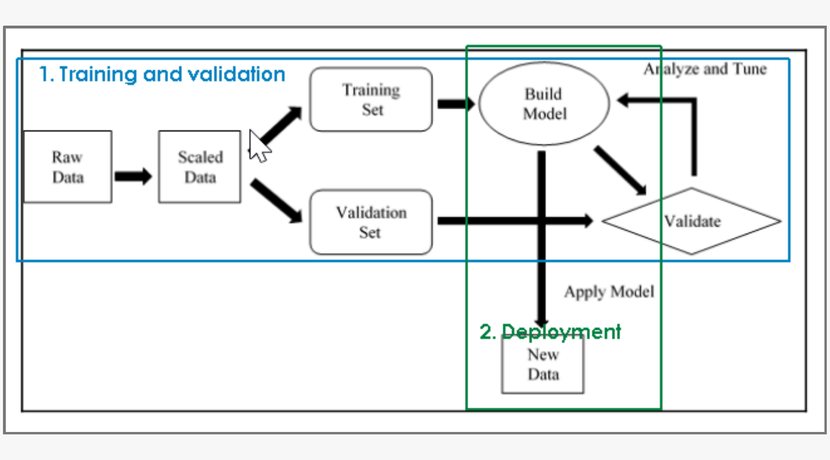
HW5: ML

Total points: 6 (+1... the total will be capped at 6)

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, 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, 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]:

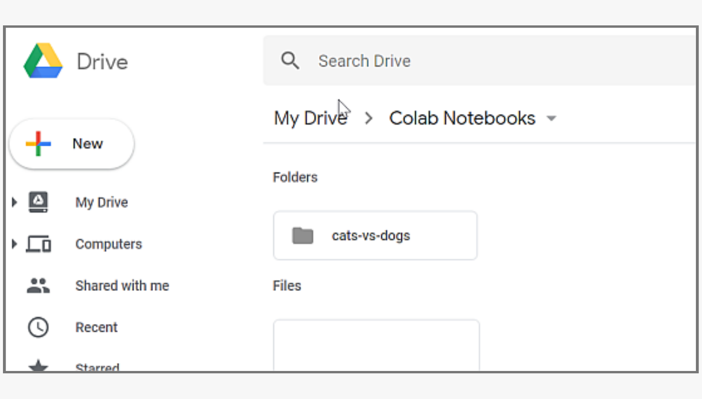


Below are the steps. Have fun!

1. Use your GMail/GDrive account to log in, go to https://drive.google.com/, 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 **Ju**lia, **Pyt**hon, or **R** (or several other languages, including JavaScript, with appropriate language 'plugins' (kernels) installed); for this HW, we'll use Python notebooks.

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



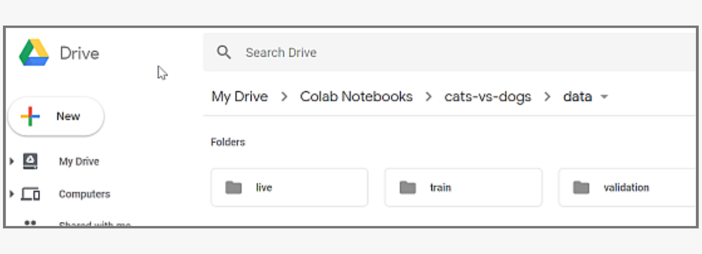
Now we need DATA [images of cats and dogs] for training and validation, and scripts for training+validation and classifying.

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

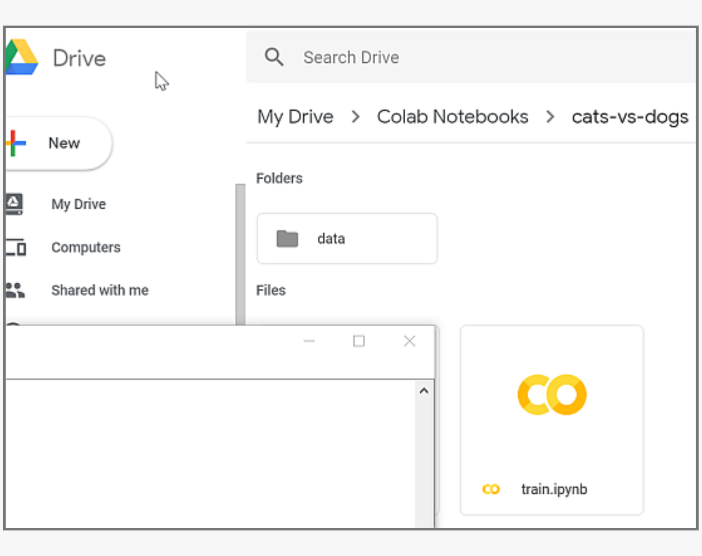
```
data/
  live/
  train/
    cats/
    dogs/
  validation/
    cats/
    dogs/
```

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*(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 Jupyter notebook. Drag and drop the notebook into cats-vs-dogs/:

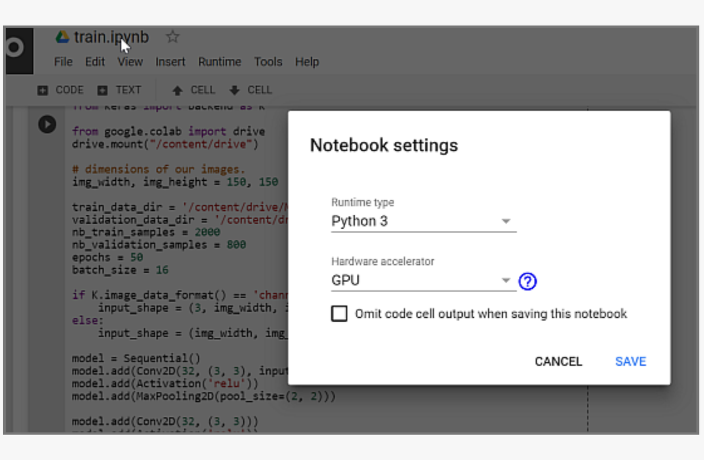


Double click on the notebook, that will open it so you can execute the code in the cell(s).

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 .hd5 file format (hierarchical, JSON-like, perfect for storing network weights)].

The code uses the Keras NN library, 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 succintly - 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.

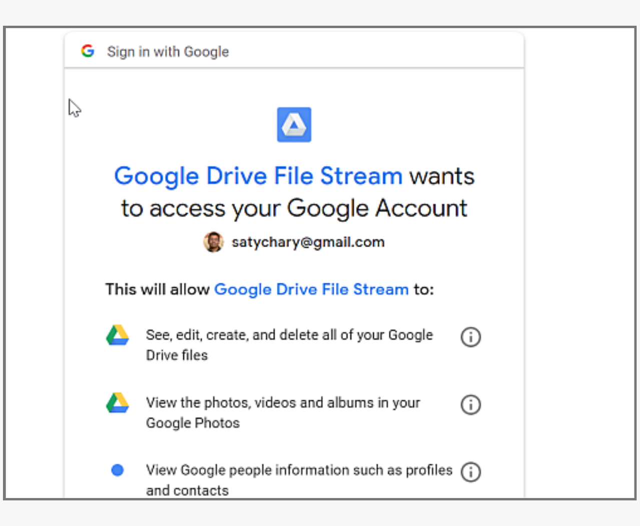
Before you run the code to kick off the training, note that you will be using GPU acceleration on the cloud (**results in ~10x speedup**) - cool! You'd do this via 'Edit -> Notebook settings'. In this notebook, this is already set up (by me), but you can verify that it's set:

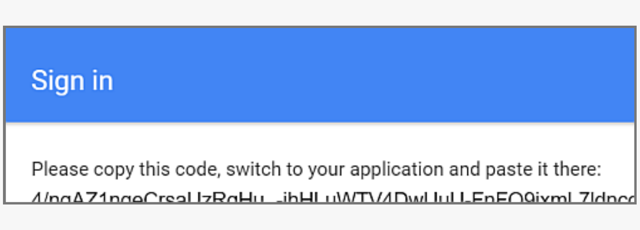


When you click on the circular 'play' button at the left of the cell, the training will start - here is a sped-up version of what you will get (your numerical values will be different):

The backprop loop runs 50 times ('epochs') 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!

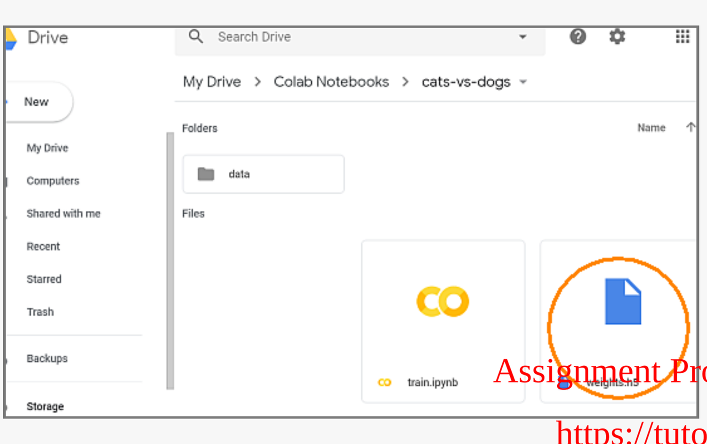
Click the play button to execute the code! The first time you run it (and anytime after logging out and logging back in), you'd need to authorize Colab to access GDrive - so a message will show up, under the code cell, asking you to click on a link whereby you can log in and provide authorization, and 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.



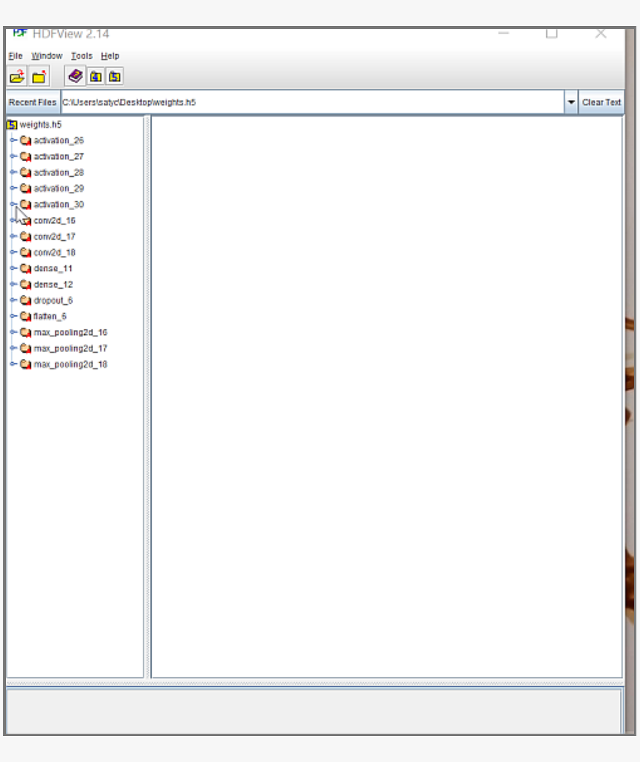


Scroll down to below the code cell, to watch the training happen. As you can see, it is going to take a short while.

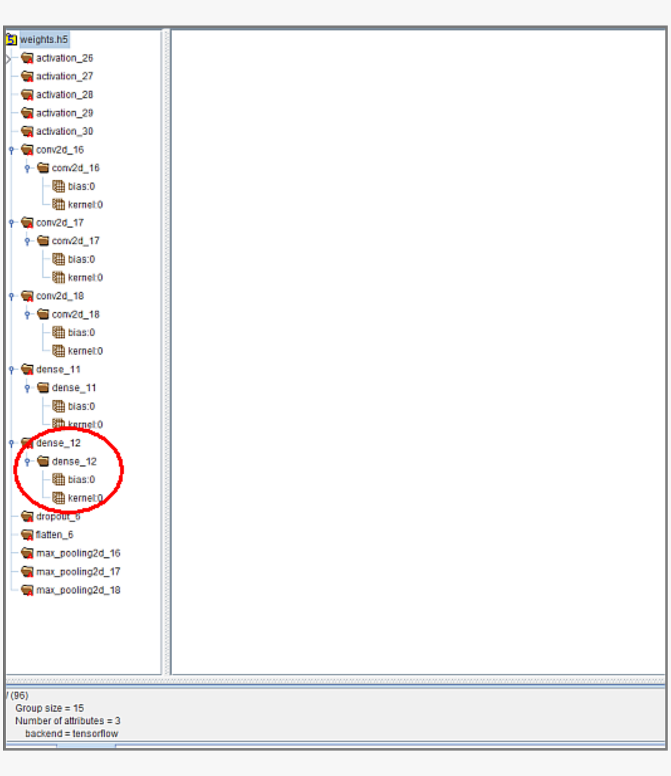
After the 50th epoch, we're all done training (and validating too, which we did 50 times, once at the end of each epoch). **What's the tangible result, at the end of our training+validating process? It's a 'weights.h5' file!** If you look in your cats-vs-dogs/ folder, it should be there:



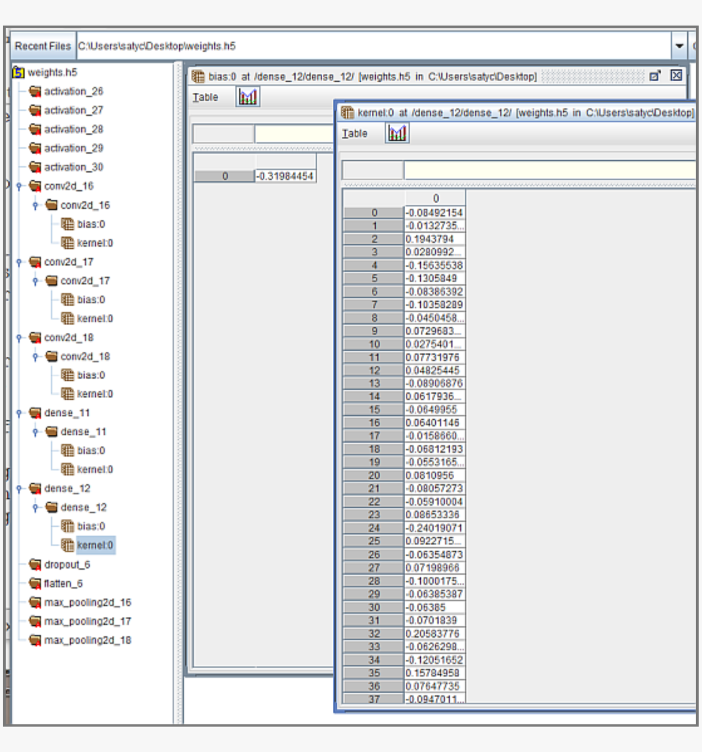
6. Soooo, what exactly [format and content-wise] is in the weights file? You can find out, by downloading HDFView-2.14.0, from https://support.hdfgroup.org/products/java/release/download.html [grab the binary, from the 'HDFView+Object 2.14' column on the left]. Install, and bring up the program. Download 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':



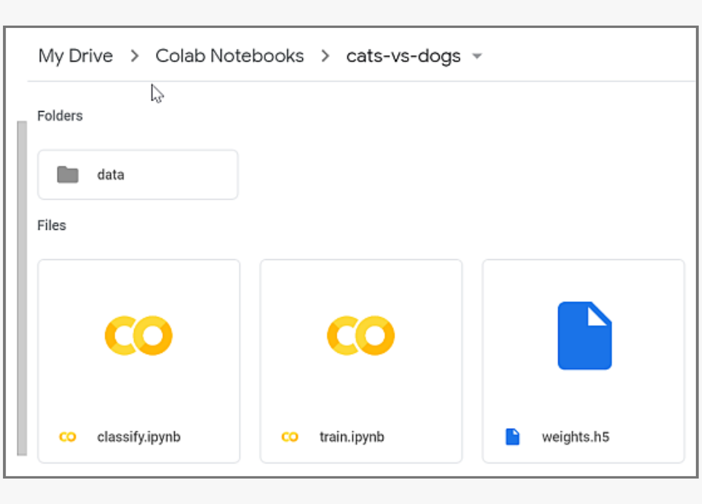
Neat! We can see the NN columns, and the biases and weights (kernels) for each. Double click on the bias and kernel items in the second (of the two) dense layers [dense_12, in my case - yours might be named something else], and stagger them so you can see both:



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 that resides in the car.

Q1 [1+1=2 points]. 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 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] :)

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 (squarishness), 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.

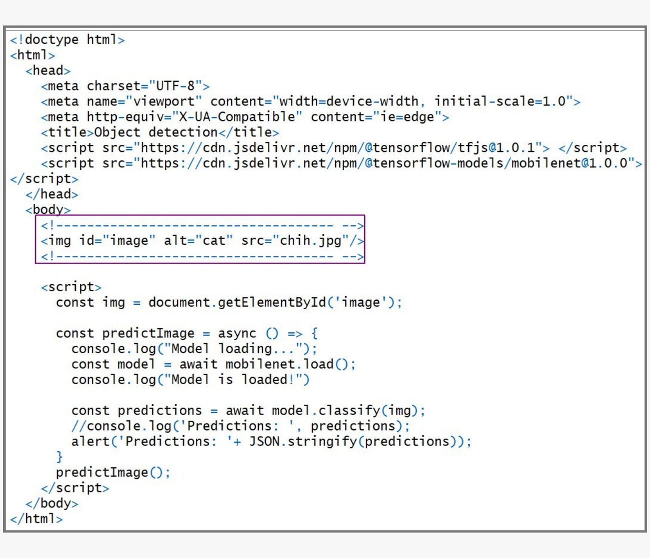
Q2 [2 points]. Create a screenshot that shows the [correct] classification (you'll also be submitting your what{1,2}.jpg images with this).

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 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.

1 point bonus. MobileNet-based detection!

Click on this web page, wait a few sec (for an alert to pop up) - dog detection! Download the html, and edit it - you'll see a small bunch of markup:

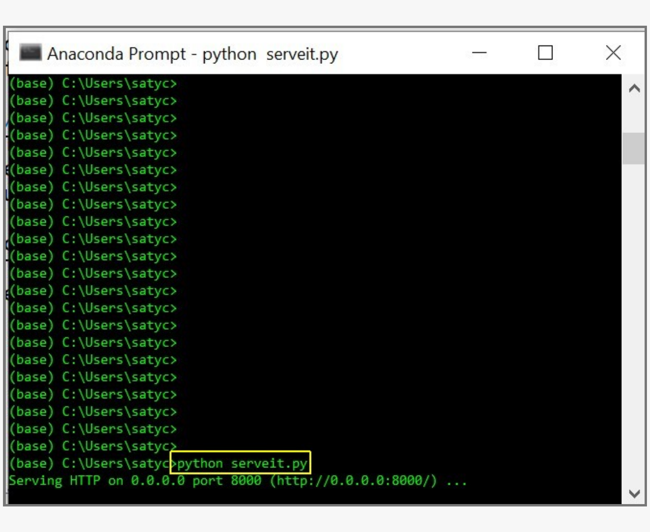


```
<!doctype html>
<html>
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Object detection</title>
    <script src="https://cdn.jsdelivr.net/npm/@tensorflow/tfjs@1.0.1"> </script>
    <script src="https://cdn.jsdelivr.net/npm/@tensorflow-models/mobilenet@1.0.0">
</script>
  </head>
  <body>
    <!------------------------------------ -->
    <img id="image" alt="cat" src="chih.jpg"/>
    <!------------------------------------ -->

    <script>
      const img = document.getElementById('image');

      const predictImage = async () => {
        console.log("Model loading...");
        const model = await mobilenet.load();
        console.log("Model is loaded!")

        const predictions = await model.classify(img);
        //console.log('Predictions: ', predictions);
        alert('Predictions: '+ JSON.stringify(predictions));
      }
      predictImage();
    </script>
  </body>
</html>
```

The page loads TensorFlow, and a pre-trained MobileNet model, off a CDN. For the detection, we specify a local image, using 'src='. You can see I'm specifying 'chih.jpg', a Chihuahua pic I downloaded off the web.

You are going to get your own Chihuahua CLOSEUP (crop what you download if you need to), and a muffin, put in each (one after another), load the page in your browser, get screenshots of the detection.

But first, you need to install Miniconda so that you can run a local server - look here under 'Installing Miniconda'. After Miniconda is installed, bring up a shell, place this script in the directory where you are (in the shell) and type the following to start a webserver ('python serveit.py '):



```
Anaconda Prompt - python  serveit.py
(base) C:\Users\satyc>
(base) C:\Users\satyc>python serveit.py
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
```

After starting the webserver, on your (Chrome) browser, go to localhost:8000/, navigate to your .html - you will see that your pic is loaded, and labeled! Take screenshots of a correct Chihuahua IDing, correct muffin IDing (as 'bakery, bakeshop', not 'muffin', because MobileNet isn't trained on muffins), and a mis-IDing (Chihuahua face closeup misclassified as a muffin (ie. 'bakery, bakeshop'), or the other way around, where a muffin gets labeled as a dog (ANY breed)).

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: three screenshots

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

Note - you can continue using Colab to run all sorts of notebooks [on Google's cloud GPUs!], including ones with TensorFlow, Keras, PyTorch... etc. ML code.