(https://www.nvidia.com/en-us/deep-learning-ai/education/)

Deployment

Which files constitute a "model"?

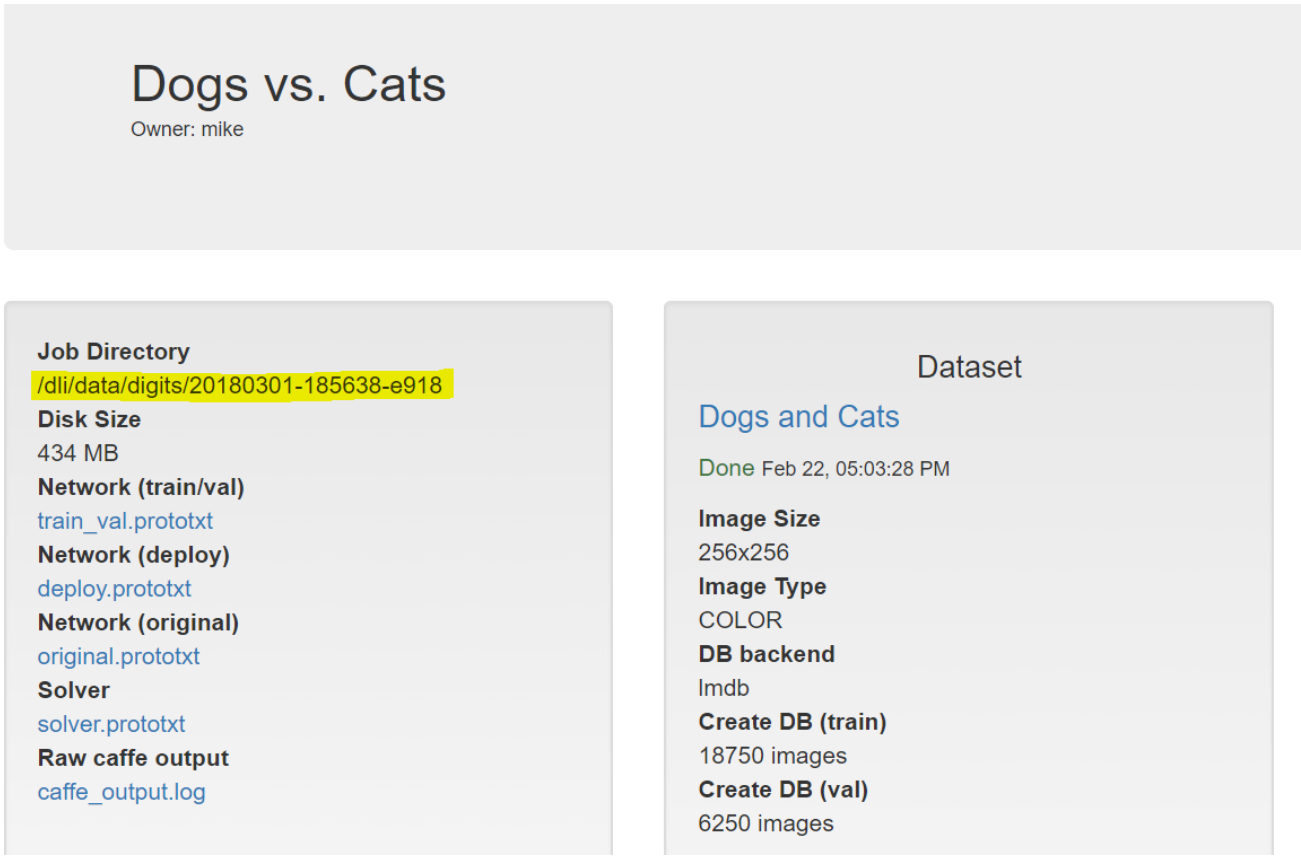
We make a trained network useful by removing it from its training environment and "deploying" it into an application. Start where we left off in DIGITS.

DIGITS places the files we need to deploy in a directory that can either be downloaded or just pointed to. Since we're going to be deploying our model on the same server where it was trained, we can just point to the folder path that DIGITS generates.

Open DIGITS (/digits).

From DIGITS home page, select the model that we named "Dogs vs. Cats".

DIGITS' "Job Page" for the model is what you see as soon as you create the model, when it is training, and/or if you select the model under DIGITS' "model" tab. The Job Directory is in the top left.



Copy the job directory (highlighted above) and replace ##FIXME## in the code block below. Once you've copied the directory, execute the cell (Shift+Enter) to store it to the variable `MODEL_JOB_DIR`

In [ ]:
```
MODEL_JOB_DIR = '##FIXME##'  ## Remember to set this to be the job directory for your m
!ls $MODEL_JOB_DIR
```

Assuming you copied and pasted well, you will see a list of all files in that directory. If the following instructions do not match what you're seeing, check the copy/paste directions.

Again, our "model" consists of two files: the architecture and the weights.

The architecture is the file called `deploy.prototxt` and the weights are in the most recent snapshot file `snapshot_iter_#.caffemodel`. In this case, snapshot number 735 contains the weights learned after all 5 epochs.

In [ ]:
```
ARCHITECTURE = MODEL_JOB_DIR + '/' + 'deploy.prototxt'
WEIGHTS = MODEL_JOB_DIR + '/' + 'snapshot_iter_735.caffemodel'
print ("Filepath to Architecture = " + ARCHITECTURE)
print("Filepath to weights = "+ WEIGHTS)
```

Next, we need to make sure that the program that we're building can both read and process those files. For this basic type of deployment, we'll need to install (or include) the framework that they were written in to be able to interpret them. We'll learn to deploy to environments that don't require installing the framework later in this course. We'll also need to use the GPU to take advantage of parallel processing. Again, our model consists of hundreds of thousands of operations that can be largely accelerated through parallelization.

In [ ]:
```
import caffe
caffe.set_mode_gpu()
```

Next, we'll create a "Classifier" object called "net". The more common the workflow, the easier existing tools will make your project. In this case, image classification is very common, so this next code block simply takes your architecture file and weights file and a bit about the data and makes common actions easy.

In [ ]:
```
# Initialize the Caffe model using the model trained in DIGITS
net = caffe.Classifier(ARCHITECTURE, WEIGHTS,
                       channel_swap =(2, 1, 0), #Color images have three channels, Red,
                       raw_scale=255) #Each pixel value is a number between 0 and 255
                       #Each "channel" of our images are 256 x 256
```

The Classifier class includes a method called "predict", which takes an input of an image as defined above and generates an output of the likelihood of the image belonging to each category.

Creating an Expected Input: Preprocessing

To start with something easy, let's attempt to correctly classify a labeled image from the dataset. We can load the image and view it by running the cell below.

In [ ]:
```
import matplotlib.pyplot as plt #matplotlib.pyplot allows us to visualize results
input_image= caffe.io.load_image('/dli/data/dogscats/train/cats/cat.10941.jpg')
plt.imshow(input_image)
plt.show()
```

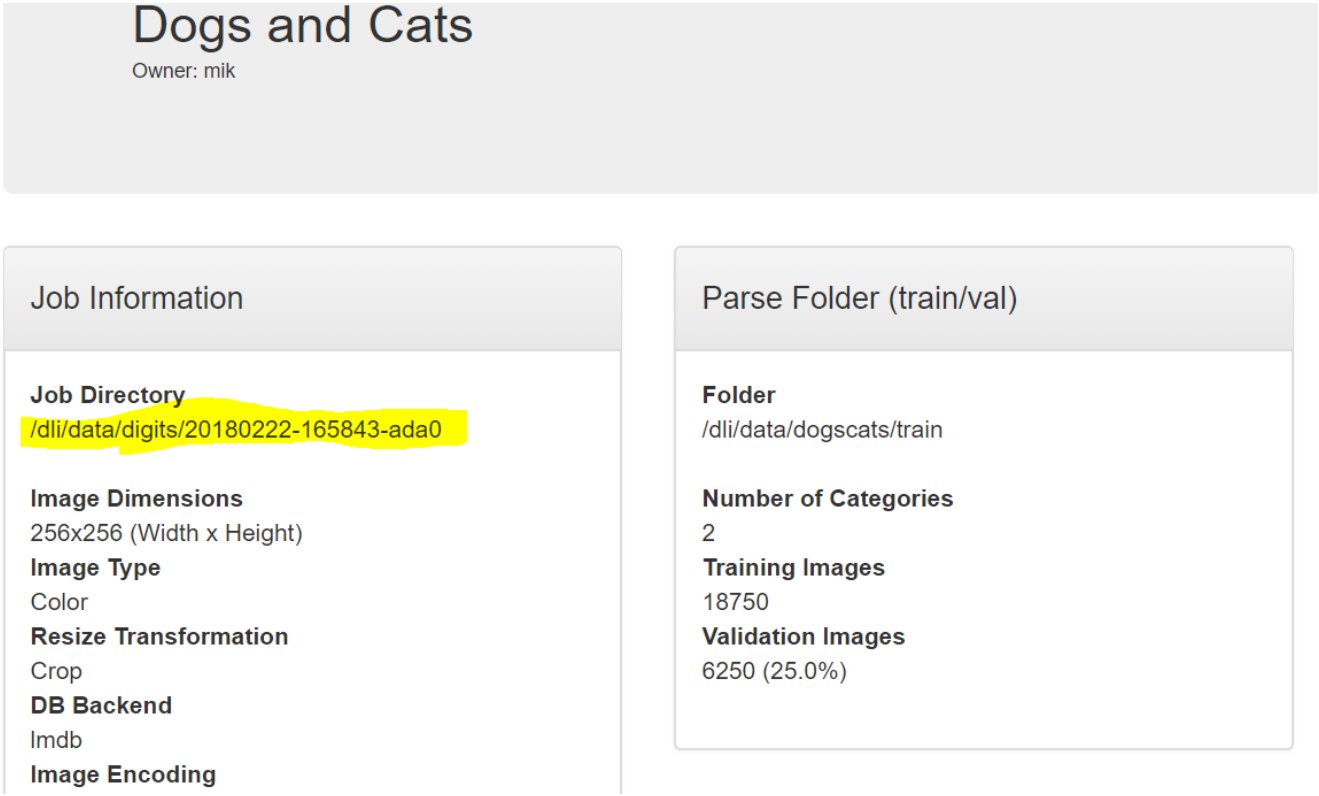
While this is the image we have, it is not the 'input' the network expects.

To prepare data for inference, we're going to follow one golden rule:

> Whatever was done prior to training must be done prior to inference

In the last section, you saw the files that were generated when DIGITS trained your model. In this section, we'll examine the files generated when DIGITS created your dataset.

The job directory for the **dataset** you just trained from is found by selecting the dataset from the model page "Dogs and Cats" and/or if you select the dataset under DIGITS' "dataset" tab. It's in the same place it was for the model, but should be a different number.



Replace ##FIXME## with it and execute the code below to set DATA_JOB_DIR to the right filepath and examine what's inside:

In [ ]:
```
DATA_JOB_DIR = '##FIXME##'  ## Remember to set this to be the job directory for your mo
!ls $DATA_JOB_DIR
```

Again, there is more information here than you need (for now). There is an infinite amount that you *could* know about data science and data prep which will become clear as you work through a variety of deep learning problems. In this case, DIGITS did two steps prior to training. We call this *preprocessing.*

1) DIGITS resized the images to 256X256 color images

In [ ]:
```
import cv2
input_image=cv2.resize(input_image, (256, 256), 0,0)
plt.imshow(input_image)
plt.show()
```

2) DIGITS *normalized* the images by subtracting the mean image from each image to reduce the computation necessary to train.

Load the mean image and subtract it from the test image below:

In [ ]:
```
mean_image = caffe.io.load_image(DATA_JOB_DIR+'/mean.jpg')
ready_image = input_image-mean_image
```

We've now taken data as it was and converted it into data that our network expects. Next, let's see what output our network creates.

Forward Propagation: Using your model

This is what we care about. Let's take a look at the function:
`prediction = net.predict([grid_square])` .

Like any function (https://www.khanacademy.org/computing/computer-programming/programming#functions), `net.predict` passes an input, `ready_image` , and returns an output, `prediction` . Unlike other functions, this function isn't following a list of steps, instead, it's performing layer after layer of matrix math to transform an image into a vector of probabilities.

Run the cell below to see the prediction from labeled the labeled data above.

In [ ]:
```
# make prediction
prediction = net.predict([ready_image])
print prediction
```

Interesting, but doesn't contain all that much information. Our network took a normalized 256x256 color image and generated a vector of length 2.

Generating a useful output: Postprocessing

At this point, we can really build whatever we want. Your only limit is your programming experience. Before getting creative, let's build something basic. This code will determine whether our network output a higher value for the likelihood of "dog" than it did for "cat." If so, it will display an image that would be appropriate if a dog approached our simulated doggy door. If not, the image represents what we'd want to happen if our network determined a cat was at the door.

In [ ]:
```
print("Input image:")
plt.imshow(input_image)
plt.show()

print("Output:")
if prediction.argmax()==0:
    print "Sorry cat:( https://media.giphy.com/media/jb8aFEQk3tADS/giphy.gif"
else:
    print "Welcome dog! https://www.flickr.com/photos/aidras/5379402670"
```

Here, now is everything in one place so you can test with an image that a doggy door might see.

In [ ]:
```
##Create an input our network expects
input_image= caffe.io.load_image('/dli/data/fromnest.PNG')
input_image=cv2.resize(input_image, (256, 256), 0,0)
ready_image = input_image-mean_image
##Treat our network as a function that takes an input and generates an output
prediction = net.predict([ready_image])
print("Input Image:")
plt.imshow(input_image)
plt.show()
print(prediction)
##Create a useful output
print("Output:")
if prediction.argmax()==0:
    print "Sorry cat:( https://media.giphy.com/media/jb8aFEQk3tADS/giphy.gif"
else:
    print "Welcome dog! https://www.flickr.com/photos/aidras/5379402670"
```

Essentially, we've created a simulator for our doggy door challenge. We've created an application that takes an input from a camera, converts it to a data type our network expects, generates an output, and then converts that output into something useful to a user.

You could see how you might easily have a positive output control a motor in a doggy door. With regards to deep learning, you have what you need! To see what other images you can try in the code block above, `list` the test images (images that weren't used for training) can be found by running the command below. Expect some of these images to output the wrong classification. Test them until you're satisfied and then continue in the course to find out how to improve performance!

In [ ]:
```
!ls /dli/data/dogscats/test
```

Putting it all together

Let's put this deployment process together to see how it might look outside of this Jupyter notebook. In the Python file at pythondeployment.py (../../../../edit/tasks/task3/task/pythondeployment.py), you'll see the same code as above, but consolidated into one file. You'll use this approach during your end of course assessment, so take a look. Insert the filepath to a test image here to visualize it.

In [ ]:
```
TEST_IMAGE = '/dli/data/dogscats/test/1.jpg'
display= caffe.io.load_image(TEST_IMAGE)
plt.imshow(display)
plt.show()
```

And then run our small python application with that image as input below. Ignore most of the output and scroll to the bottom. (Even errors and warnings are fine.)

In [ ]:
```
!python pythondeployment.py $TEST_IMAGE 2>/dev/null
```

(https://www.nvidia.com/en-us/deep-learning-ai/education/)