FUNDAMENTALS OF DEEP LEARNING FOR COMPUTER VISION

1. Biological inspiration –

This section explains the difference between the workflow of Classical Machine learning and Deep Learning. Machine Learning involves hand designed feature engineering. In Deep Learning, the algorithm itself learn complex features and representation of the data by finding a mapping function between input and output.

Solving problems with deep learning requires identifying some pattern in the world, finding examples that highlight both sides of the pattern (the input and the output), and then letting a "neural network" learn the map between the two. This opens the types of problems where computers can help us to those where we:

1. Have identified a pattern within a problem
2. Have enough data that exemplifies the pattern

In the first game (Louie or not louie) we just performed a fundamental task of computer vision: image classification, where a computer can take an image that it has never seen before and correctly attribute it to the appropriate "class".

GPU Task 1:

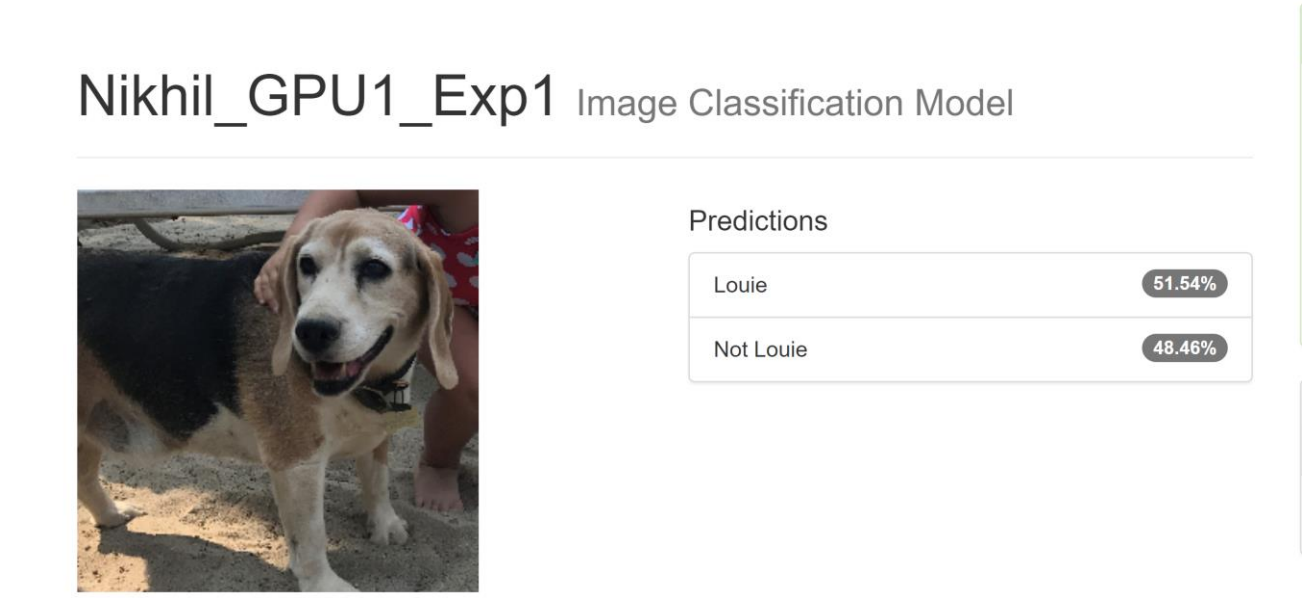
First, I created a *model* by choosing a *dataset(Images of beagles)*, a *neural network,* and how many *training epochs* to run. This highlighted two concepts:

1. Deep Neural Networks are flexible algorithms inspired by the human brain that allow practitioners to use training strategies inspired by human learning. The input of an image generated an output of the network's confidence that the image belonged to one of two classes.
2. Deep Neural Networks fall very short of the human brain. After one epoch, our model was predicting no better than chance: 50/50.

In this task, the model looked at each image once. Each time it saw an image, it generated an output, or prediction, about whether that image contains Louie or does not contain Louie. After each prediction, it learned the right answer, adjusted its model, and then tried again.

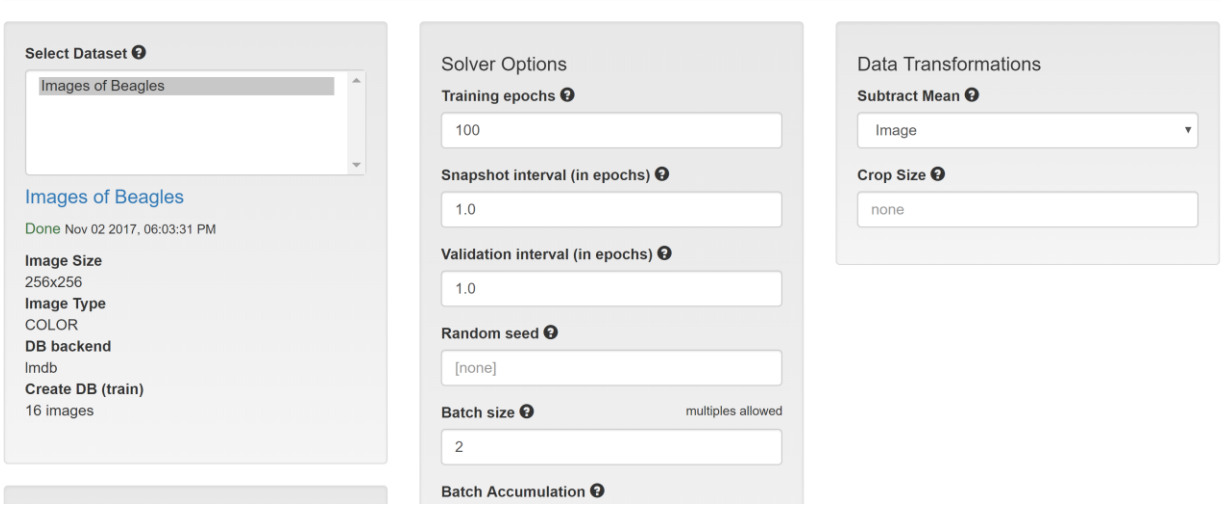
Experiment 1 :

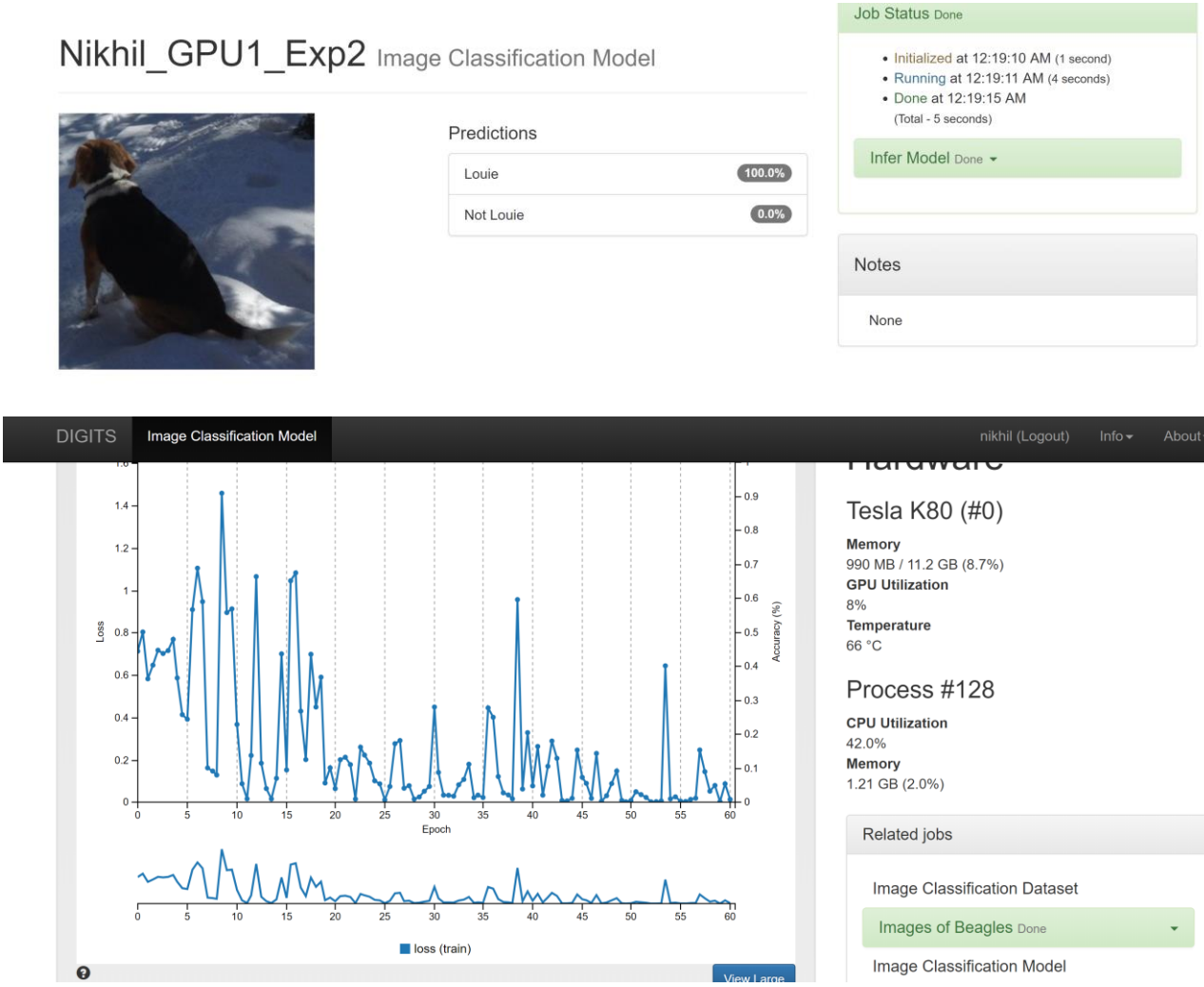
Batch size – 2 , Epochs = 5 , Learning rate = 0.01 , Optimizer = SGD, Network = LeNet



Experiment 2 :

Batch size – 2 , Epochs = 100 , Learning rate = 0.001 , Optimizer = SGD, Network = AlexNet





Experiment 3 :

Batch size – 2 , Epochs = 100 , Learning rate = 0.005 , Optimizer = Adadelta, Network = GoogleNet

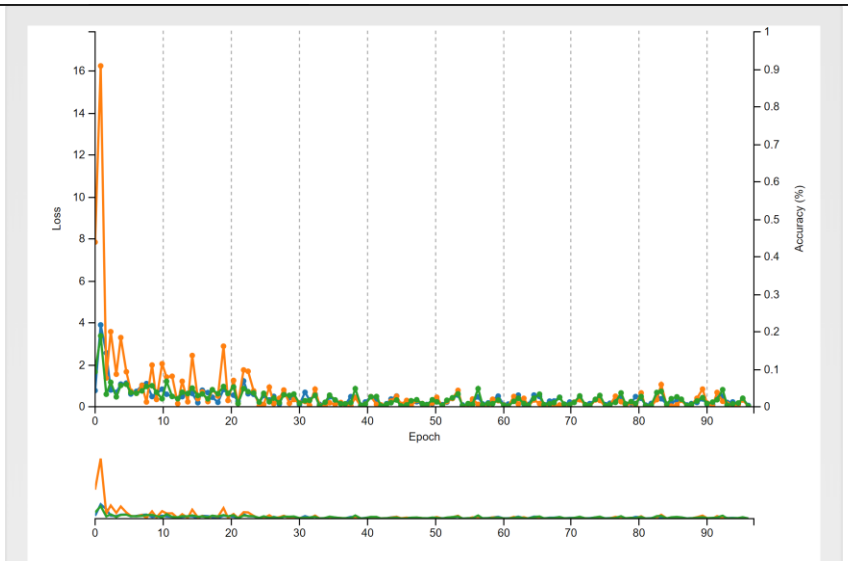


Image Classification Model

Nikhil_GPU1_Exp1 Done

Image Classification Model

Nikhil_GPU1_Exp2 Done

Notes

None

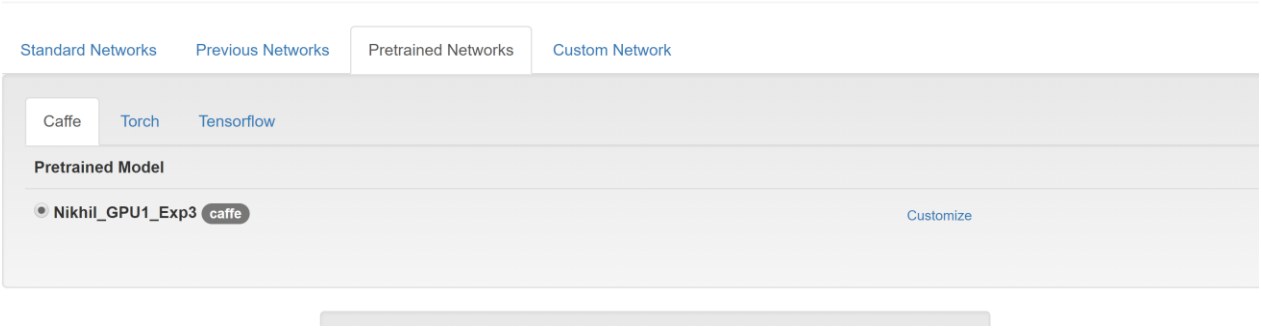
Nikhil_GPU1_Exp3 Image Classification Model

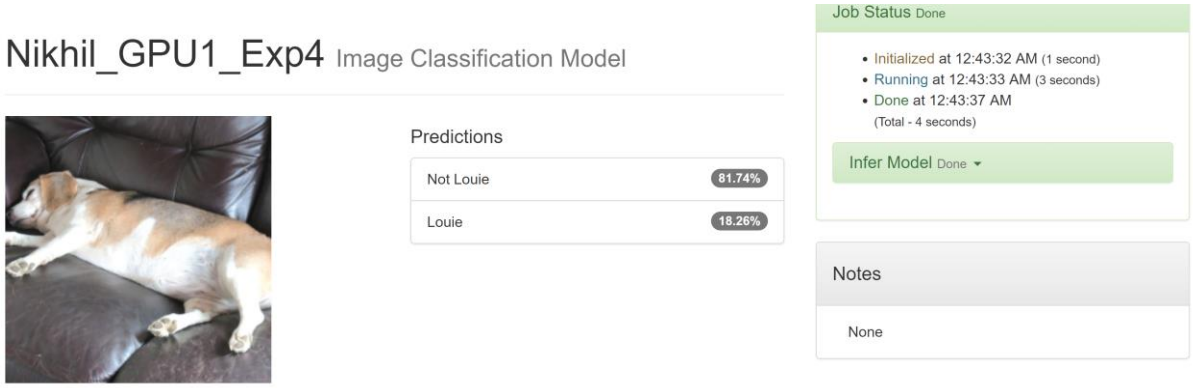
Predictions

Louie	93.69%
Not Louie	6.31%

Experiment 4 :

Batch size – 2 , Epochs = 100 , Learning rate = 0.001 , Optimizer = Adam, Network = Pretrained Network GPU_Exp3





Out of all the models, AlexNet network model performed the best with a 100% accuracy.

GPU TASK 2 : BIG DATA

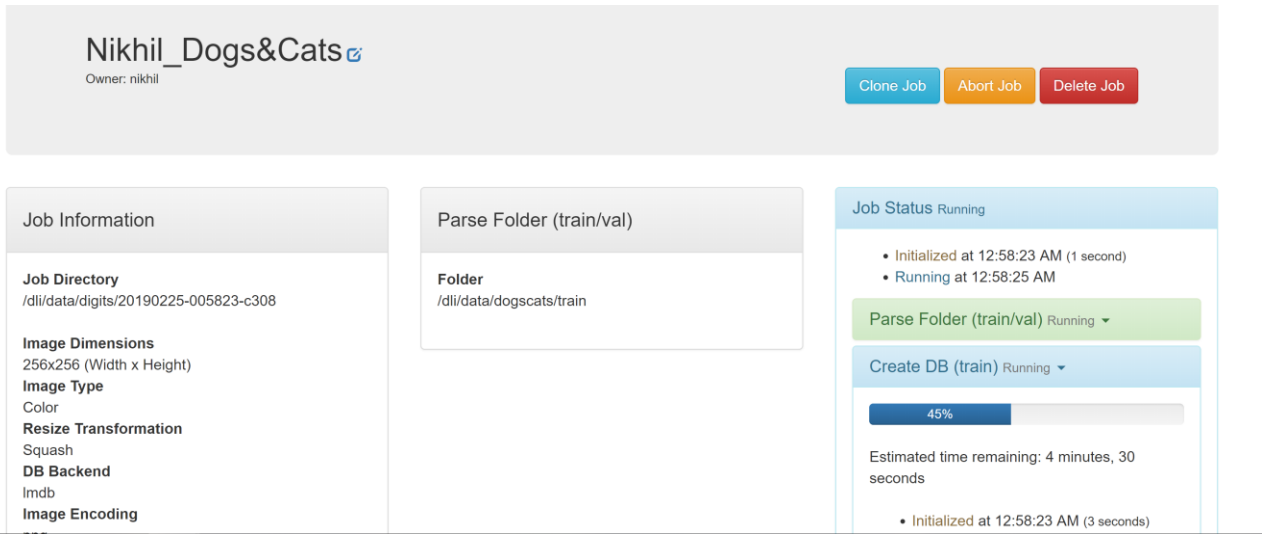
In this Task, we combined these three below ingredients to train a neural network that is effective in the real world. We'll measure our success by the model's performance on new data.

1. Deep Neural Networks
2. The GPU
3. Big data

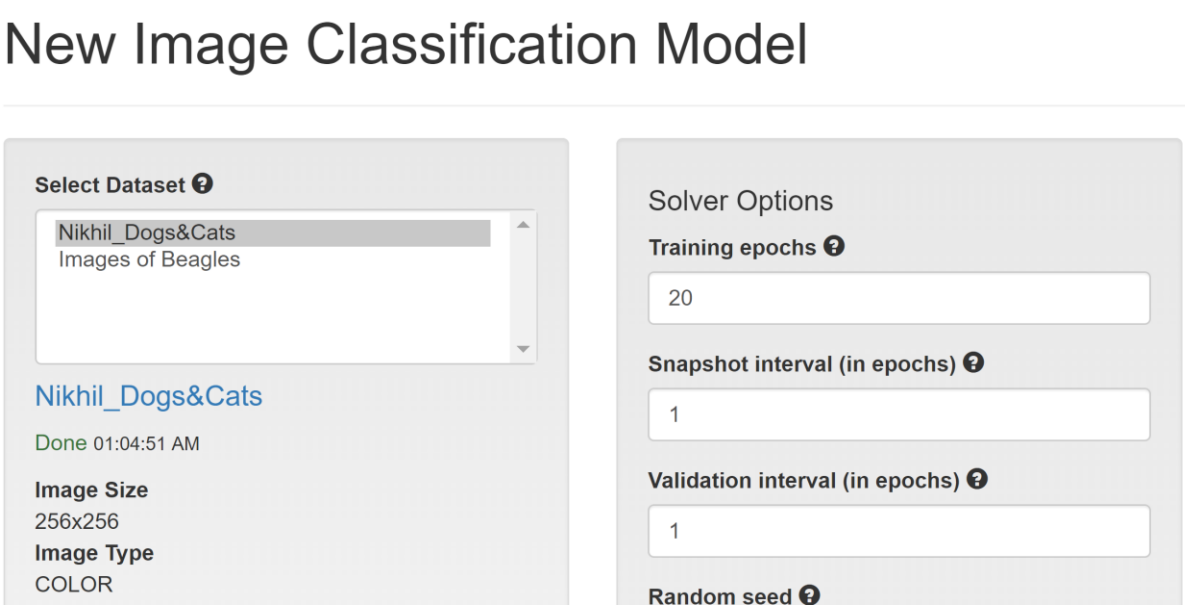
The data that a neural network is exposed to during training is the only guidance it has about the world. When you learned to identify Louie, you began with an understanding of not only dogs, but even of features that might differentiate between dogs (color, size, etc.) A neural network still just interprets data as data, so has to discover which features are useful.

Creating a model that is effective with *new data*

Dataset – Nikhil_Cats&Dogs

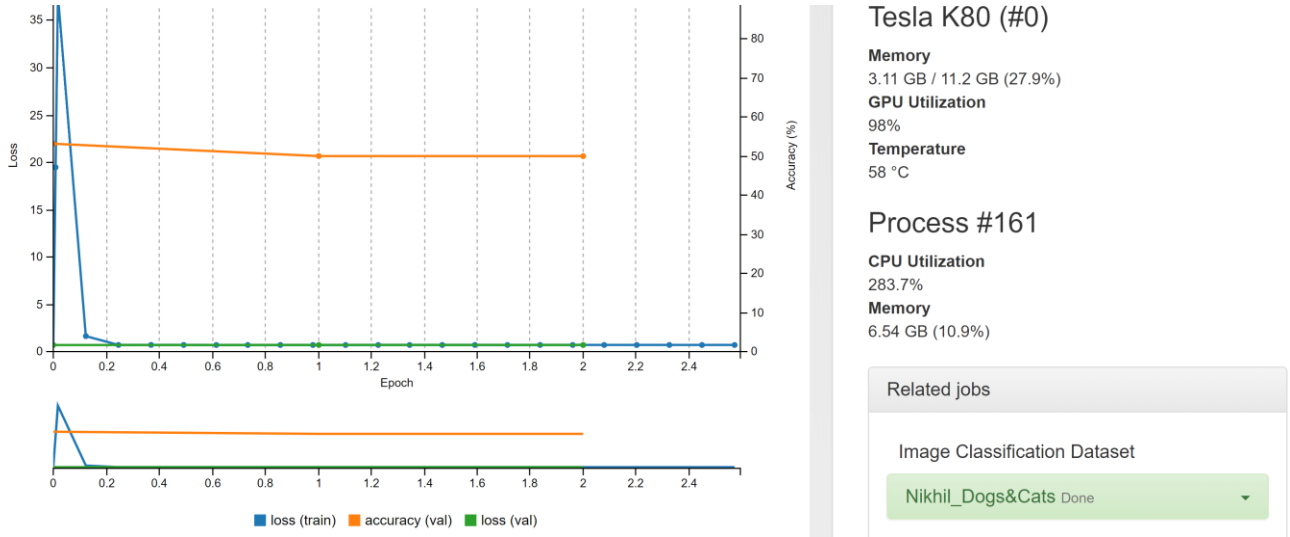


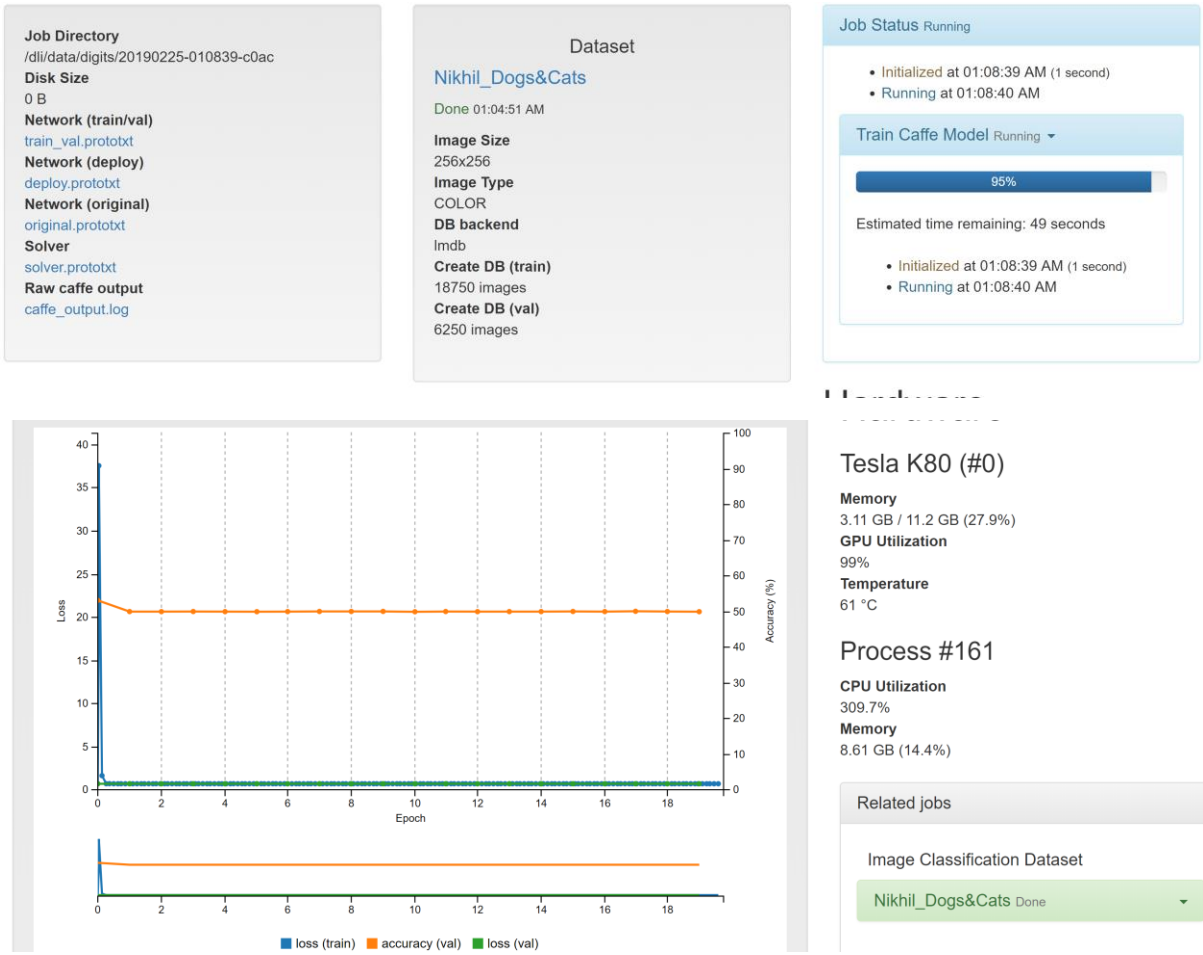
Let's Create Models now and train it on this Dataset and test on 25% validation data from this dataset.



Experiment 1 –

Batch size – 2, Epochs = 20, Learning rate = 0.01 , Optimizer = Adam, Network = AlexNet





With 20 epochs, it took 16 minutes to train the model.

Lets test using a unseen image - /dli/data/BeagleImages/Louie/louie5.JPG

Nikhil_GPU2_Exp1 Image Classification Model

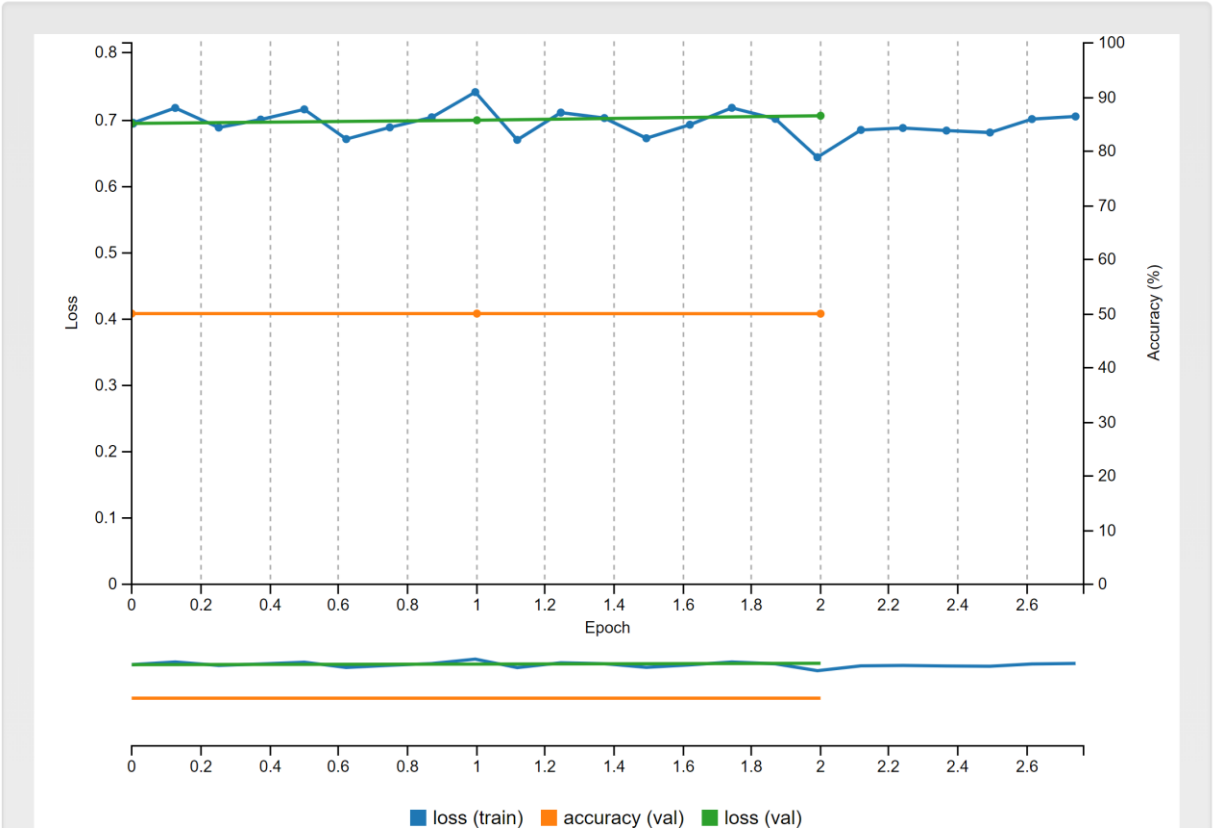
Predictions

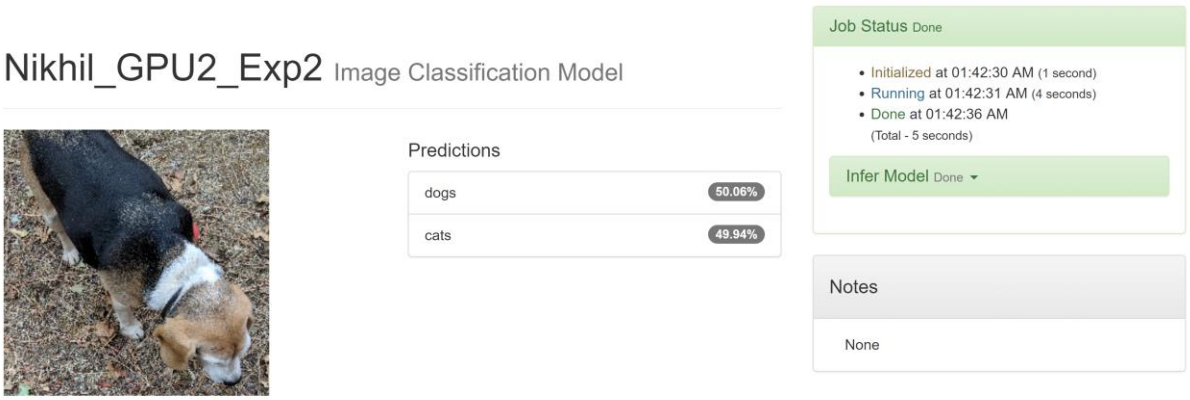
dogs	50.01%
cats	49.99%

Accuracy was 50.01%. Lets change some hyperparameters and try again .

Experiment 2 –

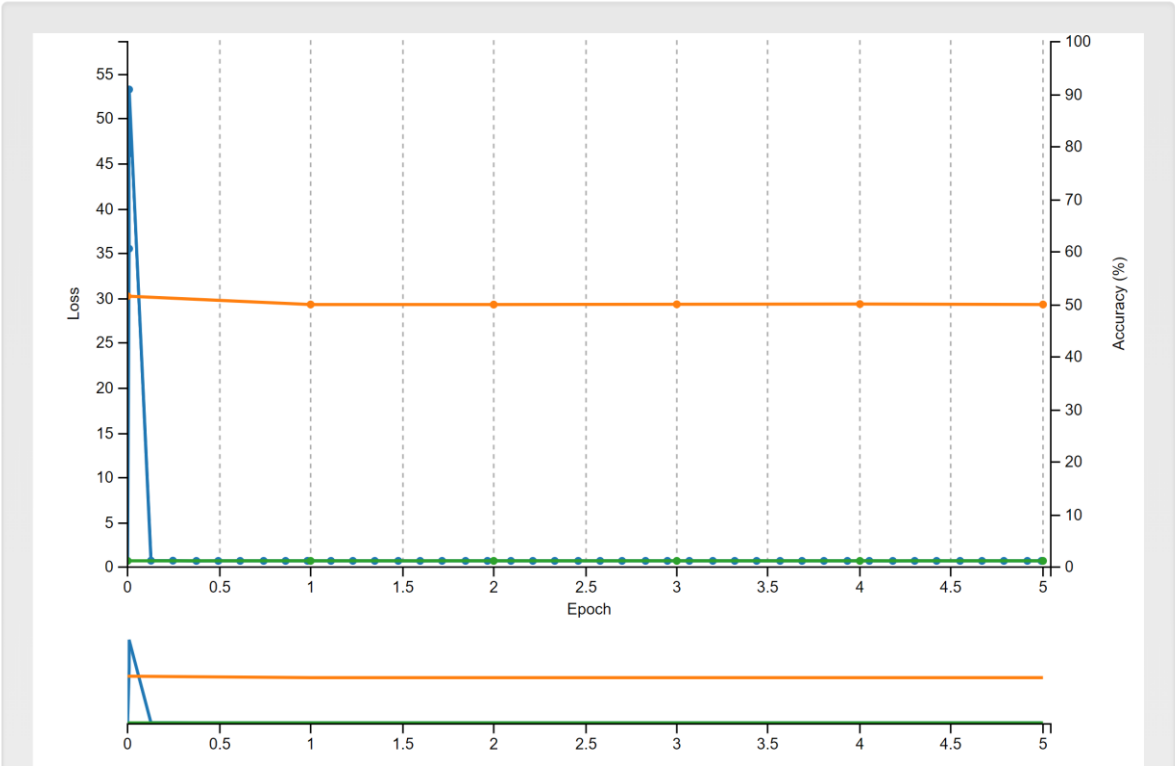
Batch size – 16, Epochs = 10, Learning rate = 0.01 , Optimizer = Adam, Network = Pretrained Network GPU2_Exp1





Experiment 3 –

Batch size – 64, Epochs = 5, Learning rate = 0.005 , Optimizer = AdaGrad, Network = AlexNet



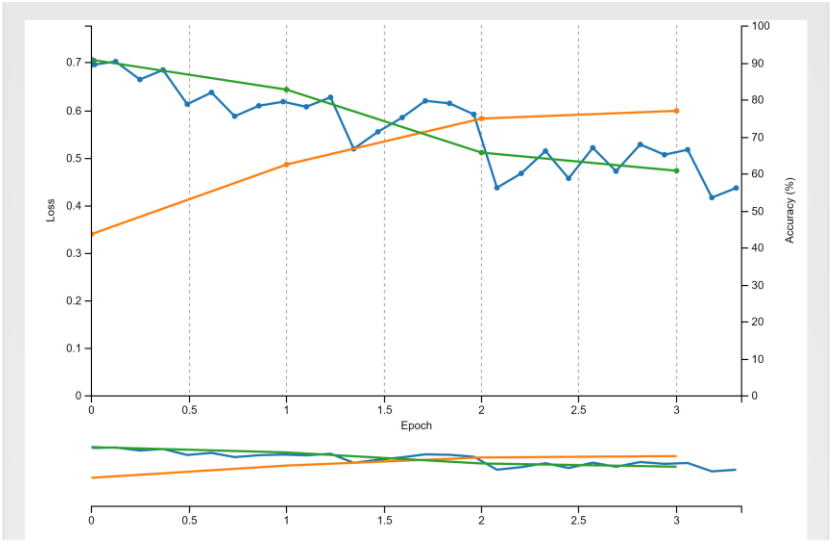
Nikhil_GPU2_Exp3 Image Classification Model

Predictions

cats	50.09%
dogs	49.91%

Experiment 4 –

Batch size – , Epochs = 5, Learning rate = 0.001 , Optimizer = Adam, Network = AlexNet , Subtract Mean – Image



Hardware

Tesla K80 (#0)

Memory
2.9 GB / 11.2 GB (25.9%)
GPU Utilization
99%
Temperature
65 °C

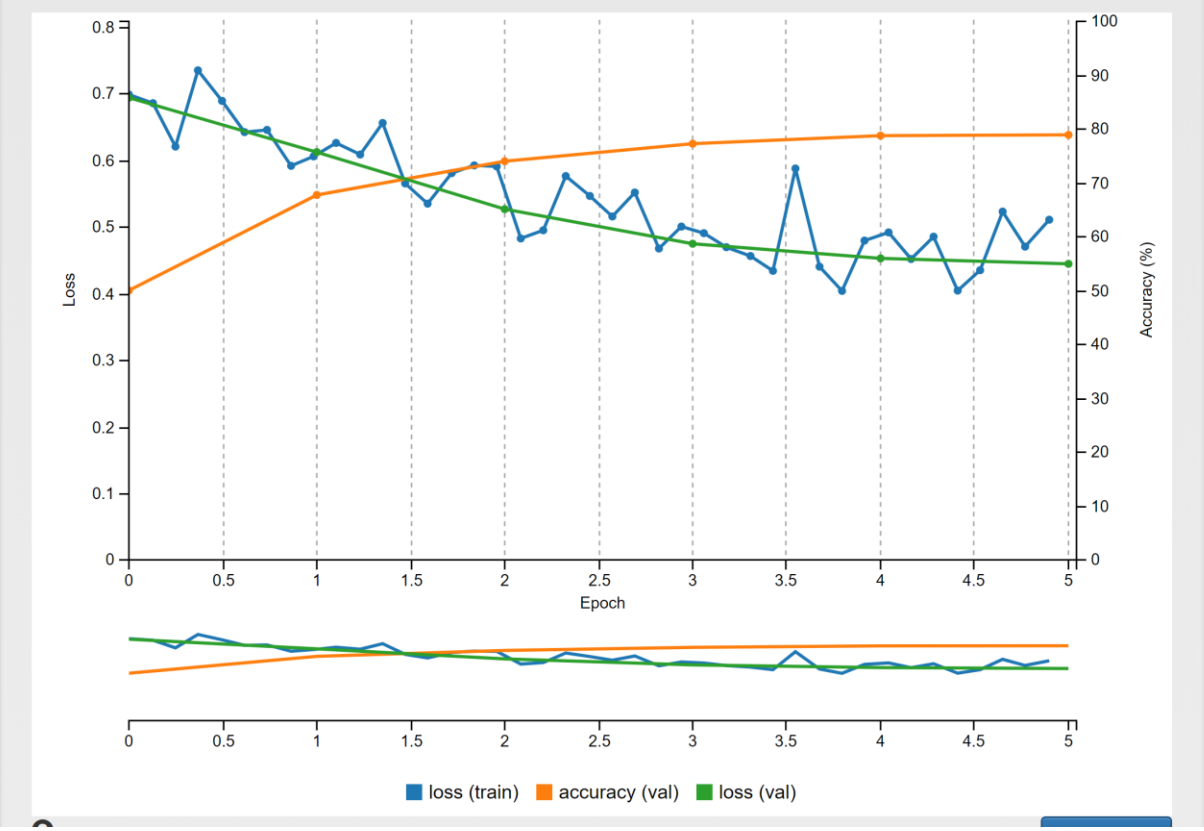
Process #96

CPU Utilization
140.9%
Memory
11.5 GB (19.3%)

Related jobs

Image Classification Dataset

Dogs and Cats Done



Accuracy is the highest in AlexNet model

GPU TASK 3 :

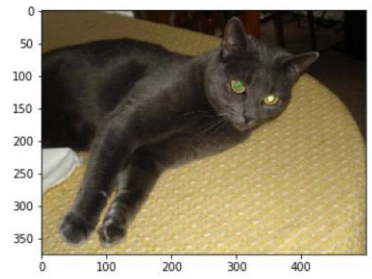
We just trained a neural network model to classify dogs from cats. In this task, we will d*eploy* that trained model into a simulator.

1. How to deploy a trained model into an application.
2. What role a trained model plays within an application.
3. To identify and use the pieces of a trained model

We also saw that a trained network consists of two components:

1. A description of the architecture of the untrained network
2. The weights that were "learned" while the network trained

In [5]:
```
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

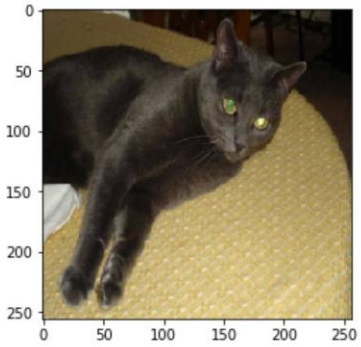
In [6]:
```
DATA_JOB_DIR = '/dli/data/digits/20180222-165843-ada0'  ## Remember to set this to be the job directory for your model
!ls $DATA_JOB_DIR
```

```
create_train_db.log  labels.txt        mean.jpg       train.txt  val.txt
create_val_db.log    mean.binaryproto  status.pickle  train_db   val_db
```

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*.

```
    print "Sorry cat:( https://media.giphy.com/media/jb8aFEQk3tADS/giphy.gif"
else:
    print "Welcome dog! https://www.flickr.com/photos/aidras/5379402670"
```

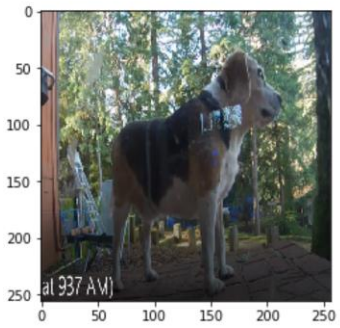
```
Input image:
```




```
Output:
Sorry cat:( https://media.giphy.com/media/jb8aFEQk3tADS/giphy.gif
```

```
    print "Sorry cat:( https://media.giphy.com/media/jb8aFEQk3tADS/giphy.gif"
else:
    print "Welcome dog! https://www.flickr.com/photos/aidras/5379402670"
```

```
Input Image:
```



```
[[ 0.39924237  0.6007576 ]]
Output:
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.

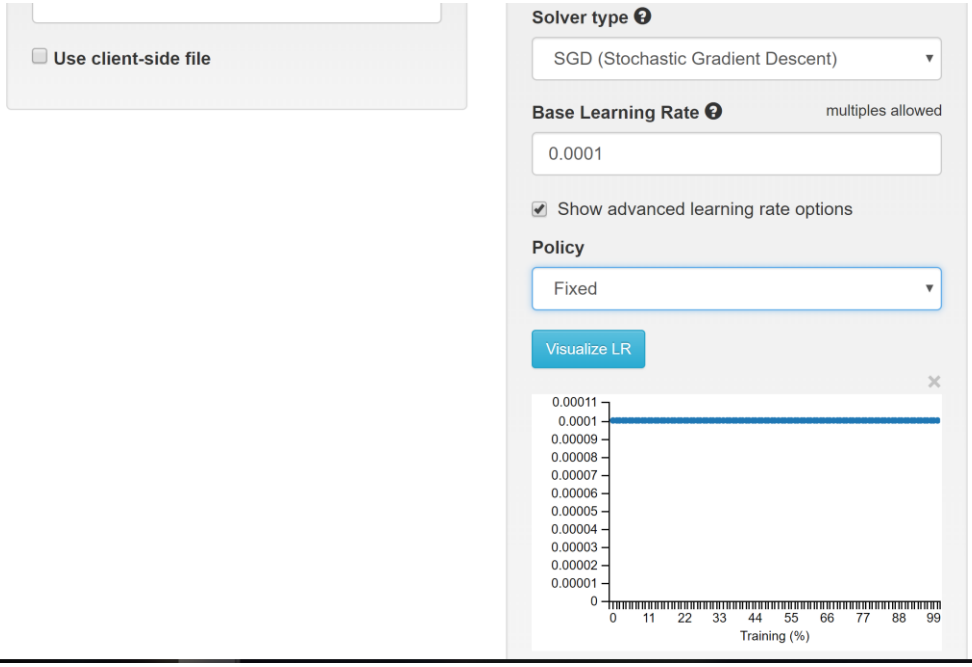
And then run our small python application with that image as input below. Ignore most of the output and scroll to the bottom. (Even errors and warnings are fine.)

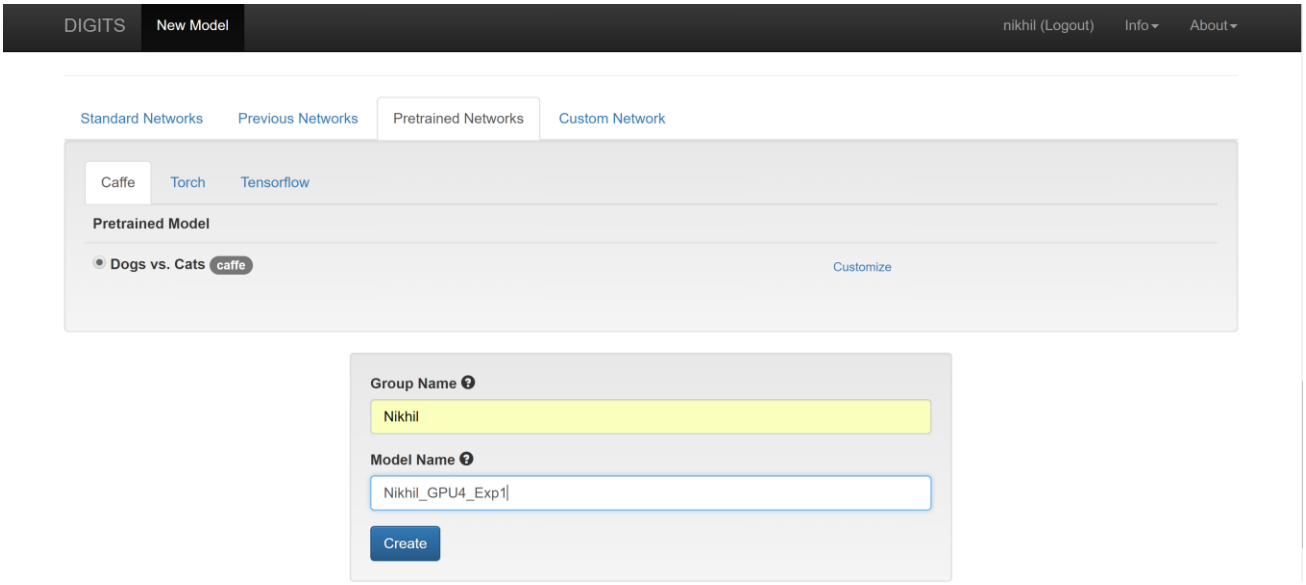
```
In [14]: !python pythondeployment.py $TEST_IMAGE 2>/dev/null
[[ 0.42844081  0.57155913]]
Output:
Welcome dog! https://www.flickr.com/photos/aidras/5379402670
None
```

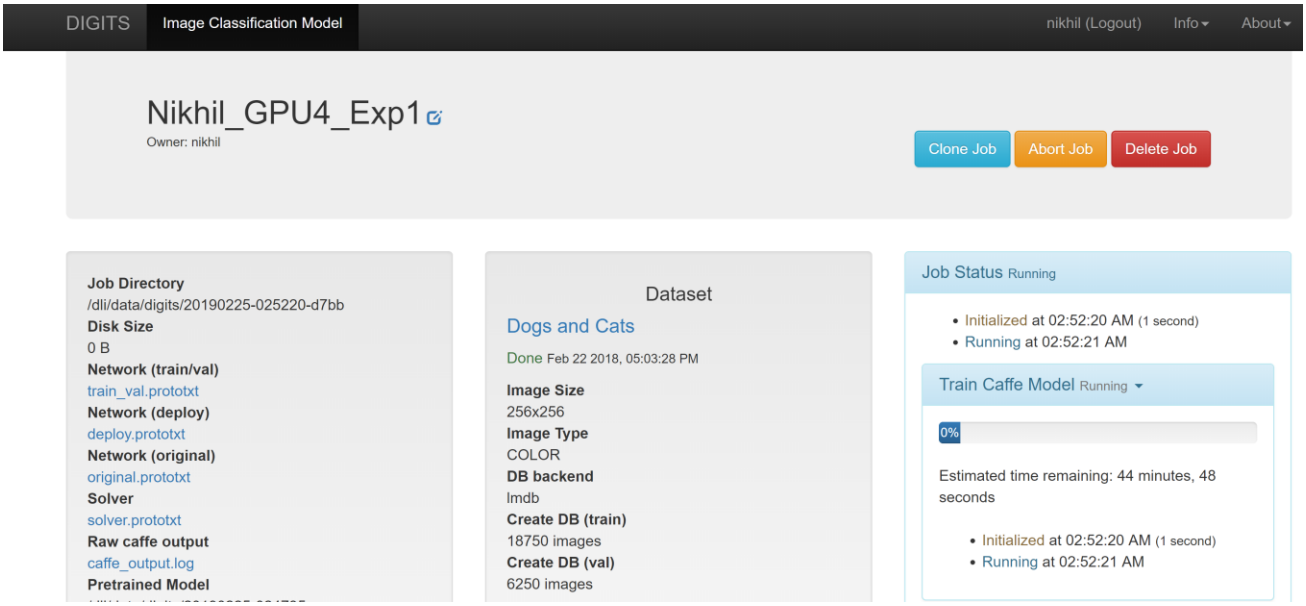
GPU TASK 4:

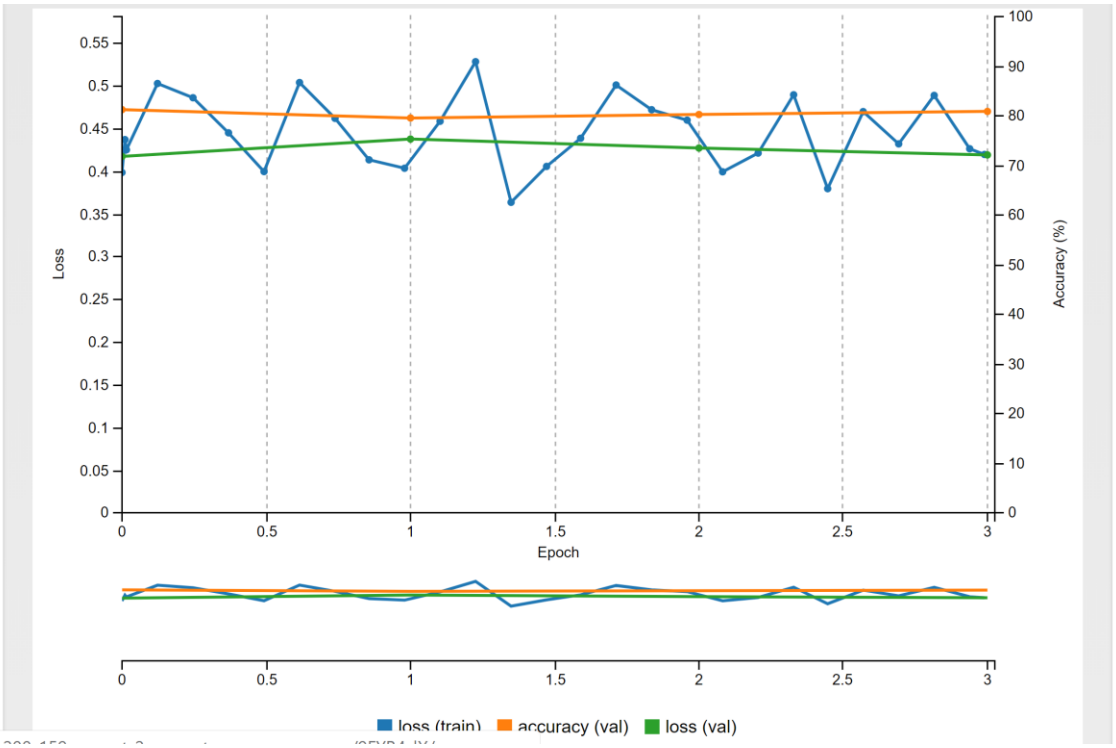
In this next section, we will engage with some strategies for improving training performance using our existing model. You will learn:

- To run more training epochs on an existing model, analogous to a human learner *studying more.*









1. As expected, the accuracy starts close to where our first model left off, 80%.
2. Accuracy DOES continue to increase, showing that increasing the number of epochs often does increase performance.
3. The rate of increase in accuracy slows down, showing that more trips through the same data can't be the only way to increase performance.

Deploying award winning models

Instead of working through the deployment of the model you just retrained, it's time to introduce a shortcut to performance: deploying expert pretrained models.

In this section, you'll learn to deploy other people's networks so that you can get the performance gains of their research, compute time, and data curation.

Let's use what we already know about deployment to run an image through this model. Start by initializing the model:

In [3]:
```
import caffe
import numpy as np
caffe.set_mode_gpu()
import matplotlib.pyplot as plt #matplotlib.pyplot allows us to visualize results

ARCHITECTURE = 'deploy.prototxt'
WEIGHTS = 'bvlc_alexnet.caffemodel'
MEAN_IMAGE = 'ilsvrc_2012_mean.npy'
TEST_IMAGE = '/dli/data/BeagleImages/louietest2.JPG'

# Initialize the Caffe model using the model trained in DIGITS
net = caffe.Classifier(ARCHITECTURE, WEIGHTS) #Each "channel" of our images are 256 x 256
```

Then create an input the network expects. Note that this is different than the *preprocessing* used in the last model. To learn how imagenet was preprocessed, the documentation was clearly presented on the Caffe website:

In [4]:
```
#Load the image
image= caffe.io.load_image(TEST_IMAGE)
```

Visualize the output –

In [5]:
```
# copy the image data into the memory allocated for the net
net.blobs['data'].data[...] = transformed_image

### perform classification
output = net.forward()

output
```

Out[5]:
```
{'prob': array([[  2.31780262e-09,   2.58294519e-09,   3.19525961e-09,
          2.09216755e-09,   5.00786212e-09,   2.04342987e-09,
          2.37229258e-09,   9.16246246e-11,   4.86508278e-10,
          5.01131137e-09,   1.35739935e-08,   9.87543114e-09,
          3.42600948e-11,   1.11983944e-09,   3.58725938e-10,
          7.21391349e-11,   1.98810324e-09,   3.15641522e-08,
          3.18406670e-08,   7.21174453e-10,   8.27692492e-09,
          1.42624259e-08,   2.83560397e-09,   3.29977219e-08,
          3.40230094e-10,   7.50220686e-09,   9.47929624e-10,
          1.18161825e-09,   2.00934576e-08,   2.79246071e-10,
          1.43229650e-09,   2.25081864e-09,   8.54826698e-09,
          1.04760878e-09,   3.35014100e-10,   1.14679100e-10,
          8.23971502e-10,   2.29677444e-10,   1.93692991e-08,
          3.21901672e-10,   2.40228504e-09,   1.76278014e-09,
          2.23937793e-08,   5.04234487e-10,   4.73921236e-10,
          5.41517942e-10,   1.59301594e-09,   2.94658253e-09,
```

Lets see which class this image belongs to by checking the maximum probability –

Work to make the output useful to a user.

What you see above is an array containing the probabilities that our image belongs to each of 1000 classes. Let's work to make this useful.

In [8]:
```
output_prob = output['prob'][0]  # the output probability vector for the first image in the batch
print 'predicted class is:', output_prob.argmax()
```

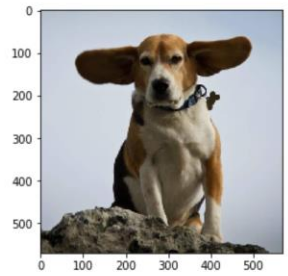
```
predicted class is: 162
```

This is closer. Take a look at imagenet's classes to see what that number corresponds to here. Better???? Let's add that functionality to our application so we have a useful end-to-end deployment.

```
In [10]: print ("Input image:")
         plt.imshow(image)
         plt.show()

         print("Output label:" + labels[output_prob.argmax()])
```

```
Input image:
```




```
Output label:n02088364 beagle
```

GPU TASK 5 : IMAGE DETECTION

The biggest takeaway here is that Deep Neural Networks turn one block of data (tensor) into another block of data (tensor). What that data represents is flexible. Our job is to find a pairing that helps us to solve the problem we've set out to solve, in this case, raw data to the (X,Y) pairings of Louie's boundaries within an image.

Workflow	Input	Output
Image Classification	Raw Pixel Values	A vector where each index corresponds with the likelihood or the image of belonging to each class
Object Detection	Raw Pixel Values	A vector with (X,Y) pairings for the top-left and bottom-right corner of each object present in the image
Image Segmentation	Raw Pixel Values	A overlay of the image for each class being segmented, where each value is the likelihood of that pixel belonging to each class
Text Generation	A unique vector for each 'token' (word, letter, etc.)	A vector representing the most likely next 'token'
Image Rendering	Raw Pixel Values of a grainy Image	Raw pixel values of a clean image

We're going to use what's called the "sliding window" approach, where we'll take split out image into small sections which we'll call grid squares. We're run

each grid square through an image classifier. If that grid square contains an image of a dog, we'll have localized Louie in the image.

From there you will work to use deep learning for more and more of your workflow, learning to adjust neural network architecture and finally, a completely new object detection specific workflow.

Next, we'll take an image directly from a front door security camera. Note, this image is larger than 256X256. We want to know *where* a dog is in this image.

In [2]:
```
# Choose a random image to test against
#RANDOM_IMAGE = str(np.random.randint(10))
IMAGE_FILE = '/dli/tasks/task5/task/images/LouieReady.png'
input_image= caffe.io.load_image(IMAGE_FILE)
plt.imshow(input_image)
plt.show()
```

Using our *learned* function: Forward Propagation

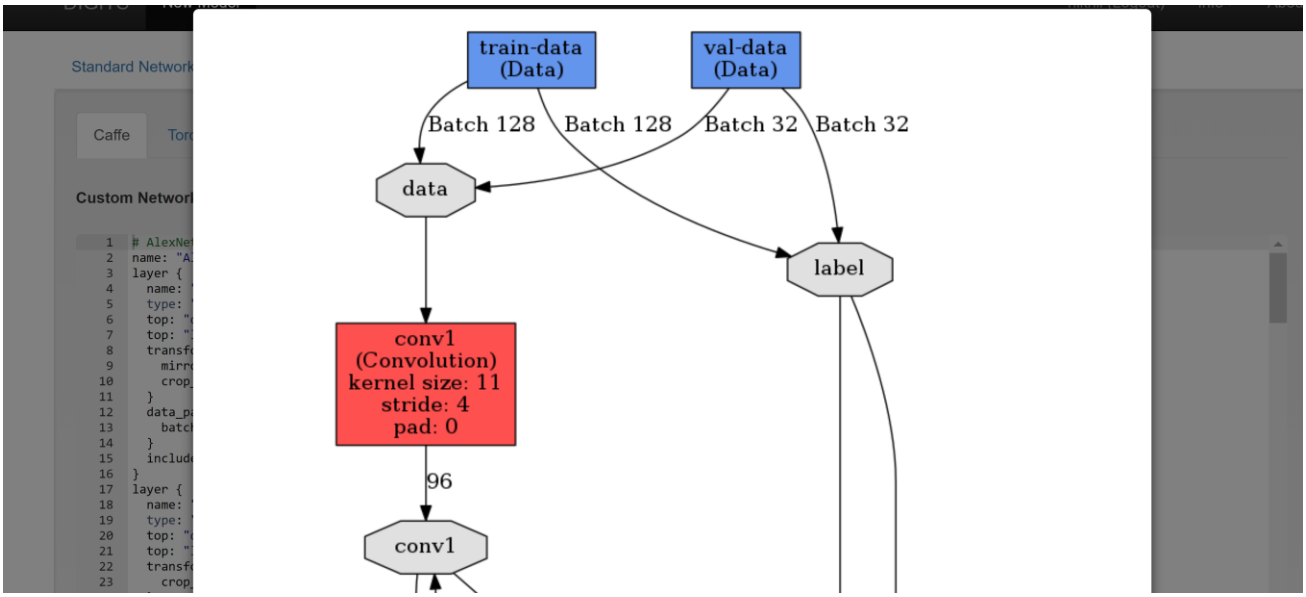
```
# Display total time to perform inference
print 'Total inference time: ' + str(end-start) + ' seconds'
```


```
Total inference time: 0.801002979279 seconds
```

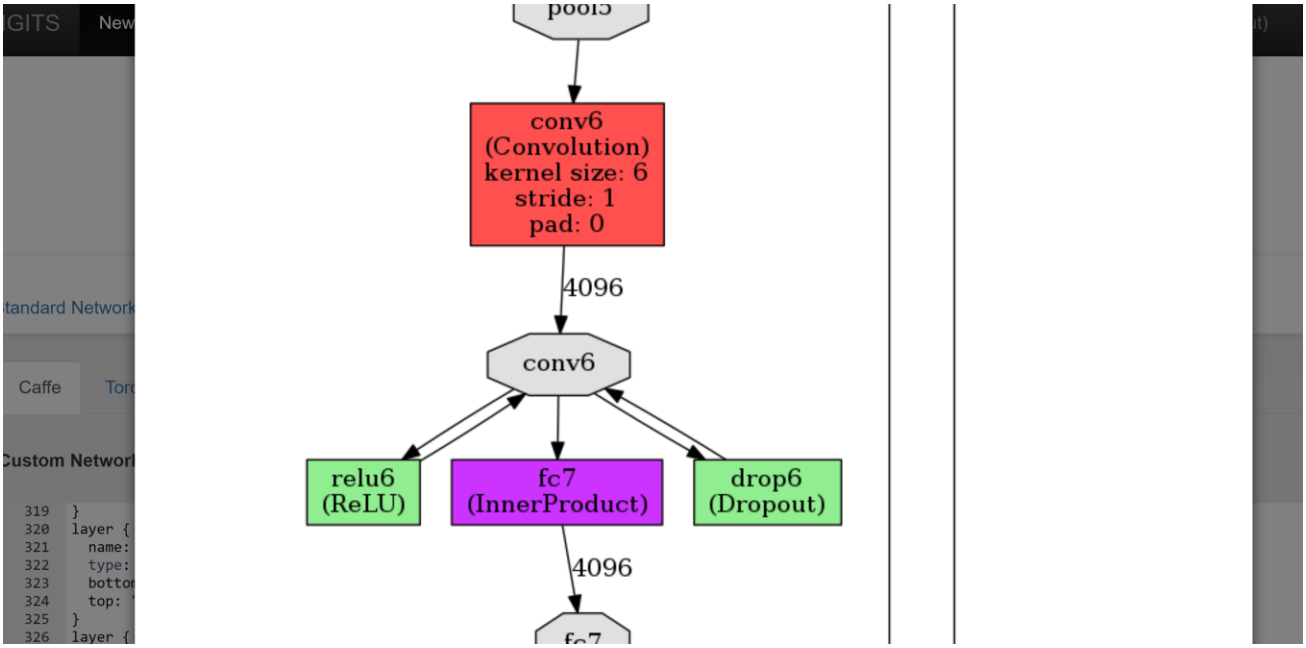
Again, this is the output of a randomly chosen wide area test image along with an array showing the predicted class for ea

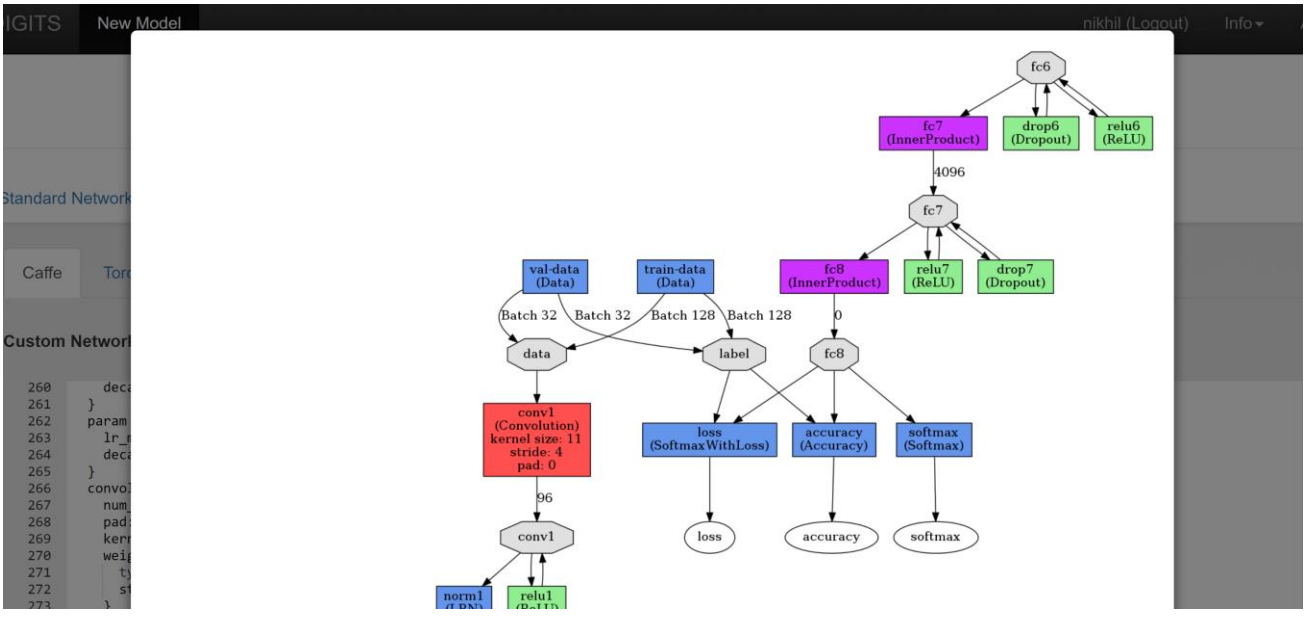
As we have now seen, the advantage of this sliding window approach is that we can train a detector using only patch-based training data (which is more widely available) and that with our current skillset, we can start working to solve the problem.

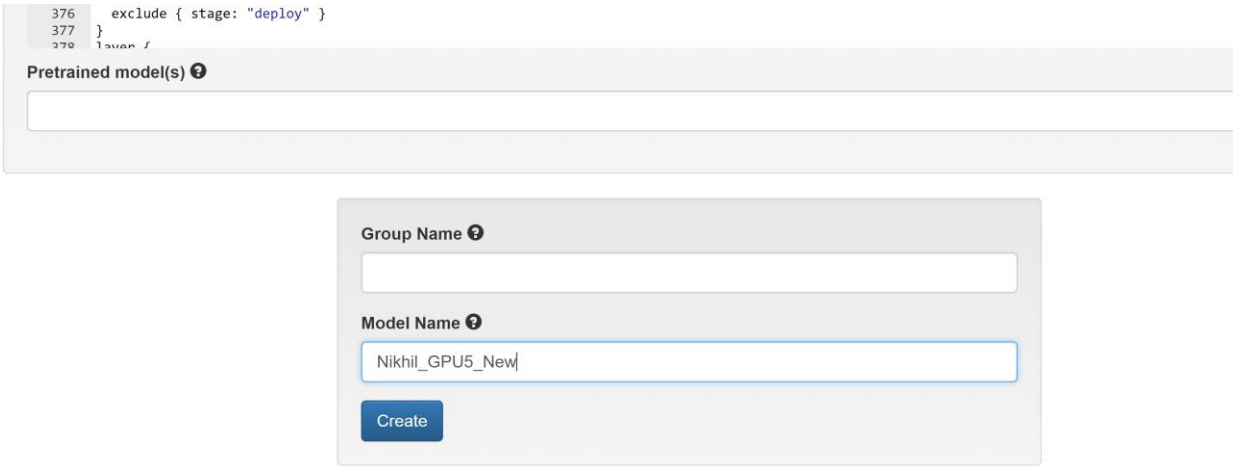
Approach 2 – rebuilding from an existing Deep learning network



We have changed the layer fc6 to conv6 now -

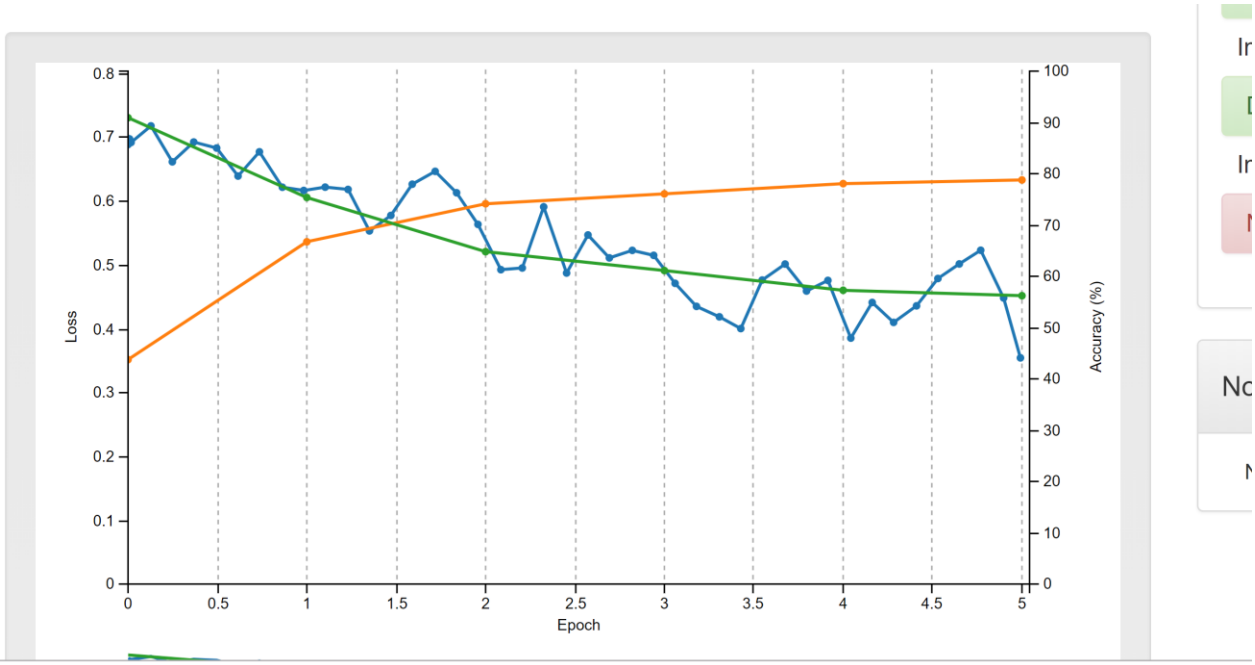

pool5
conv6
(Convolution)
kernel size: 6
stride: 1
pad: 0
4096
conv6
relu6
(ReLU)
fc7
(InnerProduct)
drop6
(Dropout)
4096

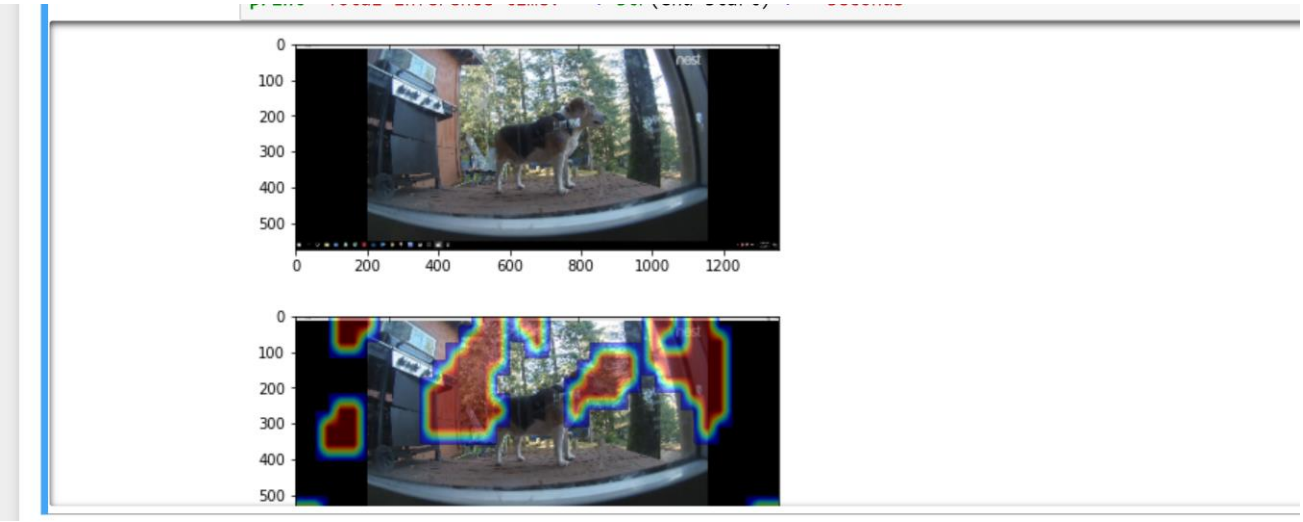

fc6
fc7
(InnerProduct)
drop6
(Dropout)
relu6
(ReLU)
4096
fc7
val-data
(Data)
train-data
(Data)
fc8
(InnerProduct)
relu7
(ReLU)
drop7
(Dropout)
Batch 32
Batch 32
Batch 128
Batch 128
0
data
label
fc8
conv1
(Convolution)
kernel size: 11
stride: 4
pad: 0
loss
(SoftmaxWithLoss)
accuracy
(Accuracy)
softmax
(Softmax)
96
conv1
loss
accuracy
softmax
norm1
relu1



Create a new model now with the customized AlexNet –

/dli/data/digits/20190225-041311-3325





Approach 3: DetectNet

We started this lab with AlexNet, an elegant solution to the very specific problem of *image classification*. We built some python around it and then performed minor brain surgery to accomplish a task that we wouldn't have been able to prior to our work in deep learning, *object detection.*

However, is there an elegant solution to *object detection?* Could we build a model that maps directly from the input you have: photos of various sizes, to the output you want: localization and detection information?

The fact that there is will help us to broaden our definition of Deep Learning and provide some insight into what *other* problems you can solve with it.

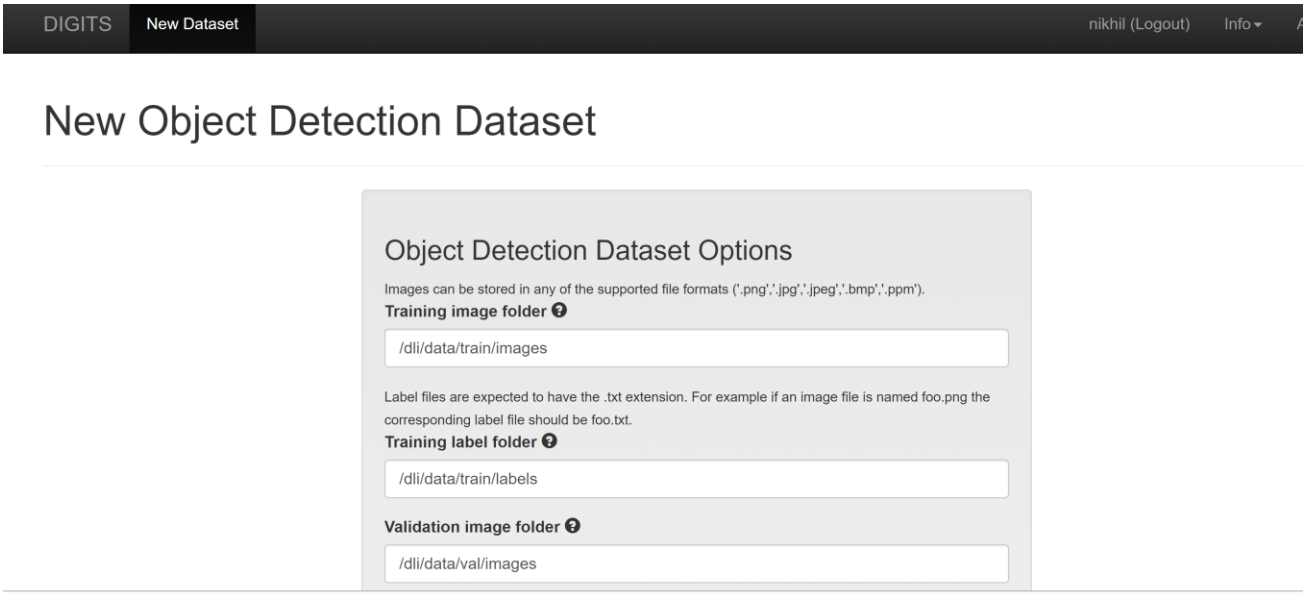
In [12]:
```
input_image = caffe.io.load_image('/dli/data/train/images/488156.jpg') #
plt.imshow(input_image)
plt.show()
```

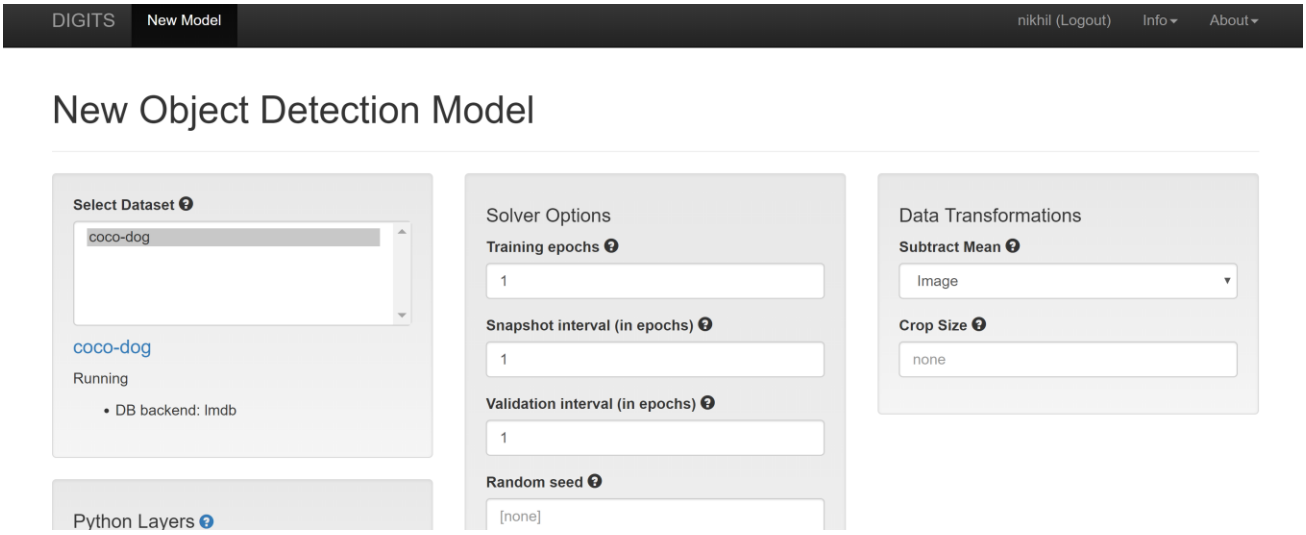
Next, its corresponding label:

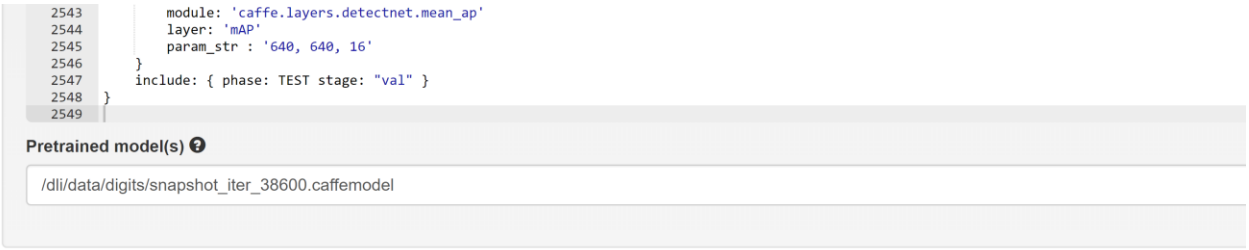
In [13]:
```
!cat '/dli/data/train/labels/488156.txt' #"cat" has nothing to do with the animals, this displays the text
```

```
dog 0 0 0 161.29 129.03 291.61 428.39 0 0 0 0 0 0 0
surfboard 0 0 0 31.25 36.44 410.41 640.0 0 0 0 0 0 0 0
```

Note:




DIGITS
New Model
nikhil (Logout)
Info
About
New Object Detection Model
Select Dataset
coco-dog
coco-dog
Running
• DB backend: lmdb
Python Layers
Solver Options
Training epochs
1
Snapshot interval (in epochs)
1
Validation interval (in epochs)
1
Random seed
[none]
Data Transformations
Subtract Mean
Image
Crop Size
none


2543 module: 'caffe.layers.detectnet.mean_ap'
2544 layer: 'mAP'
2545 param_str : '640, 640, 16'
2546 }
2547 include: { phase: TEST stage: "val" }
2548 }
2549
Pretrained model(s)
/dli/data/digits/snapshot_iter_38600.caffemodel


Group Name
Nikhil
Model Name
Nikhil_GPU5_3
Create

Job Directory
/dli/data/digits/20190225-044319-6349
Disk Size
0 B
Network (train/val)
train_val.prototxt
Network (deploy)
deploy.prototxt
Network (original)
original.prototxt
Solver
solver.prototxt
Raw caffe output
caffe_output.log
Pretrained Model
/dli/data/digits/snapshot_iter_38600.caffemodel
Visualizations
Tensorboard

Dataset

coco-dog

Done 04:40:16 AM

- DB backend: lmdb
- Create train_db DB
  - **Entry Count:** 3855
  - **Feature shape** (3, 640, 640)
  - **Label shape** (1, 52, 16)
- Create val_db DB
  - **Entry Count:** 1969
  - **Feature shape** (3, 640, 640)
  - **Label shape** (1, 51, 16)

Job Status Running

- Initialized at 04:43:19 AM (1 second)
- Running at 04:43:20 AM

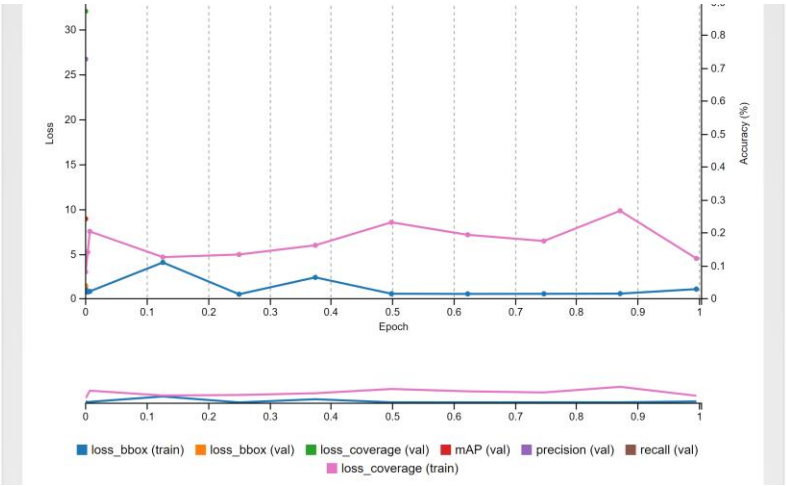
Train Caffe Model Running

0%

Estimated time remaining: 36 minutes, 49 seconds

- Initialized at 04:43:19 AM (1 second)
- Running at 04:43:20 AM

Hardware



GPU Utilization
95%
Temperature
67 °C

Process #272

CPU Utilization
142.8%
Memory
3.82 GB (6.4%)

Related jobs

Generic Dataset

coco-dog Done

Generic Image Model

Nikhil_GPU5_3 Error

Bookmark this page

Congrats. You've just completed your first course in deep learning. At this point, you have learned to:

1. Identify the ingredients required for deep learning
2. Train a Deep Neural Network to correctly classify images it has never seen before
3. Deploy Deep Neural Networks into applications
4. Identify techniques for improving the performance of deep learning applications
5. Assess the types of problems that are candidates for deep learning
6. Modify neural networks to change behavior

Go to the next section to attempt a small deep learning problem on your own and earn a certificate showing your mastery of t course.