notdogs
VGG	50.0%	0.0%	100.0%	66.7.0%
AlexNet	50.0%	0.0%	100.0%	66.7.0%
RestNet	50.0%	0.0%	100.0%	66.7.0%

Model Deep Learning (CNN)

We use a deep learning model called a convolutional neural network (CNN). Why? CNNs work particularly well for detecting features in images like colors, textures, and edges; then using these features to identify objects in the images. We use a CNN that has already learned the features from a giant dataset of 1.2 million images called ImageNet. Types of CNNs architectures we use with this project are (AlexNet, VGG, and ResNet) and we determine which is best for project application.
There are .txt files for these models for results of a given model. Run by *run_sh_model.sh* file

We have a classifier function in **classifier.py** that will allow use of these CNNs to classify the images. The **test_classifier.py** file contains an example program that demonstrates how to use the classifier function.

Project Analysis

Results

In this section, we:

- Provide the results from running the **check_images.py** for all *three* CNN model architectures
- Compare these results to the ones your program produced when you ran *run_models_batch.sh* (or *run_models_batch_hints.sh*) in the **Printing Results** section
- Discuss how the **check_images.py** addressed the four primary objectives of this Project

In this project we had 2 main objectives:

- Identifying which pet images are of dogs and which pet images aren't of dogs
- Classifying the breeds of dogs, for the images that are of dogs

The program provided us with objectives *1* and *2* when it was run the program provided us with results as provided below.

- For objective 1, notice that both *VGG* and *AlexNet* correctly identify images of "dogs" and "not-a-dog" 100% of the time.
- For objective 2, *VGG* provides the best solution because it classifies the correct breed of dog over 90% of the time.

Results Table

# Total Images	40
# Dog Images	30
# Not-a-Dog Images	10

CNN Model Architecture:	% Not-a-Dog Correct	% Dogs Correct	% Breeds Correct	% Match Labels
ResNet	90.0%	***100.0%***	90.0%	82.5%
AlexNet	***100.0%***	***100.0%***	80.0%	75.0%
VGG	***100.0%***	***100.0%***	***93.3%***	***87.5%***

Given our results, the "**best**" model architecture is ***VGG***. It outperformed both of the other architectures when considering both objectives *1* and *2*. You will notice that *ResNet* did classify dog breeds better than *AlexNet*, but only *VGG* and *AlexNet* were able to classify "dogs" and "not-a-dog" at 100% accuracy. The model *VGG* was the one that was able to classify "dogs" and "not-a-dog" with 100% accuracy and had the best performance regarding breed classification with 93.3% accuracy.

Conclusion

The results indicate that more images for better classification of images is warranted, to better the classifier.
Fine tune the models is also an option we might consider after the addition of images.

References

Udacity Nanodegree Program Dog image classification Project.