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

배포하기

"모델"은 어떤 파일들로 구성되어 있을까요?

이제 학습된 신경망을 학습 환경에서 분리하여 실제 활용을 위해 배포할 차례입니다. DIGITS에서 우리가 마지막으로 작업하던 곳에서부터 다시 출발하겠습니다.

DIGITS는 우리가 적용할 파일들을 다운로드할 수 있거나 가리킬 수 있는 경로에 저장합니다. 우리는 모델을 학습시킨 것과 동일한 서버에 모델을 적용할 것이므로 DIGIT가 생성하는 폴더 경로만을 가리키면 됩니다.

DIGITS (/digits) 열기

DIGITS 홈페이지에서 우리가 "Dogs vs. Cats"라 이름지은 모델을 선택하세요.

학습 중에 모델을 생성했거나 DIGITS의 "model" 탭에 있는 모델을 선택했을 경우 여러분이 가장 먼저 보게 될 것은 해당 모델의 DIGITS "작업 페이지(Job Page)"입니다. 작업 디렉토리(Job Directory)는 화면 좌상단에 있습니다.

- (위에서 하이라이트 된)작업 디렉토리를 복사한 후, 아래 셀의 코드 블록에서 ##FIXME## 부분을 지우고 붙여 넣으세요. 디렉토리 이름을 복사해 붙인 후, Shift+Enter로 셀을 실행하여, 디렉토리 이름을 `MODEL_JOB_DIR` 변수에 저장하세요.*

In [11]:

```
# MODEL_JOB_DIR = '##FIXME##'  ## Remember to set this to be the job directory for your model
MODEL_JOB_DIR = '/dli/data/digits/20180301-185638-e918'
!ls $MODEL_JOB_DIR
```

```
caffe_output.log    snapshot_iter_735.caffemodel    status.pickle
deploy.prototxt     snapshot_iter_735.solverstate   train_val.prototxt
original.prototxt   solver.prototxt
```

복사해서 붙이기가 잘 되었다면 여러분은 해당 디렉토리의 파일 목록을 볼 수 있을 것입니다. 앞으로 설명할 내용이 여러분의 실행 결과들과 다르다면, 먼저 복사해서 붙이기를 잘 하셨는지 확인하세요.

우리의 "model"은, 아키텍쳐와 가중치, 이렇게 두 개의 파일로 이루어져 있습니다.

아키텍쳐는 `deploy.prototxt` 라는 파일이고 가중치는 가장 최근의 스냅샷 파일인 `snapshot_iter_#.caffemodel` 입니다. 이번 경우, 스냅샷 735 번이 바로 다섯 번의 에포크(epoch)를 거쳐 학습된 가중치를 담고 있습니다.

In [12]:

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

```
Filepath to Architecture = /dli/data/digits/20180301-185638-e918/deploy.prototxt
Filepath to weights = /dli/data/digits/20180301-185638-e918/snapshot_iter_735.caffem
odel
```

다음으로, 우리가 만들고 있는 프로그램에서 이 파일들을 읽고, 처리할 수 있도록 만들어야 합니다. 이러한 기본적인 적용을 위해서는, 우리 프로그램이 모델 파일을 해석할 수 있게 만들어야 하기에, 모델 파일들이 작성된 프레임워크 환경을 설치 또는 포함시켜야 합니다. 우리는 나중에 프레임워크 설치가 불필요한 환경에 모델을 적용하는 방법을 배울 것입니다. 또한 병렬 처리의 장점을 활용하기 위해 GPU를 사용할 것입니다. 다시 말씀 드리지만, 우리의 모델은 병렬화를 통해 가속화시킬 수 있는 수 십 만 개의 연산을 포함하고 있습니다.

In [13]:

```
import caffe
caffe.set_mode_gpu()
```

다음으로 "net"이라 불리는 "분류기(Classifier)"를 만들 차례입니다. 작업 흐름이 일반적인 것일수록 기존 도구를 이용해 프로젝트 만들기가 더 용이합니다. 이번 경우, 이미지 분류는 매우 일반적이며, 따라서 아래 코드 블럭은 그저 아키텍쳐 파일, 가중치 파일, 데이터를 가져와서 손쉽게 일반 동작들을 수행합니다.

In [14]:

```
# 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 t
                       raw_scale=255) #Each pixel value is a number between 0 and 255
                       #Each "channel" of our images are 256 x 256
```

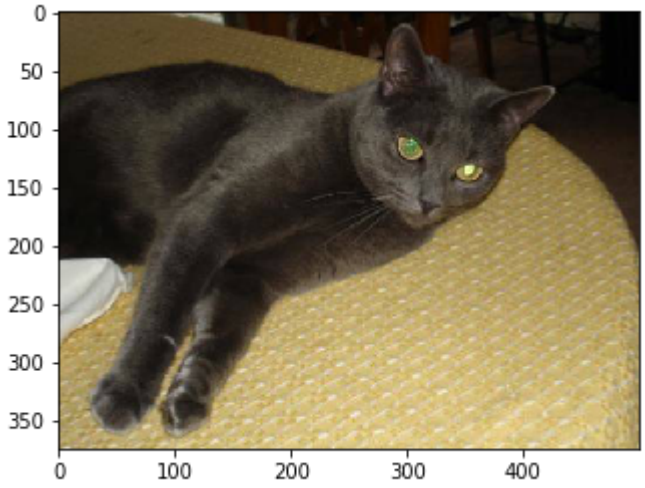
분류기 클래스는 "predict"라는 메소드를 포함하는데, 이 메소드는 위의 코드 블록에서 정의된 것과 같은 이미지를 입력으로 받아, 각 카테고리에 속하는 정도를 출력으로 내보냅니다.

원하는 입력 만들기: 전처리(Preprocessing)

데이터 세트로부터 레이블 있는 이미지를 가져다가 분류하는 쉬운 예제부터 해 봅시다. 아래 셀을 실행시키면 이미지를 불러와서 화면으로 볼 수 있습니다.

In [15]:

```
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()
```



우리가 가진 이미지는 이것이지만, 이는 신경망이 실제 '입력'으로 원하는 형식이 아닙니다.

추론을 위한 데이터를 준비할 때 지켜야 할 황금률이 있습니다.

> 모델 훈련 이전에 했던 작업은 추론 이전에도 똑같이 해야 한다.

지난 섹션에서 여러분은 DIGITS가 여러분의 모델을 훈련할 때 생성된 파일을 본 적이 있습니다. 이번 섹션에서는 DIGIT가 데이터 세트를 생성할 때 만드는 파일을 살펴볼 것입니다.

여러분이 학습에 사용한 **데이터 세트**의 작업 디렉토리는 "Dogs and Cats" 모델 페이지의 데이터 세트를 선택하거나 DIGIT의 "dataset" 탭 아래의 데이터 세트를 선택하면 찾을 수 있습니다. 이것은 모델이 있던 곳과 같은 위치인데 번호만 다릅니다.

아래 셀의 코드 블록에서 ##FIXME## 부분을 지우고 데이터 세트의 디렉토리를 붙여 넣으세요. 셀을 실행해서 DATA_JOB_DIR 변수를 설정한 후, 해당 폴더에 어떤 것들이 들어 있는지 확인하세요.

In [17]:

```
DATA_JOB_DIR = '##FIXME##'  ## Remember to set this to be the job directory for your model
DATA_JOB_DIR = '/dli/data/digits/20180222-165843-ada0'
!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
```

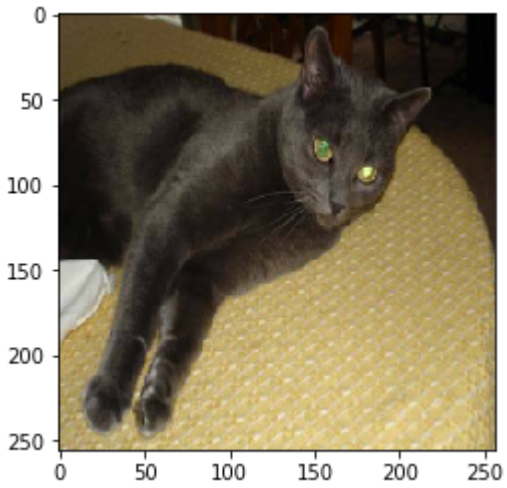
다시 말하지만 여기에는 여러분이 지금 당장 필요한 것보다 더 많은 정보가 있습니다. 데이터 과학과 데이터 준비 분야에는 알아야할 내용이 끝도 없이 많습니다만, 여러분께서 다양한 심층 학습 문제를 다루어가면서 점점 더 명확해 질 것이니 걱정하지 마세요.

지금 DIGITS는 학습에 앞서 두 단계를 진행하는데, 우리는 이것을 전처리(preprocessing)라고 합니다.

1) DIGITS는 주어진 이미지를 256x256 컬러 이미지로 바꿉니다.

In [18]:

```
import cv2
input_image=cv2.resize(input_image, (256, 256), 0,0)
plt.imshow(input_image)
plt.show()
```



2) DIGITS는 학습에 소요되는 계산량을 줄이기 위해 주어진 이미지에서 평균 이미지값을 뺌으로써 이미지를 *정규화(normalize)*합니다.

평균 이미지를 읽어들여서 테스트 이미지로부터 뺄셈을 하는 작업은 아래와 같습니다.

In [19]:

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

이제 우리는 데이터를 있는 그대로 가져온 후, 이것을 신경망이 원하는 형태로 바꾸어 주었습니다. 다음으로는 신경망에서 어떤 출력이 나오는지 살펴보겠습니다.

순전파(Forward Propagation): 모델 사용하기

중요한 부분입니다. 아래 함수를 보겠습니다.

```
prediction = net.predict([grid_square])
```

여타의 함수 (https://www.khanacademy.org/computing/computer-programming/programming#functions)와 마찬가지로, `net.predict` 는 입력(여기에서는 `ready_image`)을 전달하면 출력(여기에서는 `prediction`)을 내어 줍니다. 이 함수는 이미지를 확률 값의 벡터로 바꾸어 주기 위한 일련의 행렬 계산을 반복합니다.

아래 셀을 실행해서 앞서 본 레이블 있는 데이터로부터의 예측값을 확인해 보세요.

In [21]:

```
# make prediction
prediction = net.predict([ready_image])
print prediction
print('0인 label일 확률 , 1인 label인 확률')
```

```
[[ 0.70993775  0.29006225]]
0인 label일 확률 , 1인 label인 확률
```

흥미롭지만 그다지 새로운 정보를 포함하고 있지는 않습니다. 우리의 신경망은 정규화된 256x256 컬러 이미지를 받아들여 길이 2의 벡터를 생성해냈습니다.

유용한 출력 생성하기: 후처리(Postprocessing)

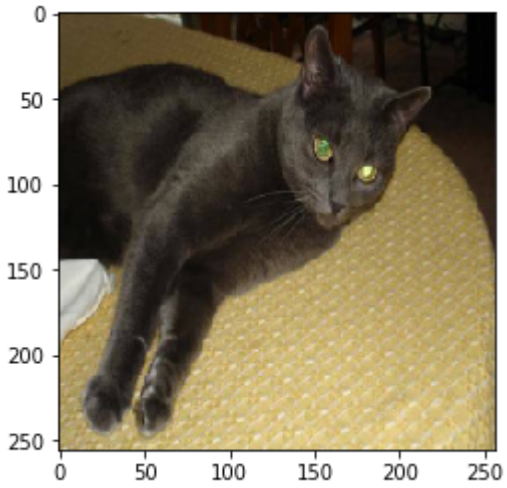
이제 우리는 원하는 것은 무엇이든 만들어낼 수 있게 되었습니다. 유일한 한계는 여러분의 프로그래밍 경험입니다. 창의적인 무언가를 만들기 전에 기본적인 것을 먼저 만들어 봅시다. 이번 코드는 우리의 신경망이 "cat"일 가능성 값보다 "dog"일 가능성 값을 더 높은 값으로 출력하는지를 결정합니다. 만일 그러하다면, 우리의 가상 개 출입구로 개가 들어올 때 적합할 이미지를 잘 출력할 것입니다. 그렇지 않다면 신경망이 고양이라고 결정할 때 나올 만한 이미지를 보여 줄 것입니다.

In [22]:

```
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:
```



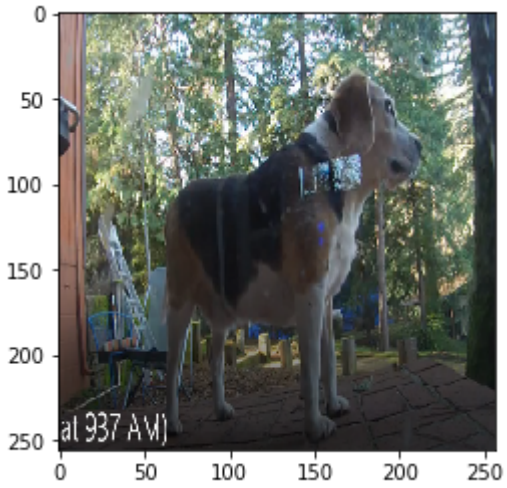

```
Output:
Sorry cat:( https://media.giphy.com/media/jb8aFEQk3tADS/giphy.gif (https://media.gip
hy.com/media/jb8aFEQk3tADS/giphy.gif)
```

이제 개 출입구가 보게 될 이미지를 이용하여 테스트를 진행하기 위해 모든 코드를 합쳐 보겠습니다.

In [23]:

```
##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"
```

```
Input Image:
```




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

이제 개 출입구 시뮬레이터를 완성했습니다. 우리는 카메라로 입력을 받아 이를 신경망이 원하는 데이터 형식으로 바꾸고, 출력을 생성하고, 그 출력을 사용자에게 의미있는 것으로 바꾸는 과정을 진행했습니다.

여러분은 이것을 이용해서 개 출입구 문에 모터를 달아서 개폐를 제어하는 장치를 만들 수도 있겠지요. 최소한 심층 학습과 관련된 부분은 다 만들었네요. 위의 코드 블록으로 시도해 볼 수 있는 다른 이미지가 어떤 것이 있는지 보기 위해 테스트 이미지(학습에 사용되지 않은 이미지)의 목록을 살펴 보세요. 아래 셀의 명령을 실행하면 됩니다. 목록에 있는 이미지 중 일부는 잘못된 분류를 출력할 수 도 있습니다. 만족하실 때까지 이미지를 테스트해보시고 강좌를 계속 들으시면서 성능을 향상시킬 방법을 찾아보세요.

In [24]:

```
!ls /dli/data/dogscats/test
```

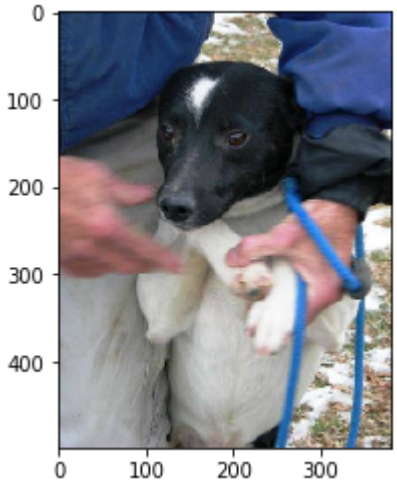
```
1.jpg       11605.jpg  1963.jpg  3570.jpg  5178.jpg  6786.jpg  8393.jpg
10.jpg      11606.jpg  1964.jpg  3571.jpg  5179.jpg  6787.jpg  8394.jpg
100.jpg     11607.jpg  1965.jpg  3572.jpg  518.jpg   6788.jpg  8395.jpg
1000.jpg    11608.jpg  1966.jpg  3573.jpg  5180.jpg  6789.jpg  8396.jpg
10000.jpg   11609.jpg  1967.jpg  3574.jpg  5181.jpg  679.jpg   8397.jpg
10001.jpg   1161.jpg   1968.jpg  3575.jpg  5182.jpg  6790.jpg  8398.jpg
10002.jpg   11610.jpg  1969.jpg  3576.jpg  5183.jpg  6791.jpg  8399.jpg
10003.jpg   11611.jpg  197.jpg   3577.jpg  5184.jpg  6792.jpg  84.jpg
10004.jpg   11612.jpg  1970.jpg  3578.jpg  5185.jpg  6793.jpg  840.jpg
10005.jpg   11613.jpg  1971.jpg  3579.jpg  5186.jpg  6794.jpg  8400.jpg
10006.jpg   11614.jpg  1972.jpg  358.jpg   5187.jpg  6795.jpg  8401.jpg
10007.jpg   11615.jpg  1973.jpg  3580.jpg  5188.jpg  6796.jpg  8402.jpg
10008.jpg   11616.jpg  1974.jpg  3581.jpg  5189.jpg  6797.jpg  8403.jpg
10009.jpg   11617.jpg  1975.jpg  3582.jpg  519.jpg   6798.jpg  8404.jpg
1001.jpg    11618.jpg  1976.jpg  3583.jpg  5190.jpg  6799.jpg  8405.jpg
10010.jpg   11619.jpg  1977.jpg  3584.jpg  5191.jpg  68.jpg    8406.jpg
10011.jpg   1162.jpg   1978.jpg  3585.jpg  5192.jpg  680.jpg   8407.jpg
10012.jpg   11620.jpg  1979.jpg  3586.jpg  5193.jpg  6800.jpg  8408.jpg
10013.jpg   11621.jpg  198.jpg   3587.jpg  5194.jpg  6801.jpg  8409.jpg
```

모두 합치기

모델 적용 과정을 모두 합쳐서 주피터 노트북 밖에서 어떻게 보일지 생각해 보세요. 파이썬 파일 pythondeployment.py (../../../../edit/tasks/task3/task/pythondeployment.py)을 보시면 위에서 다룬 것과 같은 코드들이 하나의 파일로 합쳐진 것을 보실 수 있습니다. 코스 말미의 평가 시간에 이와 같은 방법으로 코드를 만들어야 하니 잘 살펴보세요. 테스트 이미지에 대한 경로명을 아래 셀에 추가하시고 출력을 살펴보세요.

In [25]:

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



위에서 본 이미지를 입력으로 하는 파이썬 애플리케이션을 실행하세요. 에러나 경고 메시지가 나와도 좋으니 모두 무시하고 제일 아래쪽으로 스크롤하세요.

In [26]:

```
!python pythondeployment.py $TEST_IMAGE 2>/dev/null
```

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

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