Assignment4

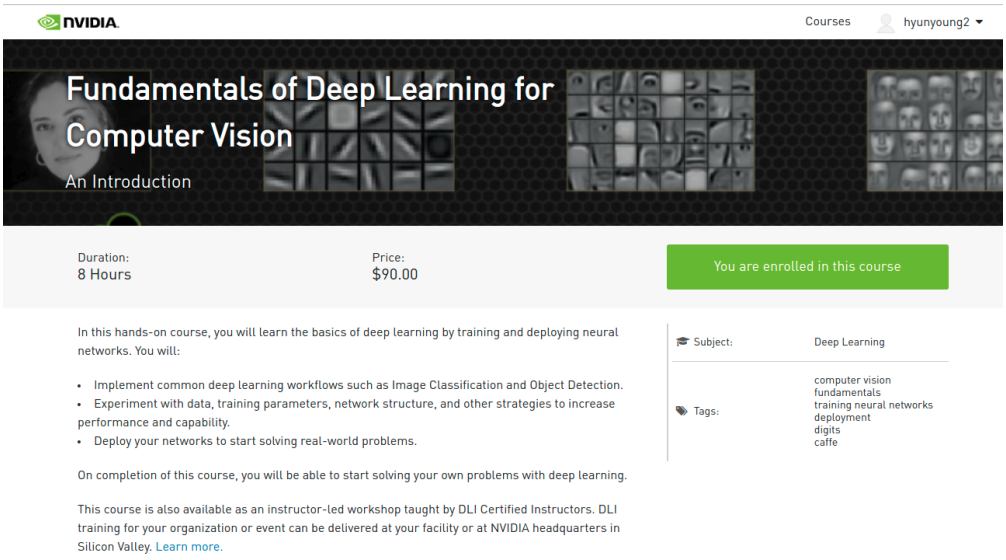
June 15, 2019

Fundamental of Deep Learning for Computer vision by NVIDIA with DIGIST of NVIDIA

This is for me to rearrange the lecture above provided by nvidia to raise my skill about deep learning,

This is summarized by me.

The following is a image which is I am getting into lecture.



Deeplearning is inspired by an undestanding of humman learning.

What is the difference between the classic matchine learning and Deep Learning?

In the classic Machine learnig, Input is change to hand-designed features

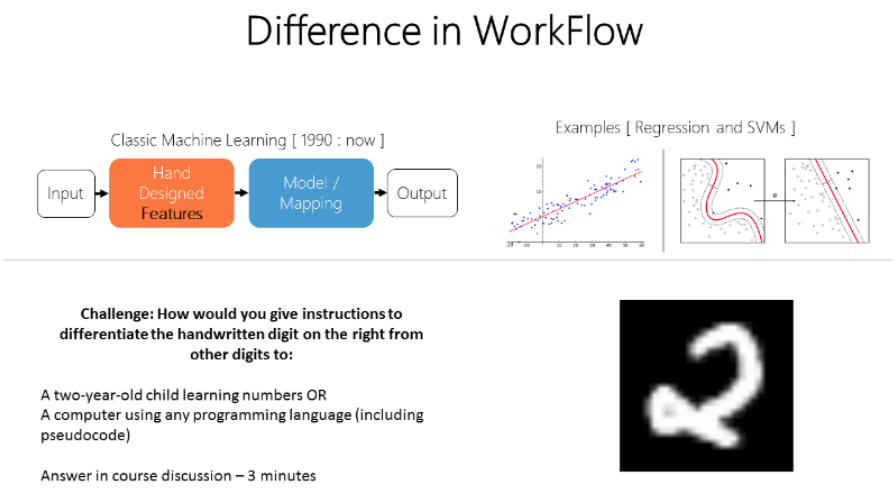
But, In the other case, it is ent-to-end learning which means you don't need to design the feature by hand from input.

1 Biological inspiration

So Deeplearning cares about only input and output. the middle neuron is learned by a lot of emxamples.

Computers and computing help us achieve more complex goals than we could do alone.

However, conventially, computers could only follow the specific instructions they were given.



When you solve problems with programming, you need to logically write instruction step-by-step for a compute to solve it.

Deep learning allows computers to "learn" from examples(data).

Solving problems with deep learning requires identifying some pattern in the world, finding exmaples that higlight bot sides of the pattern(the input and the output) and then letting a "neural network" learn the map between the two.

So Deep learning learn the pattern from the examples by removing the need to write explicit instructions for your own problem.

2 Tribe

There are a lot of machine learning tribes as follows.

3 Deep Neural Networks: GPU Task 1

The task is called image classification where the image as input of model is "Louie or not Louie"

after training, you can check your model's output like this:

Let's improve the model by training a few more times.

Let's see the loss and learning rate in a few more times than the prior, 2 epoch.

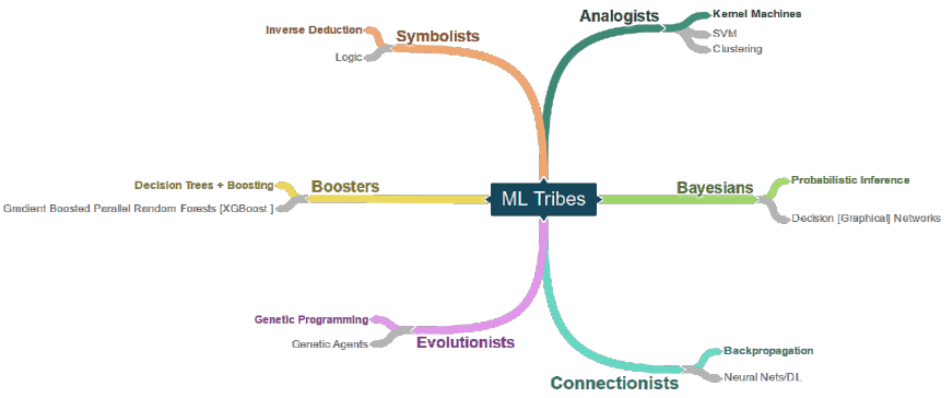
the classified result with epoch 100

4 Big Data: GPU Task 2

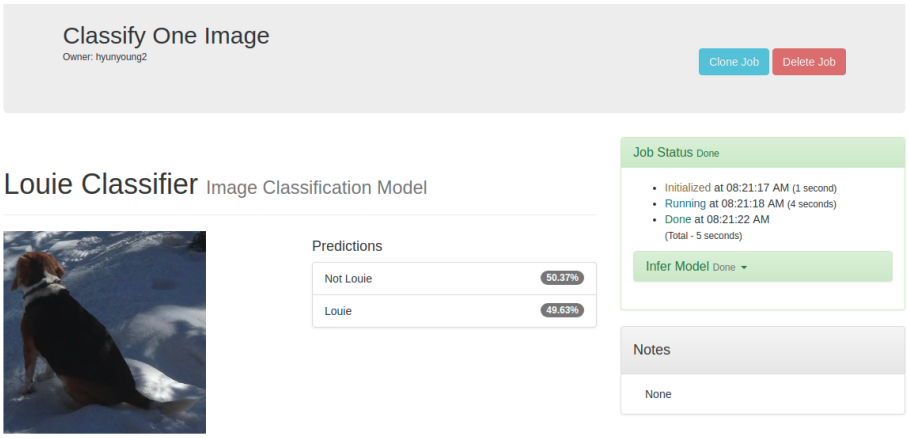
under this task, you will combine three ingredients below to train neural network

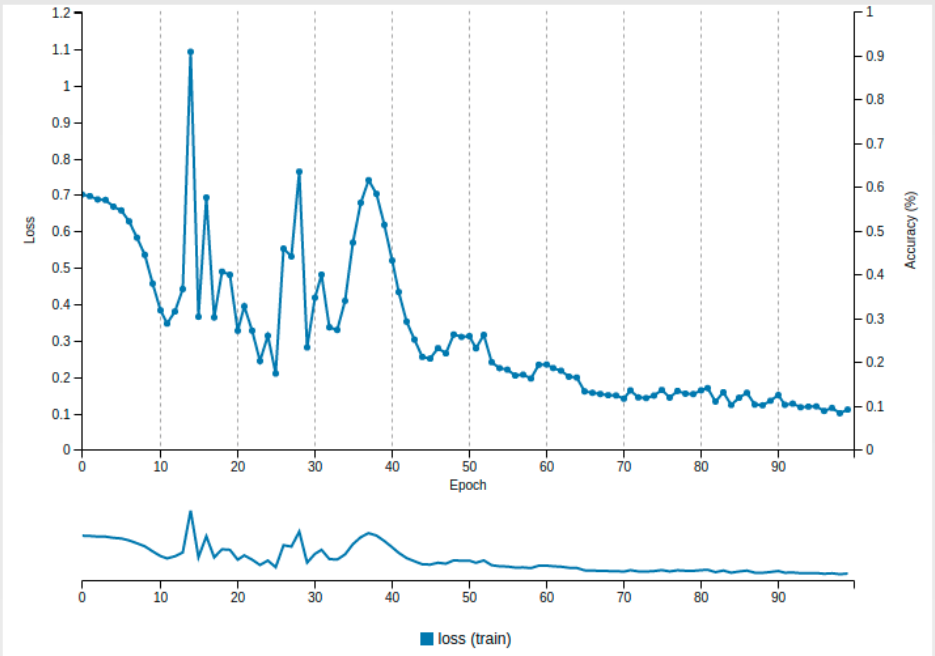
1. Deep Neural network
2. The GPU
3. Big data

ML Tribes

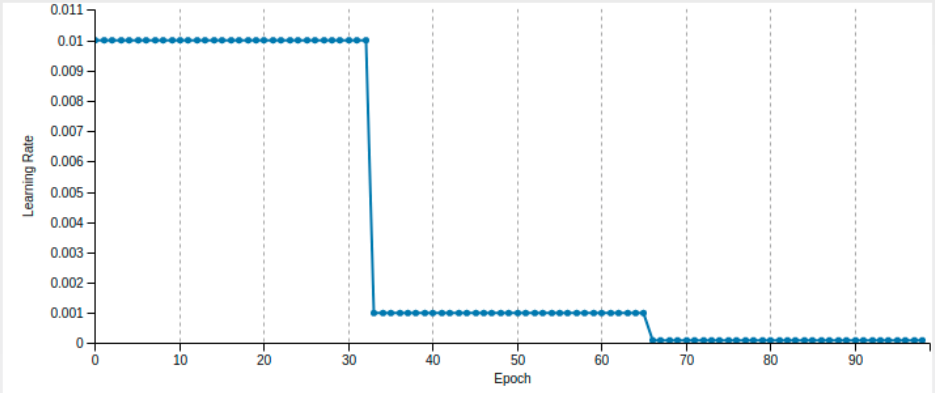


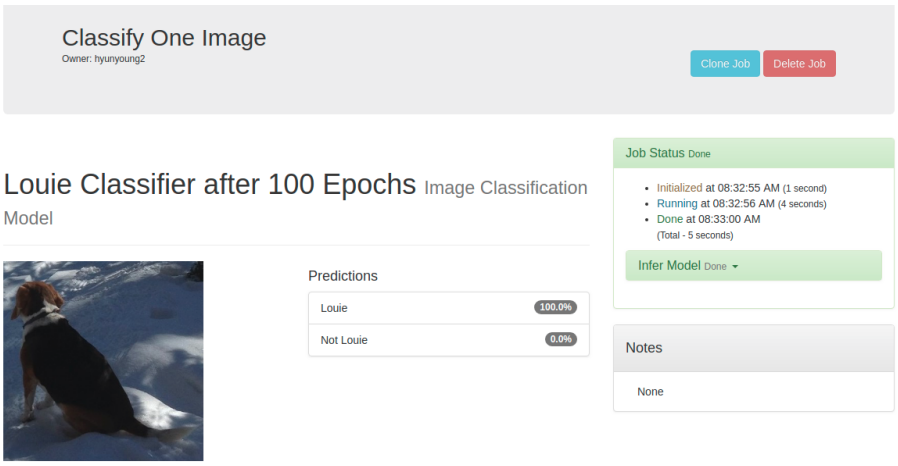
Tribe	Origins	Master Algorithm
Symbolists	Logic, philosophy	Inverse deduction
Connectionists	Neuroscience	Backpropagation
Evolutionaries	Evolutionary biology	Genetic programming
Bayesians	Statistics	Probabilistic inference
Analogizers	Psychology	Kernel machines





View Large





Before entering the task, we need the large dataset. Fortunately, kaggle has has a dataset that we can start with consisting of 18750 labeled images of dogs and cats.

From now on, we teach our model what a dog is other than who Louie is.

Let's see the resulting dataset in DIGITS

The following is the dataset split into two sets for training and validation.

The blue bar on the right side of histogram is dog images, and the other part is cat images.

If you want to know the actual data, select the "Explore the db".

You can train your model from train dataset, and then you infer the output form unseen data to classify the input making desicion based on what was learned called inference.

Les's see the process of training our model which is "Alexnet"

I will test my model with the unseen data, "louiestest2.JPG". the following is the inference of my model.

after seeing the result, you can interpret it in two ways.

```
1) It worked! We took an untrained neural network, exposed it to thousands of *labeled* images, and
```

```
2) We're not there yet. Human learners would be 100% confident that that image contained a dog. Our r
```

5 How to deploy the model

The deep learning workflow has two distinct sections: training and deployment.

Deployment is the work of taking a trained model and putting it ot work as a part of an application. You can deploy to edge devices such as robots or autonomous vehicels. you can alos deploy to a server in order to play a role in any piece of software.

Typcially the deep learning model is complex, however if you know the input and output on deep learning model.

You could deal with the model as a function to generate output regarding input.

To successfully deploy a trained model, we have two jobs.

```
1) Our first job is to provide our model an input that it expects.
```

Create DB (train)

Input File (before shuffling)
train.txt
DB Creation log file
create_train_db.log

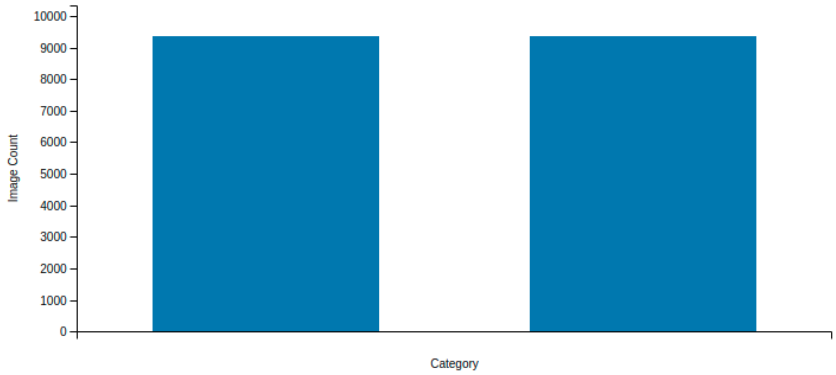
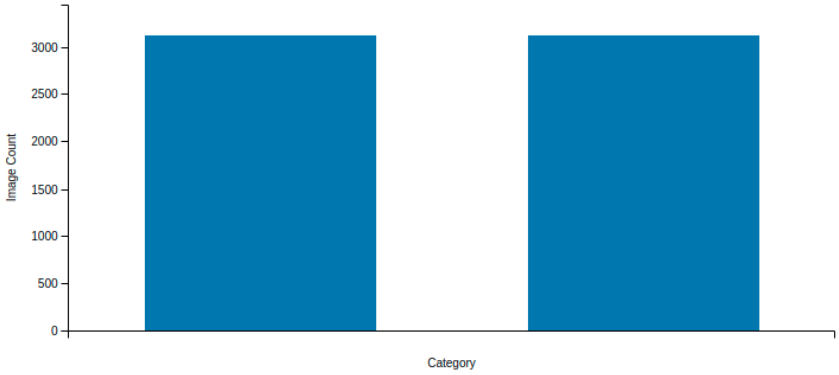


Image Mean:

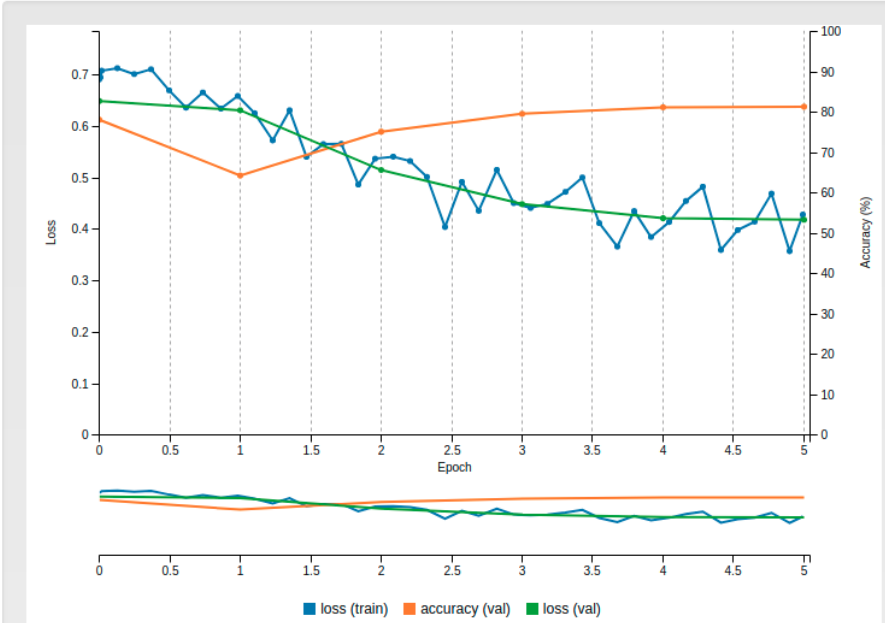
Explore the db

Create DB (val)

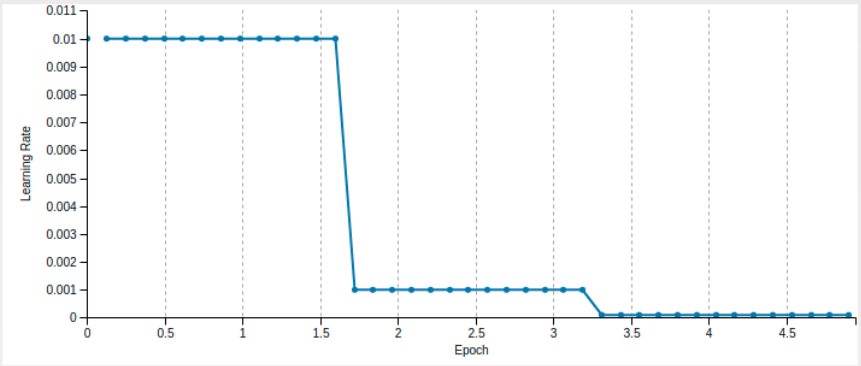
Input File (before shuffling)
val.txt
DB Creation log file
create_val_db.log



Explore the db



View Large



Classify One Image

Owner: hyunyoung2

Clone Job | Delete Job

Dogs and Cats Classifier Image Classification Model

Predictions

dogs	74.38%
cats	25.62%

Job Status Done

- Initialized at 09:14:31 AM (1 second)
- Running at 09:14:32 AM (4 seconds)
- Done at 09:14:36 AM
  (Total - 5 seconds)

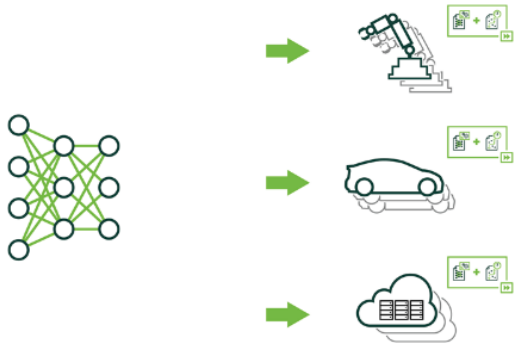
Infer Model Done

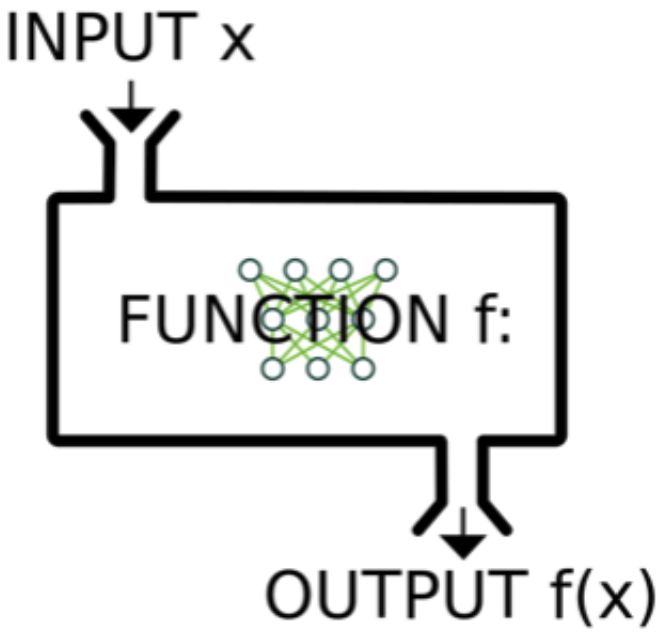
Notes

None

Deployment

How do I use a trained neural network as part of a solution?



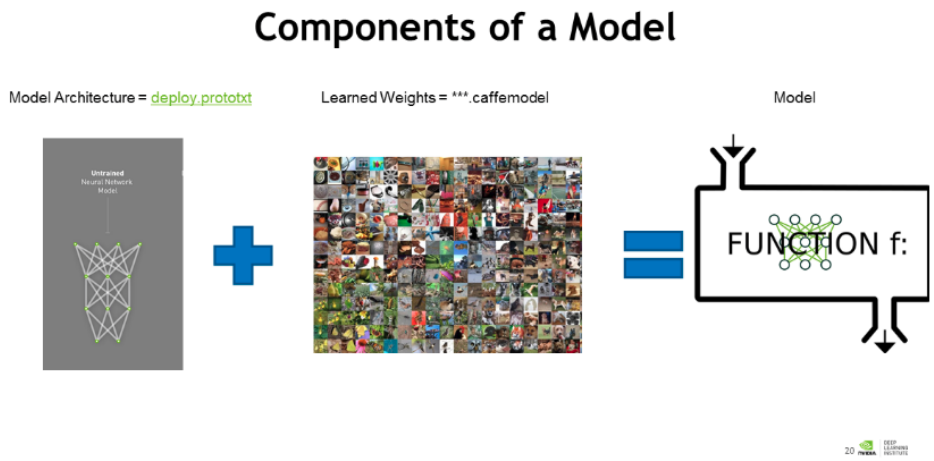


2) Our second job is to provide our end user an output that is useful.

the input that our model expects is decided by model architecture and how it was trained.

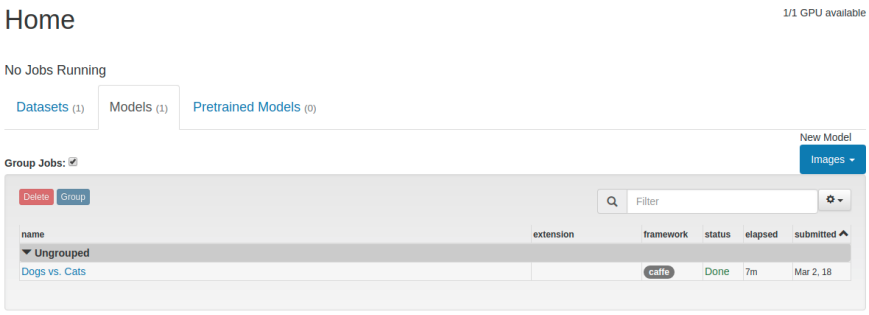
So you have to write code in the front of the model to convert the input you have to the input the model expects.

Also the output is decided by model architecture and what it is learned. So it is the same from dealing with input, the output is converted to the output end user expect from the output generated from model.



6 Deploying our Model: GPU Task 3

selct the model below



after entering the model, type in instructions below

```
MODEL_JOB_DIR = '##FIXME##'  ## Remember to set this to be the job directory for your model
!ls $MODEL_JOB_DIR
```

Since I use the caffe for the model, my model consists of two files: the architecture and the weights.

The architecture : model_name.prototxt The weight : snapshot_iter_#.caffemodel.

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

let's make net which classify object as follows.

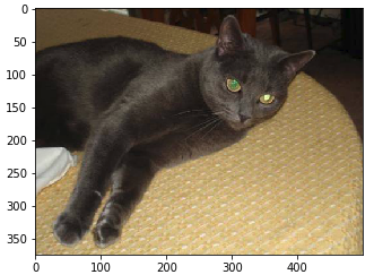
```
import caffe
caffe.set_mode_gpu()
# Initialize the Caffe model using the model trained in DIGITS
net = caffe.Classifier(ARCHITECTURE, WEIGHTS,
                channel_swap =(2, 1, 0), #Color images have three channels, Red, Green, and Blue.
                     raw_scale=255) #Each pixel value is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

7 Creating an Expected Input: Preprocessing

Let's see the preprocessing for my model

The following is no-preprocessing.

```
In [5]: import matplotlib.pyplot as plt #matplotlib.pyplot allows us to visualize results
        input_image= caffe.io.load_image('##FIXME##')
        plt.imshow(input_image)
        plt.show()
```



The follwoing is done with preprocessing for Alexnet.

The following is another preprocessing called "normalize"

so far, input is dealt with as input my model expects.

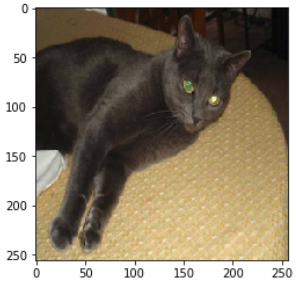
Finally, if you want vector probability, use the function below

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

8 Generating a useful output: Postprocessing

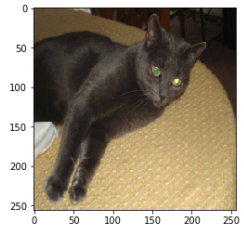
Let's test other example for the doggy door.

In [7]:
```
import cv2
input_image=cv2.resize(input_image, (256, 256), 0,0)
plt.imshow(input_image)
plt.show()
```



In [16]:
```
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"
```

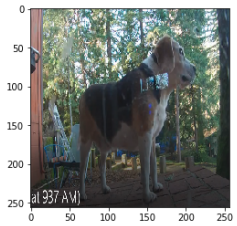
```
Input image:
```




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

In [17]:
```
##Create an input our network expects
input_image= caffe.io.load_image('##FIXME##')
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"
```

```
Input Image:
```



```
[[ 0.39924237  0.6007576 ]]
Output:
Welcome dog! https://www.flickr.com/photos/aidras/5379402670
```

9 Putting it all together

10 Performance during Training: GPU Task 4

let's work through a full deep learning workflow:

1) we prepare a dataset for training
2) we select a network to train
3) we train the network
4) we test the trained model in a training environment
5) we deploy the trained model into an application

But, You don't need to have a trained model from scratch to deploy it.

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.
- To search the hyperparameter space, analogous to a human learner responding differently to a different teaching style.
- To use the results of others' research, compute, network design, and data, analogous to a human learner copying off an expert

explore the dataset we will be using in this section, imagenet.

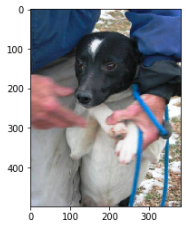
11 Learning rate

You may be wondering three things:

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, 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 [20]:
```
TEST_IMAGE = '##FIXME##'
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 [21]:
```
!python pythondeployment.py $TEST_IMAGE 2>/dev/null
```

```
[[ 0.42844081  0.57155913]]
Output:
Welcome dog! https://www.flickr.com/photos/aidras/5379402670
None
```

```
1) What is "learning rate?"
2) Why does the learning rate decrease throughout the training session?
3) Who controls that?

1) Learning rate is the rate at which each "weight" changes during training. Each weight is moving i
2) The learning rate decreases throughout the training session because the network is getting close
3) You control the learning rate. The learning rate is one of many "hyperparameters" that we set whe

Since at the end of the session the learning rate was slow, when starting from a pretrained network,
```

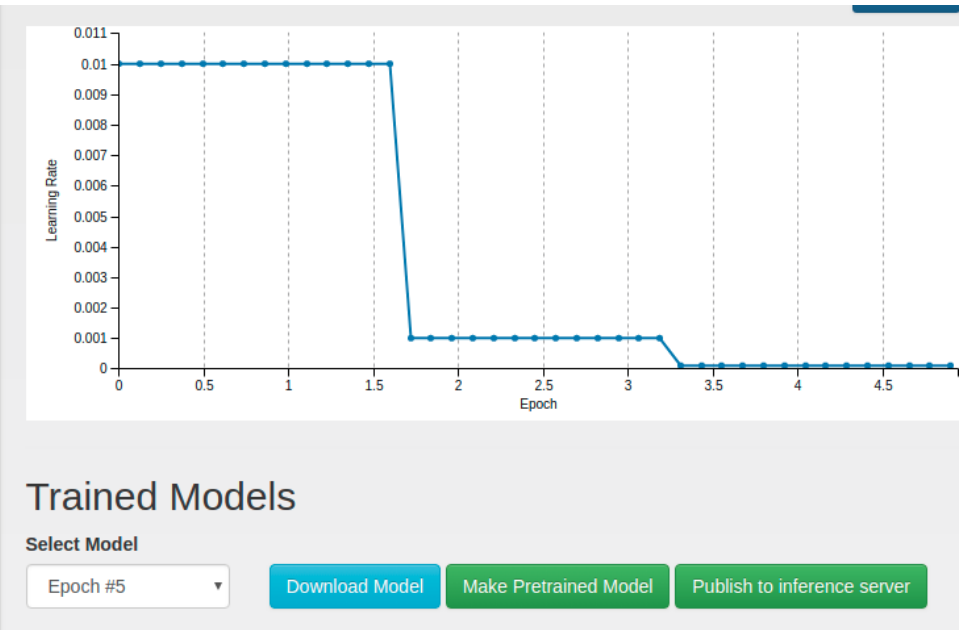
Let's see pretrained model.

```
Note the following:

As expected, the accuracy starts close to where our first model left off, 80%.
Accuracy DOES continue to increase, showing that increasing the number of epochs often does increas
The rate of increase in accuracy slows down, showing that more trips through the same data can't be
There are four categories of levers that you can manipulate to improve performance. Time spent learn

1) Data - A large and diverse enough dataset to represent the environment where our model should wor
2) Hyperparameters - Making changes to options like learning rate are like changing your training "
3) Training time - More epochs improve performance to a point. At some point, too much training will
4) Network architecture - We'll begin to experiment with network architecture in the next section.
```

Let's use pretrained AlexNet, so type in the following commands

```
!wget http://dl.caffe.berkeleyvision.org/bvlc_alexnet.caffemodel
!wget https://raw.githubusercontent.com/BVLC/caffe/master/models/bvlc_alexnet/deploy.prototxt
```

Let's download the mean image from DIGITS



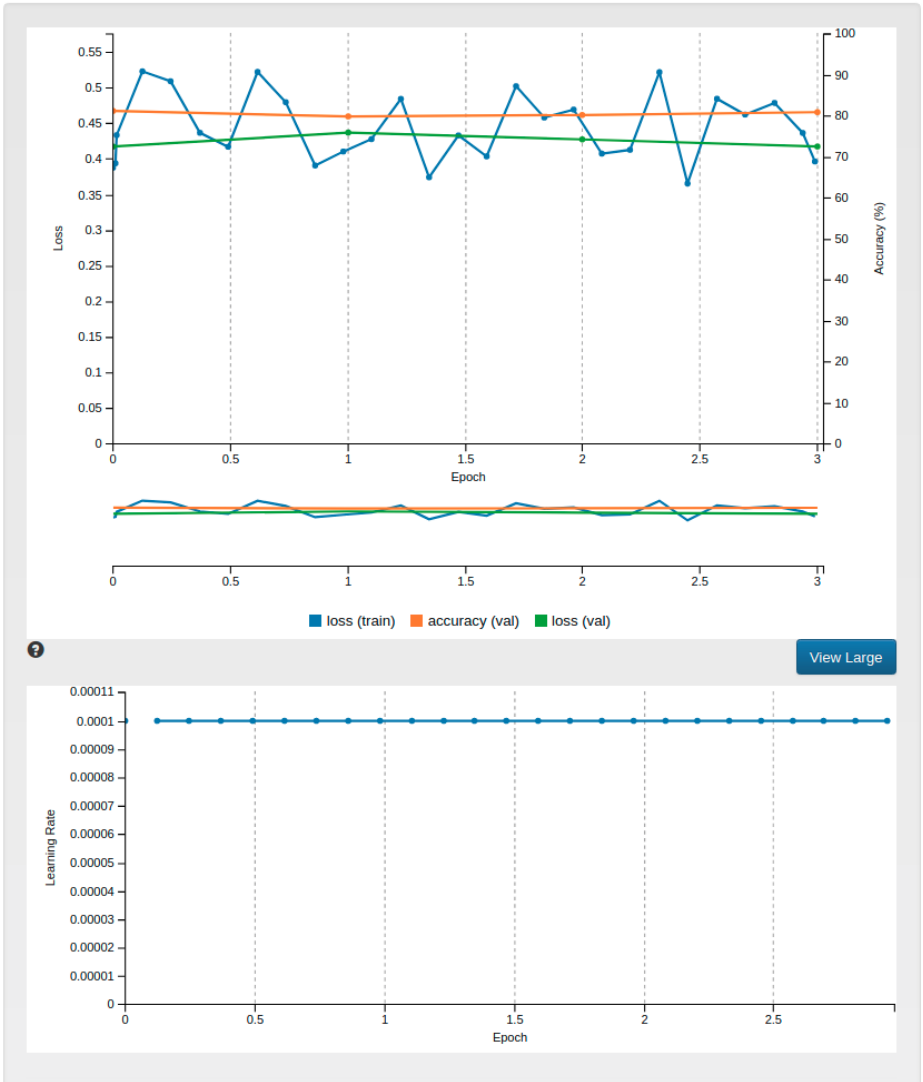

```
!wget https://github.com/BVLC/caffe/blob/master/python/caffe/imagenet/ilsvrc_2012_mean.npy?raw=
!mv ilsvrc_2012_mean.npy?raw=true ilsvrc_2012_mean.npy
```

Let's initialize the model with reference for preprocessing of ImageNet

```
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 = '/filepath/louietest2.JPG'

# Initialize the Caffe model using the model trained in DIGITS
net = caffe.Classifier(ARCHITECTURE, WEIGHTS) #Each "channe
```

Let's make input that the model expects

```
#Load the image
image= caffe.io.load_image(TEST_IMAGE)
plt.imshow(image)
plt.show()

#Load the mean image
mean_image = np.load(MEAN_IMAGE)
mu = mean_image.mean(1).mean(1)  # average over pixels to obtain the mean (BGR) pixel values
```


Loss
Accuracy (%)
Epoch
loss (train)
accuracy (val)
loss (val)
View Large
Learning Rate
Epoch

```
# create transformer for the input called 'data'
transformer = caffe.io.Transformer({'data': net.blobs['data'].data.shape})
transformer.set_transpose('data', (2,0,1)) # move image channels to outermost dimension
transformer.set_mean('data', mu)        # subtract the dataset-mean value in each channel
transformer.set_raw_scale('data', 255)      # rescale from [0, 1] to [0, 255]
transformer.set_channel_swap('data', (2,1,0))  # swap channels from RGB to BGR
# set the size of the input (we can skip this if we're happy with the default; we can also change it l
net.blobs['data'].reshape(1,        # batch size
                          3,         # 3-channel (BGR) images
                          227, 227)  # image size is 227x227

transformed_image = transformer.preprocess('data', image)
```

let's run the function,

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

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

#output
```

let's make output that end user expects

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

If you want to take a look at imagenet's classes to see what that number corresponds to the result,

visit here.

Again using wget to get a dictionary (dict) of class to label.

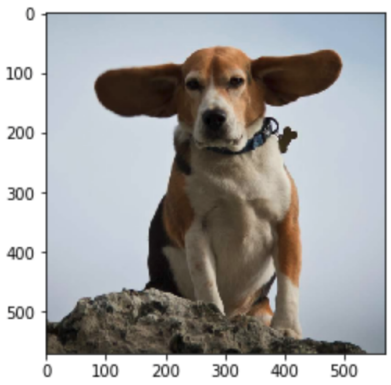
```
!wget https://raw.githubusercontent.com/HoldenCaulfieldRye/caffe/master/data/ilsvrc12/synset_wo
labels_file = 'synset_words.txt'
labels = np.loadtxt(labels_file, str, delimiter='\t')

print 'output label:', labels[output_prob.argmax()]
```

The result of application

```
print ("Input image:")
plt.imshow(image)
plt.show()

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

In [8]:
```
print ("Input image:")
plt.imshow(image)
plt.show()

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

```
Input image:
```




```
Output label:n02088364 beagle
```

12 Object Detection: GPU Task 5

With image classification, our network took an image and generated a classification. More specifically, our input was a 256X256X3 tensor of pixel values and our output was a 2-unit vector (since we had 2 classes) of probabilities.

Before seeing the obeject detection task, let's see input-output mapping in several tasks A vector where each index corresponds with the likelihood or the image of belonging to each classg.

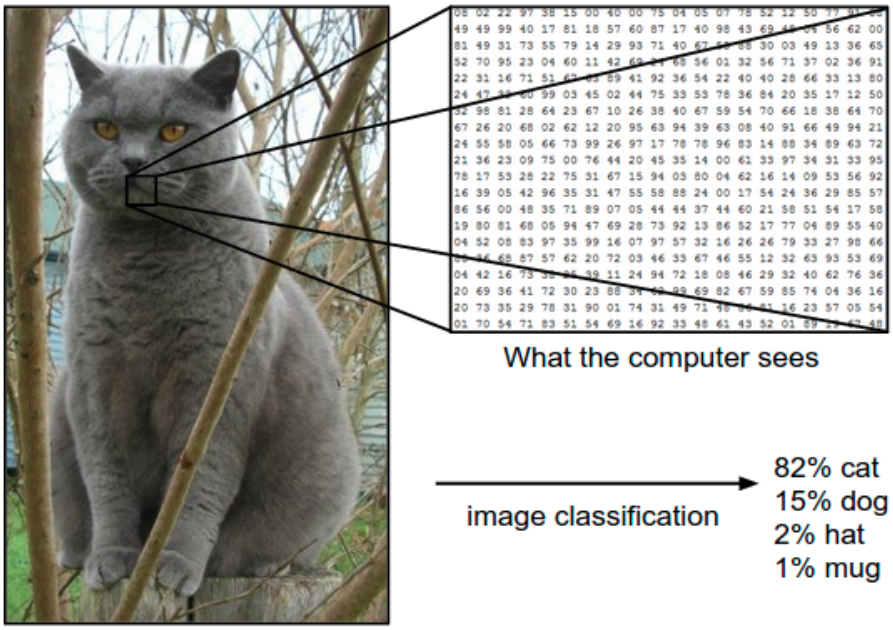


Image from the Stanford CS231 Course

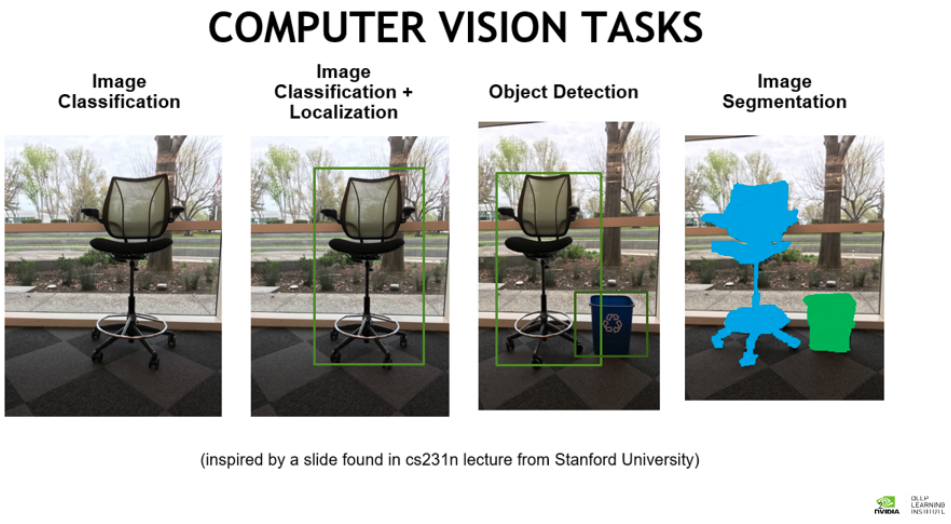
Workflow	Input	Output
Image Classification	Raw Pixel Values	A vector where each index corresponds with the likelihood or the image of belonging to each class

Workflow	Input	Output
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

Workflow	Input	Output
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

Workflow	Input	Output
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

let's see computer vision tasks

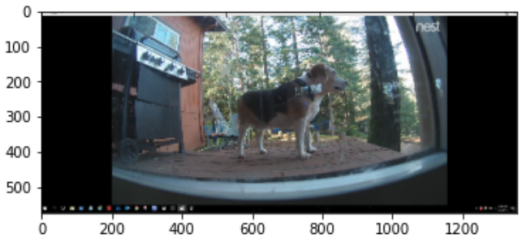


The new task is dectecting and localize objects within images.

```
import time
import numpy as np #Data is often stored as "Numpy Arrays"
import matplotlib.pyplot as plt #matplotlib.pyplot allows us to visualize results
import caffe #caffe is our deep learning framework, we'll learn a lot more about this later in this t
%matplotlib inline

MODEL_JOB_DIR = 'path/'  ## Remember to set this to be the job directory for your model
DATASET_JOB_DIR = 'path/'  ## Remember to set this to be the job directory for your dataset

MODEL_FILE = MODEL_JOB_DIR + '/deploy.prototxt'            # This file contains the description of
PRETRAINED = MODEL_JOB_DIR + '/snapshot_iter_735.caffemodel'   # This file contains the *weights*
MEAN_IMAGE = DATASET_JOB_DIR + '/mean.jpg'               # This file contains the mean image of the

# Tell Caffe to use the GPU so it can take advantage of parallel processing.
# If you have a few hours, you're welcome to change gpu to cpu and see how much time it takes to deplo
caffe.set_mode_gpu()
# Initialize the Caffe model using the model trained in DIGITS
net = caffe.Classifier(MODEL_FILE, PRETRAINED,
                       channel_swap=(2,1,0),
                       raw_scale=255,
                       image_dims=(256, 256))

# load the mean image from the file
mean_image = caffe.io.load_image(MEAN_IMAGE)
print("Ready to predict.")
```

assume that we get image from the security camera, but it is larger than input of model.

```
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()
```



Let's see forward propagation

The output above is the output of the last layer of the network we're using. The type of layer is "softmax"

Let's see another code to detect object

```
# Load the input image into a numpy array and display it
```

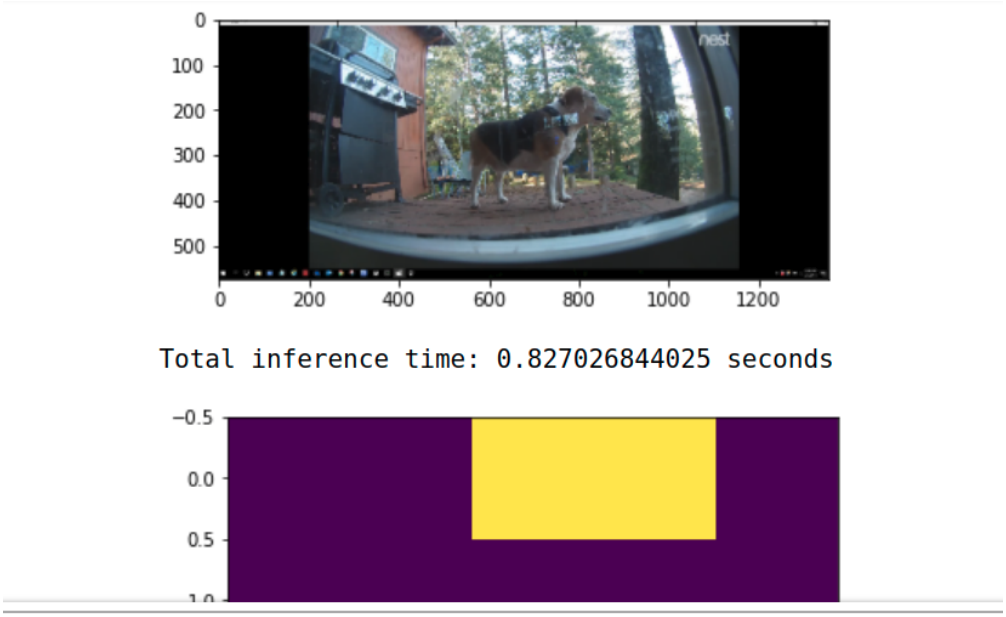
In [3]:
```
X = 0
Y = 0

grid_square = input_image[X*256:(X+1)*256,Y*256:(Y+1)*256]
# subtract the mean image (because we subtracted it while we trained th
grid_square -= mean_image
# make prediction
prediction = net.predict([grid_square])
print prediction
```

```
[[ 0.89857852  0.10142148]]
```

```
input_image = caffe.io.load_image(IMAGE_FILE)
plt.imshow(input_image)
plt.show()

# Calculate how many 256x256 grid squares are in the image
rows = input_image.shape[0]/256
cols = input_image.shape[1]/256

# Initialize an empty array for the detections
detections = np.zeros((rows,cols))

# Iterate over each grid square using the model to make a class prediction
start = time.time()
for i in range(0,rows):
    for j in range(0,cols):
        grid_square = input_image[i*256:(i+1)*256,j*256:(j+1)*256]
        # subtract the mean image
        grid_square -= mean_image
        # make prediction
        prediction = net.predict([grid_square])
        detections[i,j] = prediction[0].argmax()
end = time.time()

# Display the predicted class for each grid square
plt.imshow(detections, interpolation=None)

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

With another solution, how to use an image classification to detect objects

```
%matplotlib inline

import numpy as np
import matplotlib.pyplot as plt
import caffe
import time
```




```
MODEL_JOB_DIR = '##FIXME##'  ## Remember to set this to be the job number for your model
DATASET_JOB_DIR = '##FIXME##'  ## Remember to set this to be the job number for your dataset

MODEL_FILE = MODEL_JOB_DIR + '/deploy.prototxt'                  # Do not change
PRETRAINED = MODEL_JOB_DIR + '/snapshot_iter_735.caffemodel'     # Do not change
MEAN_IMAGE = DATASET_JOB_DIR + '/mean.jpg'                       # Do not change

# load the mean image
mean_image = caffe.io.load_image(MEAN_IMAGE)

# Choose a random image to test against
#RANDOM_IMAGE = str(np.random.randint(10))
IMAGE_FILE = '/dli/data/LouieReady.png'

# Tell Caffe to use the GPU
caffe.set_mode_gpu()
# Initialize the Caffe model using the model trained in DIGITS
net = caffe.Classifier(MODEL_FILE, PRETRAINED,
                       channel_swap=(2,1,0),
                       raw_scale=255,
                       image_dims=(256, 256))

# Load the input image into a numpy array and display it
input_image = caffe.io.load_image(IMAGE_FILE)
plt.imshow(input_image)
plt.show()
```

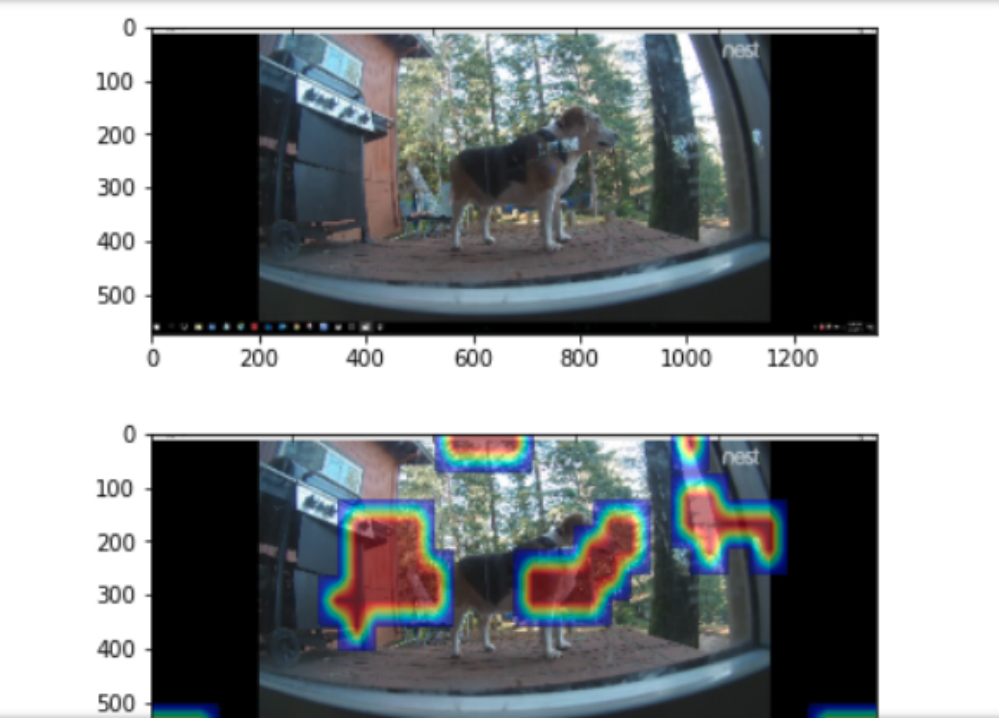
```
# Calculate how many 256x256 grid squares are in the image
rows = input_image.shape[0]/256
cols = input_image.shape[1]/256

# Subtract the mean image
for i in range(0,rows):
    for j in range(0,cols):
        input_image[i*256:(i+1)*256,j*256:(j+1)*256] -= mean_image

# Initialize an empty array for the detections
detections = np.zeros((rows,cols))

# Iterate over each grid square using the model to make a class prediction
start = time.time()
for i in range(0,rows):
    for j in range(0,cols):
        grid_square = input_image[i*256:(i+1)*256,j*256:(j+1)*256]
        # make prediction
        prediction = net.predict([grid_square])
        detections[i,j] = prediction[0].argmax()
end = time.time()

# Display the predicted class for each grid square
plt.imshow(detections)
plt.show()

# Display total time to perform inference
print 'Total inference time (sliding window without overlap): ' + str(end-start) + ' seconds'

# define the amount of overlap between grid cells
OVERLAP = 0.25
grid_rows = int((rows-1)/(1-OVERLAP))+1
grid_cols = int((cols-1)/(1-OVERLAP))+1

print "Image has %d*%d blocks of 256 pixels" % (rows, cols)
print "With overlap=%f grid_size=%d*%d" % (OVERLAP, grid_rows, grid_cols)

# Initialize an empty array for the detections
detections = np.zeros((grid_rows,grid_cols))

# Iterate over each grid square using the model to make a class prediction
start = time.time()
for i in range(0,grid_rows):
    for j in range(0,grid_cols):
        start_col = int(j*256*(1-OVERLAP))
        start_row = int(i*256*(1-OVERLAP))
       grid_square = input_image[start_row:start_row+256, start_col:start_col+256]
```

```
        # make prediction
        prediction = net.predict([grid_square])
        detections[i,j] = prediction[0].argmax()
end = time.time()

# Display the predicted class for each grid square
plt.imshow(detections)
plt.show()

# Display total time to perform inference
print ('Total inference time (sliding window with %f%% overlap: ' % (OVERLAP*100)) + str(end-start)

# now with batched inference (one column at a time)
# we are not using a caffe.Classifier here so we need to do the pre-processing
# manually. The model was trained on random crops (256*256->227*227) so we
# need to do the cropping below. Similarly, we need to convert images
# from Numpy's Height*Width*Channel (HWC) format to Channel*Height*Width (CHW)
# Lastly, we need to swap channels from RGB to BGR
net = caffe.Net(MODEL_FILE, PRETRAINED, caffe.TEST)
start = time.time()
net.blobs['data'].reshape(*[grid_cols, 3, 227, 227])

# Initialize an empty array for the detections
detections = np.zeros((rows,cols))

for i in range(0,rows):
    for j in range(0,cols):
        grid_square = input_image[i*256:(i+1)*256,j*256:(j+1)*256]
        # add to batch
        grid_square = grid_square[14:241,14:241] # 227*227 center crop
        image = np.copy(grid_square.transpose(2,0,1)) # transpose from HWC to CHW
        image = image * 255 # rescale
        image = image[(2,1,0), :, :] # swap channels
        net.blobs['data'].data[j] = image
    # make prediction
    output = net.forward()[net.outputs[-1]]
    for j in range(0,cols):
        detections[i,j] = output[j].argmax()
end = time.time()

# Display the predicted class for each grid square
plt.imshow(detections)
plt.show()

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

With the fully Convolutional network,

```
%matplotlib inline

import numpy as np
import matplotlib.pyplot as plt
import caffe
import copy
from scipy.misc import imresize
import time

MODEL_FILE = JOB_DIR + '/deploy.prototxt'                  # Do not change
PRETRAINED = JOB_DIR + '/snapshot_iter_735.caffemodel'    # Do not change

# Tell Caffe to use the GPU
caffe.set_mode_gpu()

# Load the input image into a numpy array and display it
input_image = caffe.io.load_image(IMAGE_FILE)
plt.imshow(input_image)
plt.show()

# Initialize the Caffe model using the model trained in DIGITS
# This time the model input size is reshaped based on the randomly selected input image
net = caffe.Net(MODEL_FILE,PRETRAINED,caffe.TEST)
net.blobs['data'].reshape(1, 3, input_image.shape[0], input_image.shape[1])
net.reshape()
transformer = caffe.io.Transformer({'data': net.blobs['data'].data.shape})
transformer.set_transpose('data', (2,0,1))
transformer.set_channel_swap('data', (2,1,0))
transformer.set_raw_scale('data', 255.0)

# This is just a colormap for displaying the results
my_cmap = copy.copy(plt.cm.get_cmap('jet')) # get a copy of the jet color map
my_cmap.set_bad(alpha=0) # set how the colormap handles 'bad' values

# Feed the whole input image into the model for classification
start = time.time()
out = net.forward(data=np.asarray([transformer.preprocess('data', input_image)]))
end = time.time()

# Create an overlay visualization of the classification result
im = transformer.deprocess('data', net.blobs['data'].data[0])
classifications = out['softmax'][0]
classifications = imresize(classifications.argmax(axis=0),input_image.shape,interp='bilinear').
classifications[classifications==0] = np.nan
plt.imshow(im)
plt.imshow(classifications,alpha=.5,cmap=my_cmap)
plt.show()
```

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



With DetectNet which an Fully Convolutional Networkd, The bulk of the layers in DetectNet are identical to the well-known GoogLeNet network.

Let's see the type of data, input and output for dectecting

Note:

1) The input and its corresponding output are correlated based on the file number.
2) The vector you're seeing consists of the (x,y) coordinates of the top left and bottom corner of the dog in the input image.
3) If we had enough data with surfboards, we could train a surfboard detector instead of a dog detector! We'll set our dataset to only look for dogs.

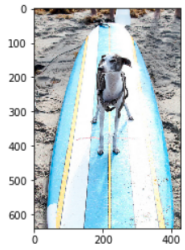
The following is the result from DetectNet.

The takeaway is

```
1. The right network and data for the job at hand are vastly superior to hacking your own solution.
2. The right network (and sometimes even pretrained model) might be available in DIGITS or on the [M
```

```
So one problem solving framework you can use is:

- determine if someone else has already solved your problem, use their model
- if they have not, determine if someone else has already solved a problem like yours, use their netw
```

In [9]:
```
input_image = caffe.io.load_image('##Image.png##') #!ls this directory to see what other images and label
plt.imshow(input_image)
plt.show()
```



Next, its corresponding label:

In [10]:
```
!cat 'Image.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
```

```
- if they have not, determine if you can identify the shortcomings of someone else's solution in sol
- if you can not, use an existing solution other problem solving techniques (eg. python) to solve yo
- either way, continue to experiment and take labs to increase your skill set!
```

13 Reference

- ImageNet
- Use the NVIDIA AMI on AWS (10 minutes)
- Get started with nvidia-docker (5 minutes)
- Get started with the NVIDIA DIGITS container (5 minutes)

Hyunyoung2_detectnet Generic Image Model

Found 1 bounding box.

Source image

Inference visualization

Job Status Done

- Initialized at 02:47:45 PM (1 second)
- Running at 02:47:46 PM (7 seconds)
- Done at 02:47:54 PM
  (Total - 8 seconds)

Infer Model Done

Notes

None