# #TODO: 6 Printing The Results

**Implement the print\_results() function to print a summary of the results (as well as incorrect classifications of dogs and breeds if requested).**

## Coding within the check\_images.py and print\_results.py

### Code to Edit

This section will help you code the undefined function **print\_results** within print\_results.py. With this function you 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 dog, one needs to include the results dictionary.

* Code for the function definition def print\_results(): indicated by #TODO: 6 within print\_results.py
  + Using the comments and the docstring within print\_results.py to define **print\_results**
* Code within the main() function within check\_images.py indicated by #TODO: 6
  + Replace the None within the function call to **print\_results** with in\_arg.arch

### Expected Outcome

When completed, this code will print the summary of the results that will be used to answer objectives 1 and 2 of this project.

### Checking Your Code

For this you will just be running the completed program to visually check the following:

* Running the program results in the statistics and counts being properly printed and formatted. Results from the code check for Calculating Results should match the values printed for those 6 statistics.
* Leaving off the two default arguments in the function call to **print\_results**, results in no misclassifications being printed (That's an expected default behavior).
* Adding the values of True for the two default arguments in the function call to **print\_results**, results in the misclassifications being printed (That's also an expected default behavior).

### Final Program Run

Once you are satisfied the program is running properly use batch processing (see section on Batch Processing below) to run the program for all **three** CNN model architectures. You will use these results to compare them with our results in the section **Final Results**.

#### Project Workspace - Printing Results

* The next concept will have your workspace to work on #TODO: 6
* Editing of check\_image.py and print\_results.py can be done within the **Project Workspace - Printing Results**.

## For additional information and help on #TODO: 6, please look at the information below:

## Printing Results

The first thing to be printed is a general statement that indicates which of the three CNN model architectures you are using. You can pass the information in the model as an input parameter to be able to print it.

Next you will be printing the overall count which will be the same for all three CNN model architectures. This can be done by calling for those counts using the appropriate key within a print statement.

* Number of Images
* Number of Dog Images
* Number of "Not-a" Dog Images

Finally, you will be iterating through the results\_stats dictionary printing out the statistic's name and value for all of the percentages (e.g. key that starts with the letter "p"). Recall that we had recommended that you give the same prefix (e.g. pct\_) to all of the percentage statistics, so that they could all be printed out as a group.

Percentage Calculations:

* % Correct Dogs
* % Correct Breed
* % Correct "Not-a" Dog
* % Match (optional - this includes both dogs and not-a dog)

## Printing Misclassifications

This function allows one to optionally print cases of dog and breed misclassifications.

This optional feature is provided to allow improved debugging of the code. Additionally, this feature provides the ability to determine if there are certain breeds of dogs that the algorithms have difficulty identifying.

### Default Arguments for Misclassification

The function **print\_results** contains two default arguments for printing misclassified dogs and breeds. (In the lesson **Functions** you first learned about default arguments).

**Default Arguments**:

* print\_incorrect\_dogs - defaults to False
* print\_incorrect\_breed - defaults to False

#### Purpose

The purpose of default arguments can be the following:

* To provide a wider range of behaviors for a function, without having to code multiple (similar) functions.
* To guarantee that certain arguments are always assigned a value within a function.
* To provide a default behavior for a function.

### Misclassified Dogs

Labels are misclassified as dogs when both labels aren't in agreement regarding whether or not an image is of a dog.

Prior to iterating through the results dictionary to find dog misclassifications, you must first check that the user wants to print misclassified dogs and that dog misclassifications occurred with a conditional statement.

This check is done when:

* User wants to print misclassifications:
  + print\_incorrect\_dogs == True
* Some dogs were misclassified:
  + n\_correct\_dogs + n\_correct\_notdogs != n\_images

If the check is True, then print the pet image and classifier labels for misclassified dogs when:

* The labels disagree on whether or not an image is of a "dog"
  + sum(results\_dic[key][3:]) == 1

### Misclassified Breed's of Dog

Labels have a misclassification of breeds of dog when both labels indicate that the image is a dog; but, labels aren't in agreement regarding the dog's breed.

Prior to iterating through the results dictionary to find breed misclassifications, you must first check that the user wants to print misclassified breeds and that breed misclassification occurred with a conditional statement.

This check is done when:

* User wants to print misclassifications:
  + print\_incorrect\_breed == True
* Some breeds were misclassified:
  + n\_correct\_dogs != n\_correct\_breed

If the check is True, then print the pet image and classifier labels for misclassified breeds when:

* When the labels agree that image is of a dog, but disagree on the breed of dog
  + sum(results\_dic[key][3:]) == 2 and results\_dic[key][2] == 0

## Batch Processing

Now that you have completed coding check\_images.py, you are ready to run it on all 3 models. One way to do this is to call the program from the terminal window for one of the models, wait until it completes running, record it's results, and then repeat for the other two models.

An easier way to handle this task is with batch processing using a shell script. For this exercise, you will find the bash program **run\_models\_batch.sh** in the workspace. Open that file and you will notice comments use # just like python and the rest look the same as the commands you type into the terminal window to run your program (see code below).

*# Code from run\_models\_batch.sh*

python check\_images.py --dir pet\_images/ --arch resnet --dogfile dognames.txt

> resnet\_pet-images.txt

python check\_images.py --dir pet\_images/ --arch alexnet --dogfile dognames.txt

> alexnet\_pet-images.txt

python check\_images.py --dir pet\_images/ --arch vgg --dogfile dognames.txt

> vgg\_pet-images.txt

You will also notice that each file ends with > filename.txt. The > is a pipe and it pipes the output from the console into a file. The file contains the filename of the model being used. This way after each run, the results are automatically stored in your workspace.

To run file **run\_models\_batch.sh** in the workspace, open a terminal window (in Unix/Linux/OSX/Lab Workspace) and type the following:

sh run\_models\_batch.sh

If you want to batch process the program on a Windows computer you will need to follow the instructions found [**here**](https://github.com/udacity/AIPND/blob/master/notes/lab_intro-to-python-lab.md#running-batch-files-on-windows-os-locally).

Once you have ran all three models using run\_models\_batch.sh (run\_models\_batch.bat on Windows) compare your results with those you will find in the section **Final Results**.

# Classify Uploaded Images

## Classifying Uploaded Images

This section will help you test your program by classifying your own images.

Your check\_images.py program should now successfully classify the 40 images from the pet\_imagesfolder. In this section you will upload 4 images to the uploaded\_images folder, then run the file run\_models\_batch\_uploaded.sh to classify those 4 images.

### Directions for Finding Images and Uploading Images

Below are directions for finding images and processing them so they can be classified by the check\_images.py program.

* Process the images so that:
  + Images are in jpeg format with extension jpg
  + Images are approximately square in shape (their height and width are approximately the same number of pixels).
* Find the following **3** images (or take the following 3 pictures):
  + Dog**Image** - named Dog\_01.jpg. Make sure you know the breed of dog that the image is of.
  + Pet**or**Animal**Image** that's not a dog - named Animal\_Name\_01.jpg , where Animal\_Name is the name of the animal in the picture. This name is formatted such that if more than one word makes up the animal name those words are separated by an underbar ( \_ ).
    - For example:
      * Image of a Black Bear is named Black\_bear\_01.jpg
      * Image of a Frog is named Frog\_01.jpg
  + An **image of something** that's not an animal - named Object\_Name\_01.jpg, where Object\_Name is the name of the object in the picture. This name is formatted such that if more than one word makes up the object name those words are separated by an underbar ( \_ ).
    - For example:
      * Image of a Coffee Mug is named Coffee\_mug\_01.jpg
      * Image of a Bucket is named Bucket\_01.jpg
* Create a fourth Image of a Dog using Dog\_01.jpg
  + Using Dog\_01.jpg image horizontally flip the image and name it Dog\_02.jpg. This will mean that Dog\_02.jpg is a mirror image of Dog\_01.jpg. If you are having difficulty with the horizontalflip alteration of Dog\_01.jpg, just rotate Dog\_01.jpg image by 180 degrees so that Dog\_02.jpg is an upside down version of Dog\_01.jpg.

##### Upload all four images to the uploaded\_images folder within the **Project Workspace - Uploaded**

* Double click on the uploaded\_images folder within the **Project Workspace - Uploaded**.
* Next, click on the white **+** symbol above />home>workspace>uploaded\_images text
* Next, select **Upload File** from the dropdown menu
* Next, select one of the four files to upload to the uploaded\_images folder and click on the Openbutton
* Repeat the same process to upload the rest of the four files to the uploaded\_images folder
* To return to the workspace folder:
  + Click on the white **<** symbol above />home>workspace>uploaded\_images text

### Directions for Running check\_images.py using all 3 Model Architectures

Now that you have completed coding check\_images.py, you are ready to run it on all 3 models to classify the four images in the uploaded\_images folder. To do this you will be calling the following shell script that will output the following results files:

* resnet\_uploaded-images.txt - that contains the results using CNN model architecture ResNet
* alexnet\_uploaded-images.txt - that contains the results using CNN model architecture AlexNet
* vgg\_uploaded-images.txt - that contains the results using CNN model architecture VGG

To run file **run\_models\_batch\_uploaded.sh** in the workspace, open a terminal window within the **Project Workspace - Uploaded Images** and type the following, then hit enter:

sh run\_models\_batch\_uploaded.sh

This will run check\_images.py using all three model architectures to classify the four images in uploaded\_images folder outputting their results files into the workspace.

### Questions to Answer regarding Uploaded Image Classification:

Once the program stops running and the results files appear in the workspace, open and review each of the three to answer the following questions:

**1.** Did the three model architectures classify the breed of dog in Dog\_01.jpg to be the same breed? If not, report the differences in the classifications.

**2.** Did each of the three model architectures classify the breed of dog in Dog\_01.jpg to be the same breed of dog as that model architecture classified Dog\_02.jpg? If not, report the differences in the classifications.

**3.** Did the three model architectures correctly classify Animal\_Name\_01.jpg and Object\_Name\_01.jpg to **not** be dogs? If not, report the misclassifications.

**4.** Based upon your answers for questions **1.** - **3.** above, select the model architecture that you feel did the best at classifying the four uploaded images. Describe why you selected that model architecture as the best on uploaded image classification.

#### Project Workspace - Classify Uploaded Images can be found in the next concept.

# 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

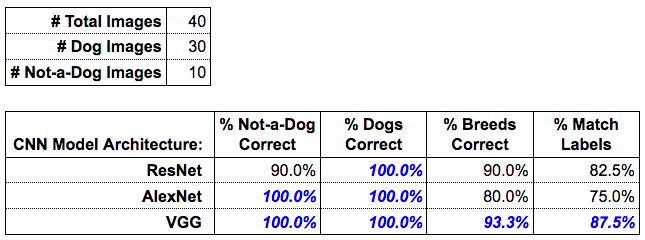
### In this project we had 2 main objectives:

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

Your program should have provided you with objectives 1 and 2 when it was run. In the table below, you will find our results for each of the model architectures. Your program should provide you with the same results as we have 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

**[[](https://classroom.udacity.com/nanodegrees/nd089/parts/e702bd72-f29e-4b33-a233-45af3b83d5fa/modules/9ec3aa45-b2bd-47ca-a513-31b21c852783/lessons/59930ed5-6f24-4cd2-bf0a-c0a0075cd7b4/concepts/c96c72c1-9d3f-4e92-92dc-e15c4c68ae20)](https://classroom.udacity.com/nanodegrees/nd089/parts/e702bd72-f29e-4b33-a233-45af3b83d5fa/modules/9ec3aa45-b2bd-47ca-a513-31b21c852783/lessons/59930ed5-6f24-4cd2-bf0a-c0a0075cd7b4/concepts/c96c72c1-9d3f-4e92-92dc-e15c4c68ae20)**

**[Project Results](https://classroom.udacity.com/nanodegrees/nd089/parts/e702bd72-f29e-4b33-a233-45af3b83d5fa/modules/9ec3aa45-b2bd-47ca-a513-31b21c852783/lessons/59930ed5-6f24-4cd2-bf0a-c0a0075cd7b4/concepts/c96c72c1-9d3f-4e92-92dc-e15c4c68ae20)**

Given our results, the "**best**" model architecture is VGG. It out performed 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.

**Use the next (last!) workspace to present and print your final results. You can use the table above as a reference**.