

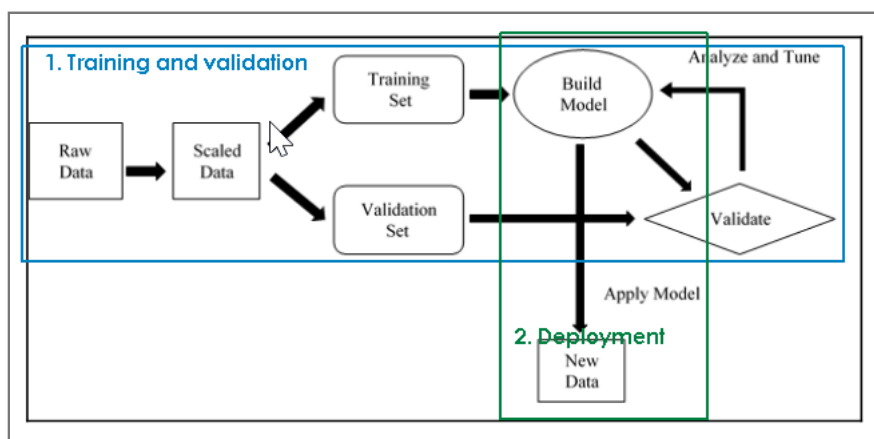
## HW5: ML

Total points: 5

This last hw is on supervised **machine learning**! As you now know, it's **data-related** (lots, and lots, and lots of it), after all :)

Here is a summary of what you'll do: **on Google's Colab (<https://colab.research.google.com/>), train a neural network on differentiating between a cat pic and dog pic, then use the trained network to classify a new (cat-like or dog-like) pic into a cat or dog.** This is a 'soup-to-nuts' (start to finish) assignment that will get your feet wet (or plunge you in!), doing ML - a VERY valuable skill - training a self-driving car (<http://apollo.auto/>), for example, would involve much more complexity, but would be based on the same workflow.

You are going to carry out 'supervised learning', as shown in this annotated graphic [from a book on TensorFlow]:

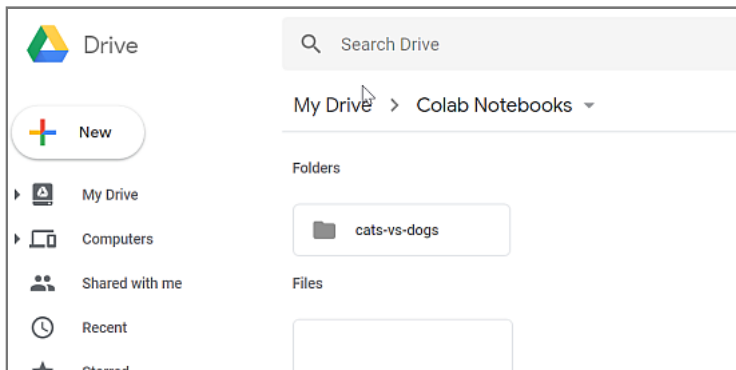


Below are the steps. Have fun!

1. Use your GMail/GDrive account to log in, go to <https://drive.google.com/> (<https://drive.google.com/>), click on the '+ New' button at the top left of the page, look for the 'Colab' app [after + New, click on More >, then + Connect more apps] and connect it - this will make the app [which connects to the mighty Google Cloud on the other end!] be able to access (read, write) files and folders in your GDrive.

2. You'll notice that the above step created a folder called Colab Notebooks, inside your GDrive - this is good, because we can keep Colab-related things nicely organized inside that folder. Colab is a cloud environment (maintained by Google), for executing Jupyter 'notebooks'. A Jupyter notebook (.ipynb extension, 'Iron Python Notebook') is a JSON file that contains a mix of two types of "cells" - text cells that have Markdown-formatted text and images, and code cells that contain, well, code :) The code can be in **Julia**, **Python**, or **R** (or several other languages, including JavaScript, with appropriate language 'plugins' (kernels) installed); for this HW, we'll use Python notebooks. Here (nb/RealtimeRO.ipynb) is a COVID-19 notebook. Download it (make sure that ends up on your machine as a .ipynb extension; if your downloading turned it into a .txt, rename it to be .ipynb), then drag and drop it into your Colab GDrive folder. Colab will open the notebook - look through it, to see a mix of text, equations (can contain figures, videos... too), and, most significantly, code. The code in our notebook uses data from <https://covidtracking.com/> (<https://covidtracking.com/>), to calculate 'RO' values. In Colab, do 'Runtime -> Run all' (pics/RealtimeRO.jpg) to run the code in all the cells; after the code is done running, scroll through, to see the results (several plots). **ALL the computations happened on the cloud, with data fetched by the code, from a URL (<https://covidtracking.com/api/v1/states/daily.csv>) (<https://covidtracking.com/api/v1/states/daily.csv>) - pretty neat!**

3. Within the Colab Notebooks subdir/folder, create a folder called cats-vs-dogs, for the hw:



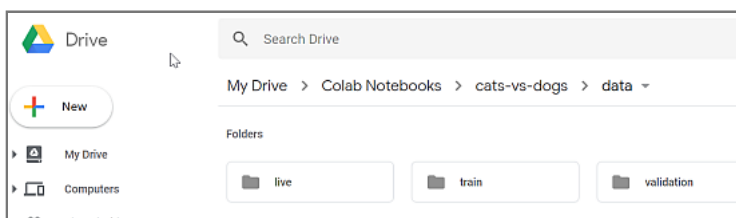
Now we need DATA [images of cats and dogs] for training and validation, and scripts for training+validation and classifying.

4. Download this (data/data.zip) .zip data file (~85MB), unzip it. You'll see this structure:

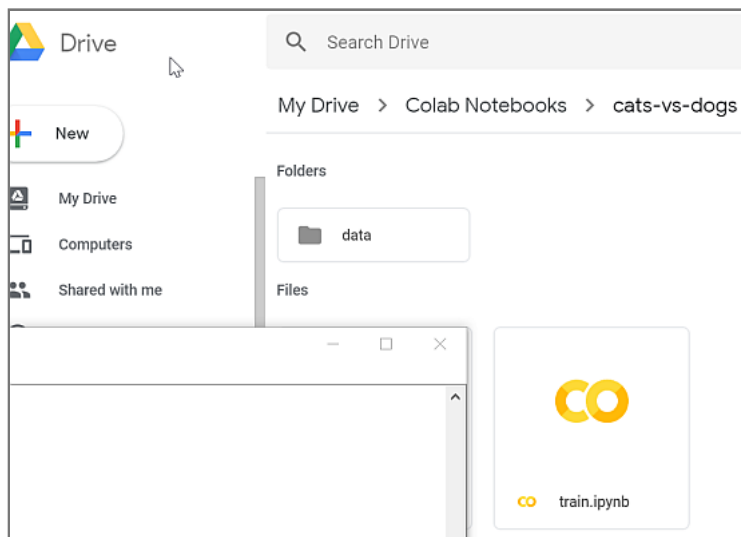
```
data/
  live/
  train/
    cats/
    dogs/
  validation/
    cats/
    dogs/
```

The train/ folder contains 1000 kitteh images under cats/, and 1000 doggo/pupper ones in dogs/. Have fun, looking at the adorable furballs :) Obviously **you** know which is which :) A neural network is going to start from scratch, and learn the difference, just based on these 2000 'training dataset' images. The validation/ folder contains 400 images each, of more cats and dogs - these are to feed the trained network, compare its classification answers to the actual answers so we can compute the accuracy of the training (in our code, we do this after each training epoch, to watch the accuracy build up, mostly monotonically). And, live/ is where you'd be placing new (to the NN) images of cats and dogs [that are not in the training or validation datasets], and use their filenames to **ask the network to classify them**: an output of 0 means 'cat', 1 means 'dog'. **Fun!**

Simply drag and drop the data/ folder on to your My Drive/Colab Notebooks/cats-vs-dogs/ area, and wait for about a half hour for the 2800 ( $2 \times (1000 + 400)$ ) images to be uploaded. After that, you should be seeing this [click inside the train/ and validation/ folders to see that the cats and dogs pics have been indeed uploaded]:



5. OK, **time to train a network!** Download this (nb/train.ipynb) Jupyter notebook. Drag and drop the notebook into cats-vs-dogs/:

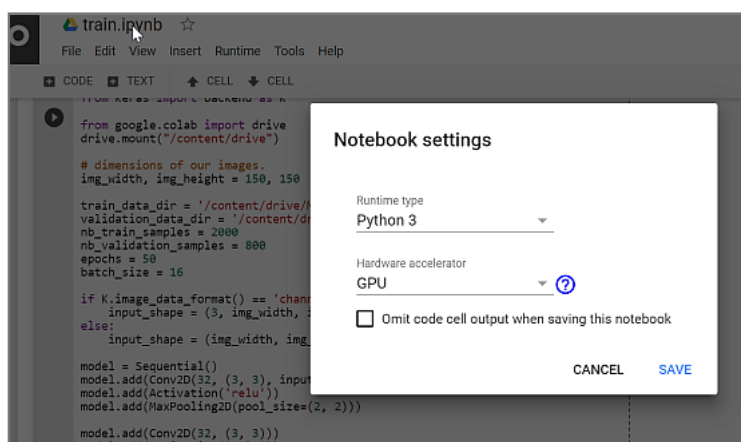


Double click on the notebook, that will open it so you can execute the code in the cell(s).

As you can see, it is a VERY short piece of code [not mine, except annotations and mods I made] where a network is set up [starting with 'model = Sequential()'], and the training is done using it [model.fit\_generator()]. In the last line, the RESULTS [learned weights, biases, for each neuron in each layer] are stored on disk as a weights.h5 file [a .h5 file is binary, in the publicly documented .h5 file format ([https://en.wikipedia.org/wiki/Hierarchical\\_Data\\_Format](https://en.wikipedia.org/wiki/Hierarchical_Data_Format)) (hierarchical, JSON-like, perfect for storing network weights)].

The code uses the Keras NN library (<https://keras.io/>), which runs on graph (dataflow) execution backends such TensorFlow(TF), Theano, CNTK [here we are running it over TF via the Google cloud]. With Keras, it is possible to express NN architectures succinctly (<https://www.datacamp.com/community/blog/keras-cheat-sheet>) - the TF equivalent (or Theano's etc.) would be more verbose. As a future exercise, you can try coding the model in this hw, directly in TF or Theano or CNTK - you should get the same results.

Before you run the code to kick off the training, note that you will be using GPU acceleration on the cloud (**results in ~10x speedup**) - cool! You'd do this via 'Edit -> Notebook settings'. In this notebook, this is already set up (by me), but you can verify that it's set:

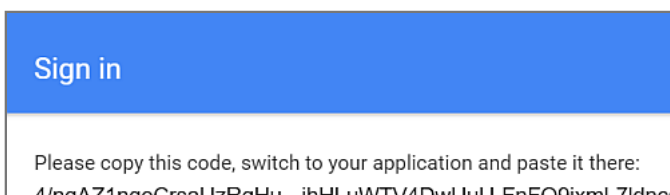
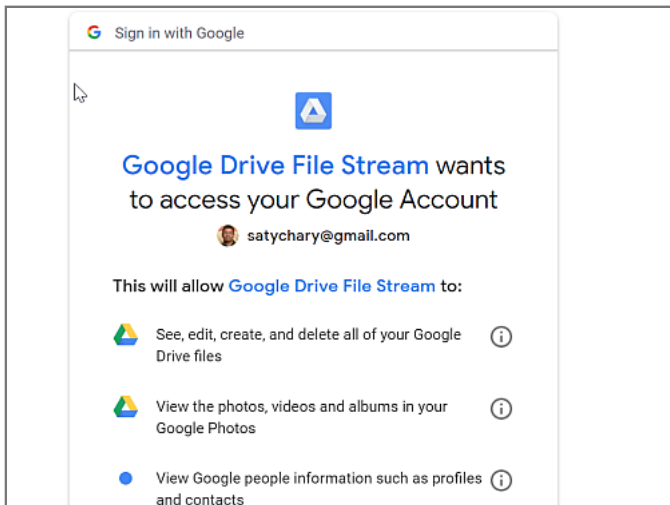


When you click on the circular 'play' button at the left of the cell, the training will start - here is a sped-up version of what you will get (your numerical values will be different):

0:00 / 0:11

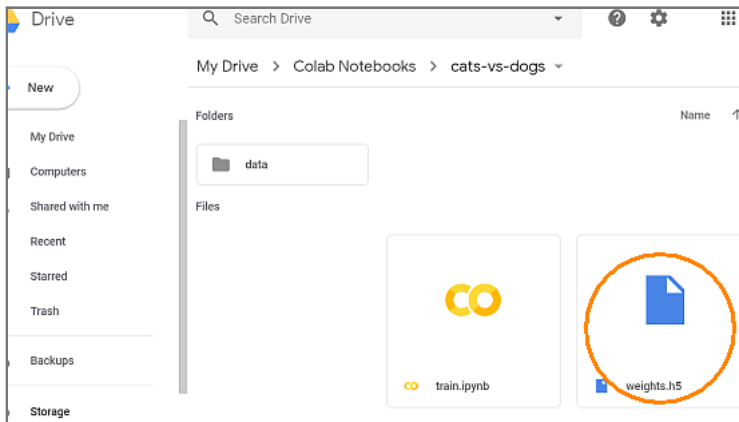
The backprop loop runs 50 times ('epochs' (<https://keras.io/getting-started/faq/#what-does-sample-batch-epoch-mean>)) through all the training data. The acc: column shows the accuracy [how close the training is, to the expected validation/ results], which would be a little over 80% - NOT BAD, for having learned from just 1000 input images for each class!

Click the play button to execute the code! The first time you run it (and anytime after logging out and logging back in), you'd need to authorize Colab to access GDrive - so a message will show up, under the code cell, asking you to click on a link whereby you can log in and provide authorization, and copy and paste the authorization code that appears. Once you do this, the rest of the code (where the training occurs) will start to run.

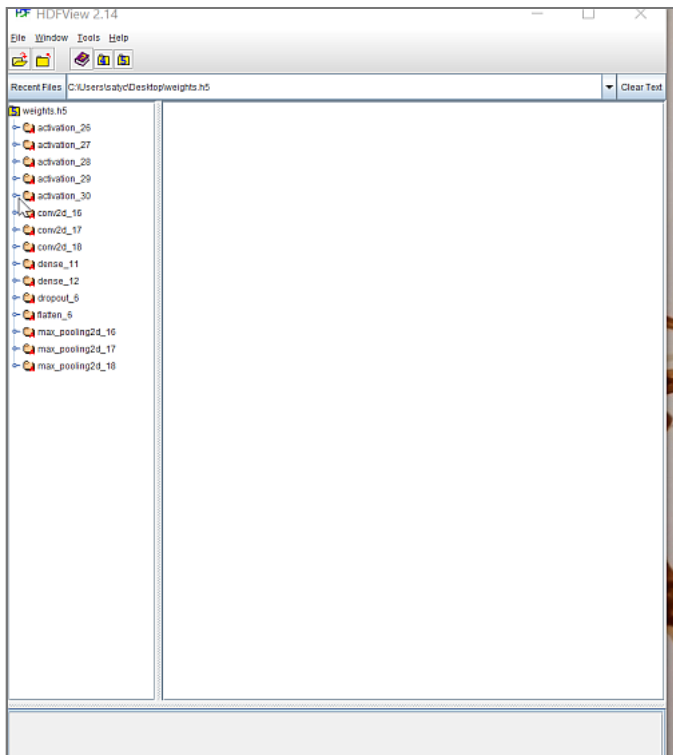


Scroll down to below the code cell, to watch the training happen. As you can see, it is going to take a short while.

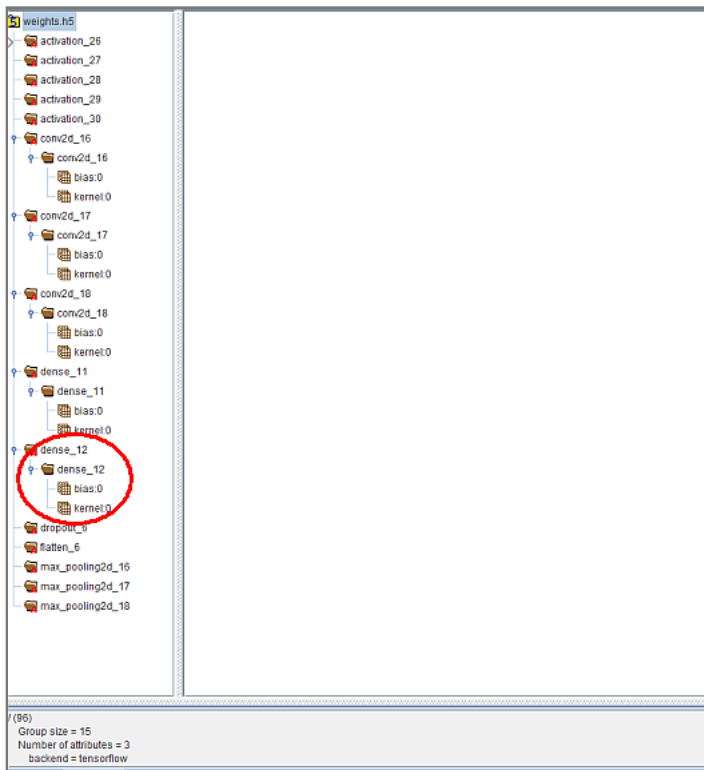
After the 50th epoch, we're all done training (and validating too, which we did 50 times, once at the end of each epoch). **What's the tangible result, at the end of our training+validating process? It's a 'weights.h5' file!** If you look in your cats-vs-dogs/ folder, it should be there:



6. Soooo, what exactly [format and content-wise] is in the weights file? You can find out, by downloading HDFView-2.14.0, from <https://support.hdfgroup.org/products/java/release/download.html> (grab the binary, from the 'HDFView+Object 2.14' column on the left). Install, and bring up the program. Download the .h5 file from GDrive to your local area (eg. desktop), then drag and drop it into HDView:



Right-click on weights.h5 at the top-left, and do 'Expand All':



Neat! We can see the NN columns, and the biases and weights (kernels) for each. Double click on the bias and kernel items in the second (of the two) dense layers [dense\_12, in my case - yours might be named something else], and stagger them so you can see both:

Recent Files C:\Users\atyc\Desktop\weights.h5

weights.h5

activation\_26  
activation\_27  
activation\_28  
activation\_29  
activation\_30  
conv2d\_16  
conv2d\_17  
conv2d\_18  
conv2d\_19  
conv2d\_20  
conv2d\_21  
conv2d\_22  
conv2d\_23  
conv2d\_24  
conv2d\_25  
conv2d\_26  
conv2d\_27  
conv2d\_28  
conv2d\_29  
conv2d\_30  
conv2d\_31  
conv2d\_32  
conv2d\_33  
conv2d\_34  
conv2d\_35  
conv2d\_36  
conv2d\_37  
conv2d\_38  
conv2d\_39  
conv2d\_40  
conv2d\_41  
conv2d\_42  
conv2d\_43  
conv2d\_44  
conv2d\_45  
conv2d\_46  
conv2d\_47  
conv2d\_48  
conv2d\_49  
conv2d\_50  
conv2d\_51  
conv2d\_52  
conv2d\_53  
conv2d\_54  
conv2d\_55  
conv2d\_56  
conv2d\_57  
conv2d\_58  
conv2d\_59  
conv2d\_60  
conv2d\_61  
conv2d\_62  
conv2d\_63  
conv2d\_64  
conv2d\_65  
conv2d\_66  
conv2d\_67  
conv2d\_68  
conv2d\_69  
conv2d\_70  
conv2d\_71  
conv2d\_72  
conv2d\_73  
conv2d\_74  
conv2d\_75  
conv2d\_76  
conv2d\_77  
conv2d\_78  
conv2d\_79  
conv2d\_80  
conv2d\_81  
conv2d\_82  
conv2d\_83  
conv2d\_84  
conv2d\_85  
conv2d\_86  
conv2d\_87  
conv2d\_88  
conv2d\_89  
conv2d\_90  
conv2d\_91  
conv2d\_92  
conv2d\_93  
conv2d\_94  
conv2d\_95  
conv2d\_96  
conv2d\_97  
conv2d\_98  
conv2d\_99  
conv2d\_100

bias:0 at /dense\_12/dense\_12/ [weights.h5 in C:\Users\atyc\Desktop]

0	-0.31984454

kernel:0 at /dense\_12/dense\_12/ [weights.h5 in C:\Users\atyc\Desktop]

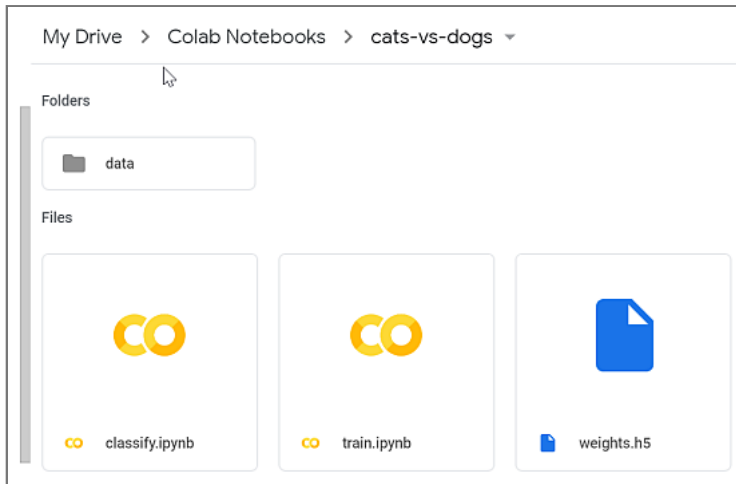
0	-0.08492154
1	-0.0132735...
2	0.1943794
3	0.0280992
4	-0.15635538
5	-0.1305849
6	-0.08386392
7	-0.10358289
8	-0.0450458...
9	0.0729693
10	0.0275401
11	0.07731976
12	0.04825445
13	-0.08906876
14	0.0617936...
15	-0.0649955
16	0.06401146
17	-0.0158660
18	-0.05812193
19	-0.0553165
20	0.0810956
21	-0.08057273
22	-0.05910004
23	0.08653338
24	-0.24019071
25	0.0922715
26	-0.06354873
27	0.07198966
28	-0.1000175...
29	-0.06385387
30	-0.06385
31	-0.0701839
32	0.20593775
33	-0.0626298
34	-0.12051652
35	0.15784958
36	0.07647735
37	-0.0947011

**Computing those floating point numbers is WHAT -EVERY FORM- OF NEURAL NETWORK TRAINING IS ALL ABOUT!** A self-driving car, for example, is also trained the same way, resulting in weights that can classify live traffic data (scary, in my opinion). Here, collectively (taking all layers into account), **it's those floating point numbers that REPRESENT the network's "learning" of telling apart cats and dogs!** The "learned" numbers (the .h5 weights file, actually) can be sent to anyone, who can instantiate a new network (with the same architecture as

the one in the training step), and simply re/use the weights in weights.h5 to start classifying cats and dogs right away - no training necessary. The weight arrays represent "catness" and "dogness", in a sense :) We would call the network+weights, a 'pre-trained model'. In a self-driving car, the weights would be copied to the processing hardware (<https://www.wired.com/story/self-driving-cars-power-consumption-nvidia-chip/>) that resides in the car.

**Q1** [0.5+0.5=1 point]. Submit your weights.h5 file. Also, create a submittable screengrab similar to the above [showing values for the second dense layer (eg. dense\_12)]. For fun, click around, examine the arrays in the other layers as well. Again, it's all these values that are the end result of training, on account of iterating and minimizing classification errors through those epochs.

7. Now for the fun part - finding out how well our network has learned! Download this (nb/classify.ipynb) Jupyter notebook, and upload it to your cats-vs-dogs/ Colab area:



When you open classify.ipynb, you can see that it contains Keras code to read the weights file and associate the weights with a new model (which needs to be 100% identical to the one we had set up, to train), then take a new image's filename as input, and **predict (model.predict()) whether the image is that of a cat [output: 0], or a dog [output: 1]!** Why 0 for cat and 1 for dog? Because 'c' comes before 'd' alphabetically [or because (pics/purple.png)] :)

Supply (upload, into live/) a what1.jpg cat image, and what2.jpg dog image, then execute the cell. Hopefully you'd get a 0, and 1 (for what1.jpg and what2.jpg, respectively). The images can be any resolution (size) and aspect ratio (squamishness), but nearly-square pics would work best. Try this with pics of your pets, your neighbors', images from a Google search, even your drawings/paintings... **Isn't this cool? Our little network can classify!**

Just FYI, note that the classification code in classify.ipynb could have simply been inside a new cell in train.ipynb instead. The advantage of multiple code cells inside a notebook, as opposed to multiple code blocks in a script, is that in a notebook, code cells can be independently executed one at a time (usually sequentially) - so if both of our programs were in the same notebook, we would run the training code first (just once), followed by classification (possibly multiple times); a script on the other hand, can't be re/executed in parts.

**Q2** [2 points]. Create a screenshot that shows the [correct] classification (you'll also be submitting your what{1,2}.jpg images with this).

What about misclassification? After all, we trained with "just" 1000 (not 1000000) images each, for about an 80% accurate prediction. What if we input 'difficult' images, of a cat that looks like it could be labeled a dog, and the other way around? :)

**Q3** [2 points]. Get a 'Corgi' image [the world's smartest (<https://www.buzzfeed.com/mjs538/why-corgis-are-the-smartest-animals>) dogs!], and a 'dog-like' cat image [hint, it's all about the ears!], upload to live/, attempt to (mis)classify, ie. create incorrect results (where the cat pic outputs a 1, and the dog's, 0), make a screenshot. Note that you need to edit the code to point myPic and myPic2 to these image filenames.

**No-points bonus.** Add a third class, eg. racoon ([https://www.google.com/search?q=racoon&rlz=1C1CHBF\\_enUS723US723&sxsrf=ACYBGNQEveXxDEN19VOiT6UfOQg94Wpe1Q:1574885052359&source=to\\_the\\_training](https://www.google.com/search?q=racoon&rlz=1C1CHBF_enUS723US723&sxsrf=ACYBGNQEveXxDEN19VOiT6UfOQg94Wpe1Q:1574885052359&source=to_the_training)). You'd need to create (acquire) (pics/img\_scraping.jpg) your own training data (500 images might be

sufficient, or, even fewer, eg. 300), and **modify the code** just a tiny bit in order to train, and test, on this third class, too. The third class can even be a non-animal, eg. trash (cardboard or plastic or metal or glass), ball, furniture, book... anything at all! Here (nb/train3Classes.ipynb) is the training script, and here (nb/classify3Classes.ipynb) is the classifying script.

---

Here's a checklist of what to submit [**as a single .zip file**]:

- weights.h5, and a screenshot from HDFView
  - your 'good' cat and dog pics, and screenshot that shows proper classification
  - your 'trick' cat and dog pics, and screenshot that shows misclassification
  - if you do the bonus question: your extra data (images) folder for the third class, a 'new' third class image you used for classification, a screenshot that shows correct classification of it, and a README with the code mods you made (this can be actual code with comments, or just a description) - note that the bonus question carries 0 points, so you can skip it if you like
- 

All done - hope you had fun, and learned a lot!

Note - you can continue using Colab to run all sorts of notebooks (<https://github.com/jupyter/jupyter/wiki/A-gallery-of-interesting-Jupyter-Notebooks>) [on Google's cloud GPUs!], including ones with TensorFlow, Keras, PyTorch... etc. ML code.

---