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

모델 개선하기

여러분은 성공적으로 모델 학습을 배우셨으니 이제 예술적인 경지의 모델을 향하여 매진합시다. 본 실습에서는 심층 학습 수행자로서의 여러분이 원하는 결과를 향해 나아가는 데 유용하게 사용될 지렛대를 배우게 됩니다. 그 과정에서 이를 가능하게 만드는 기술적 사항을 둘러싼 껍질을 한 층 벗겨 보도록 하겠습니다.

개/고양이 분류기를 다시 불러 옵시다.

DIGITS 홈 화면 시작하기 (/digits)

여러분이 만든 "Dogs vs. Cats" 모델을 선택하세요.

DIGITS는 우선 모델이 학습 중에 만든 그래프를 보여줄 것입니다.

학습 손실(training loss), 검증 손실(validation loss), 정확도(accuracy)의 3 가지 수치가 나와 있습니다. 중간에 간혹 값이 뛰는 경우가 있을 수도 있지만, 학습 손실과 검증 손실 값은 에포크가 거듭됨에 따라 점점 줄어들어야 합니다. 정확도는 모델이 검증 데이터를 올바르게 분류하는 능력을 측정합니다. 마우스를 데이터 포인트 위에 올려두면 정확한 값을 볼 수 있습니다. 이 경우 마지막 에포크 후의 정확도는 약 80% 입니다. 초기 신경망이 랜덤으로 생성되기에 여러분의 결과는 약간 다를 수도 있습니다.

그래프를 분석해 보면 눈에 띄는 것 하나는 시간이 지남에 따라 정확도는 증가하고 손실은 감소한다는 점입니다. 자연스럽게 다음과 같은 질문이 떠오를 것입니다. "학습을 오래 하면 할수록 모델은 계속 좋아지는 것일까?" 잠시 이에 대해 실험하고 논의해 보겠습니다.

더 공부하기

모든 부모님과 선생님의 가르침을 따라 모델에게 더 공부하라고 요청해서 모델의 정확도를 개선해 봅시다.

에포크(epoch)는 신경망에 데이터를 한 번 완전히 보여주는 것을 의미합니다. 본 강좌 초반에 우리는 1 에포크(epoch)를 그림 카드 한 세트를 한 번 다 보는 것에 비유했습니다. "더 공부하기"는 우리 모델이 공부한 마지막 지점부터 시작하여 에포크를 추가로 실행하는 것입니다.

이 *상태*에 접근하려면 여러분의 모델 페이지의 제일 아랫부분까지 스크롤해서 "사전에 학습된 모델 만들기(Make Pretrained Model)" 버튼을 클릭하세요.

이 동작은 두 가지를 저장합니다.

1. "AlexNet"을 선택함으로써 결정된 "신경망 아키텍쳐"
2. 여러분의 모델이 초기 다섯 번의 에포크를 거치면서 조정한 파라미터(parameters) 형태로 "학습한" 내용

학습률

모델의 현 상태와 함께 상태가 어떻게 변하는지를 이해하는 것이 중요합니다. 학습 그래프 아래에 있는 그래프를 보세요.

세 가지 의문이 드실 것입니다.

Q1) "학습률(learning rate)"이 무엇인가?

Q2) 학습 세션에 걸쳐 왜 학습률은 줄어드는가?

Q3) 누가 학습률을 제어하는가?

A1) 학습률은 학습 중에 "가중치"가 변하는 속도를 말합니다. 각 가중치는 특정 위치의 손실 값과 학습률의 곱을 줄이는 방향으로 이동합니다.

A2) 학습률이 학습 세션에 걸쳐 점점 감소하는 이유는 신경망이 이상적인 솔루션에 점점 접근하기 때문입니다. 학습할 신경망은 첫 단계에서는 장차 자신이 무엇을 보게 될지 아무것도 알지 못합니다. 개는 신경 쓰지 마세요. 신경망은 이미지나 센서 데이터를 보게 될지 생각지도 못하고 있습니다. 이 경우 이상적인 솔루션 쪽으로 크게 뛰어가는 것이 합리적입니다. 몇 번의 에포크 후, 가중치는 점점 더 좋아지고 신경망이 보는 개개의 이미지에 덜 반응하는 것이 중요합니다.

A3) 학습률은 여러분이 제어하는 것입니다. 학습률은 학습 세션을 설정할 때 정하는 여러 "하이퍼파라미터" 중의 하나입니다. 잠시 후 학습률을 조정할 텐데 그 이유는 다음과 같습니다.

학습 세션 말미에는 학습률이 느려지기 때문에 기학습된 신경망을 가지고 작업을 시작할 때 우리는 제일 마지막 상태에서 시작해야 합니다. 기학습 신경망의 학습률을 0.0001로 하고 학습을 시작합시다.

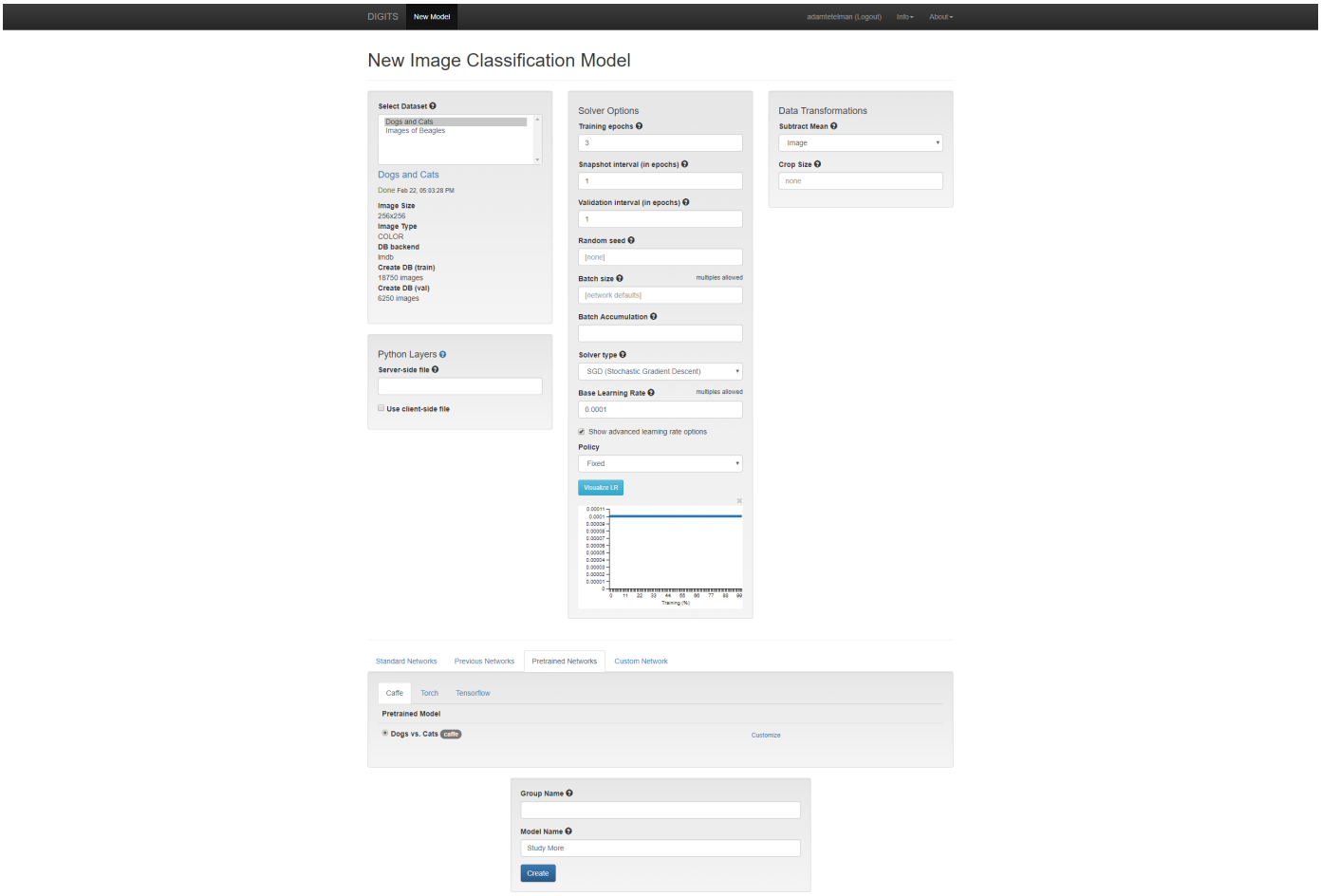
사전에 학습된 모델로부터 시작하기

이제 이 시작점으로부터 새로운 모델을 만들 수 있습니다. 화면 좌상단의 "DIGITS"를 클릭하여 DIGITS의 홈 화면으로 돌아간 후, 이전과 같이 새로운 이미지 분류 모델을 만드세요.

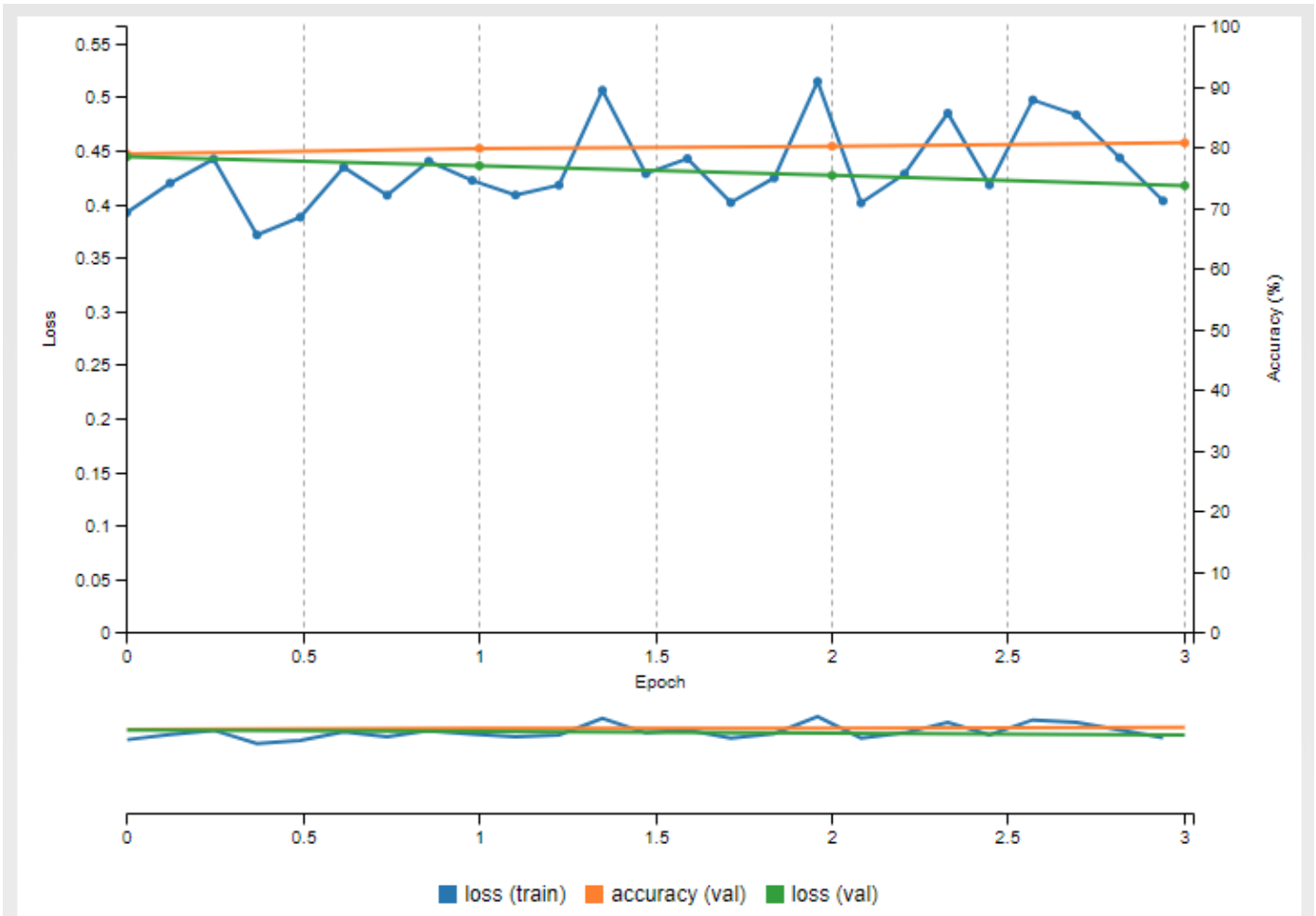
New Model (Images) -> Classification

- 이전과 같은 Dogs and Cats 데이터 세트를 선택하세요.
- 3에서 8 사이의 에포크 값을 설정하세요. (새 모델을 처음부터 만들 때는 여기에서 아예 큰 값을 주고 시작할 수 있음에 유의 하세요.)
- 학습률을 0.0001로 하세요.
- "고급 옵션(advanced option)"에서 학습률을 "고정(fix)"하세요.
- 이번에는 "표준 신경망(Standard Network)"이 아니라 "사전에 학습된 모델(Pretrained Network)"을 선택하세요.
- 여러분이 만드신 기학습 모델인 "Dogs vs. Cats"를 선택하세요.
- 모델에 이름을 붙이세요. 우리는 "Study more"로 하겠습니다.
- 생성(Create)을 클릭하세요.

여러분의 설정은 아래와 같이 보여야 합니다.



모델을 생성할 때, 아래 그래프를 보실 수 있습니다.



아래 사항에 유의하세요.

1. 기대한 바와 같이, 정확도는 우리의 원래 모델이 그랬던 바와 같이 80% 근처의 값입니다.
2. 정확도는 계속 증가합니다. 이는 에포크가 반복되면 보통 성능도 함께 증가함을 의미합니다.
3. 정확도가 증가하는 *속도*는 점차 감소합니다. 즉, 같은 데이터를 더 많이 보여주는 것이 성능 향상의 유일한 방법은 아니라는 뜻입니다.

여러분이 성능 향상에 이용할 수 있는 지렛대는 네 가지가 있습니다. 이들에 정성을 기울이면 성능 향상으로 보답받으실 것입니다.

1) **데이터** - 우리 모델이 사용될 환경을 나타내는 크고 충분히 다양한 데이터 세트. 데이터 큐레이션은 그 자체가 예술입니다.

2) **하이퍼파라미터** - 학습률과 같은 옵션의 변경은 학습 "스타일"을 바꿀 수 있습니다. 오늘날, 올바른 하이퍼파라미터는 경험에 기반하여 수동으로 결정됩니다. 어떤 종류의 작업이 어떤 하이퍼파라미터에 잘 반응하는지에 대한 직관력을 키움으로써 여러분의 능력은 향상될 것입니다.

3) **학습 시간** - 더 많은 에포크는 어느 수준까지는 성능을 향상시킵니다. 그 어느 수준을 넘어서는 과도한 학습은 오버피팅(overfitting)을 일으킵니다. 다만 이것이 여러분이 하는 유일한 개입은 아닙니다.

4) **신경망 아키텍쳐** - 다음 섹션에서 신경망 아키텍쳐에 대한 실험을 진행할 것입니다. 이것은 심층 학습을 통해 문제를 해결하기 위해서는 신경망 아키텍처에 대한 지식이 필요하다는 미신에 맞서기 위해 마지막 개입으로 나열되어 있습니다. 이 분야는 매력적이고 강력하며, 수학에 관한 연구를 통해 여러분의 실력을 향상시킬 수 있습니다.

수상작 모델을 적용하기

방금 여러분이 다시 학습시킨 모델을 적용하는 대신에, 성능 향상의 지름길을 소개할 시간입니다. 바로 전문가들이 미리 학습시킨 모델을 적용하는 것입니다.

이 섹션에서는 다른 사람들이 만든 신경망을 적용하는 방법을 학습함으로써, 그들이 연구하고, 오랜 계산을 하고, 데이터 큐레이션에 신경 써서 얻은 우수한 성능을 쉽게 가져다 사용할 수 있게 될 것입니다.

본 강좌에서 사작한 심층 학습 작업 흐름이 *이미지 분류*에 대한 것어었음을 기억하세요. 우리가 이 작업부터 시작한 이유는 이미지 분류가 심층 학습 분야에서 가장 발전된 분야이기 때문입니다. 이러한 발전은 "ImageNet" (https://qz.com/1034972/the-data-that-changed-the-direction-of-ai-research-and-possibly-the-world/)이라는 경연에서 솔루션을 향상시켜 온 연구 집단의 노력에 힘입은 바가 큽니다.

"ImageNet"은 1000 여 개의 클래스로 나뉜 일반 이미지의 거대한 데이터 세트를 가지고 있습니다. 경연에서는 이들 데이터 세트에 대하여 가장 작은 손실 값을 기록한 팀에게 상을 수여합니다. 우리가 이용한 AlexNet은 2012 년에 우승을 한 신경망입니다. 이후로 구글과 마이크로소프트 팀들이 수상을 했습니다.

신나는 점은 이것입니다. 우리는 그들의 신경망 아키텍쳐 뿐만 아니라 위에서 나온 네 가지 지렛대, 즉, 데이터, 하이퍼파라미터, 학습 시간, 신경망 아키텍쳐를 모두 적용하여 만들어낸 학습 가중치도 가져다 쓸 수 있습니다. 학습이나 데이터 수집을 할 필요 없이 수상작 모델을 *적용*할 수 있습니다.

이 모델을 적용하기 위해 필요한 것은 모델의 아키텍쳐와 가중치 뿐입니다. 구글에서 "pretrained model alexnet imagenet caffe"라고 검색하면 이 모델을 다운로드할 수 있는 페이지들을 볼 수 있습니다.

우리는 wget이라는 도구로 다운로드할 것입니다. wget은 여러분의 로컬 컴퓨터를 거칠 필요 없이 작업 중인 서버로 직접 다운로드하는 멋진 방법입니다.

In [1]:

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

```
--2020-02-24 11:36:54--  http://dl.caffe.berkeleyvision.org/bvlc_alexnet.caffemodel
 (http://dl.caffe.berkeleyvision.org/bvlc_alexnet.caffemodel)
Resolving dl.caffe.berkeleyvision.org (dl.caffe.berkeleyvision.org)... 128.32.162.15
0
Connecting to dl.caffe.berkeleyvision.org (dl.caffe.berkeleyvision.org)|128.32.162.1
50|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 243862414 (233M) [application/octet-stream]
Saving to: ‘bvlc_alexnet.caffemodel’

bvlc_alexnet.caffem 100%[===================>] 232.56M  30.0MB/s    in 8.1s

2020-02-24 11:37:03 (28.6 MB/s) - ‘bvlc_alexnet.caffemodel’ saved [243862414/24386
2414]

--2020-02-24 11:37:03--  https://raw.githubusercontent.com/BVLC/caffe/master/models/
bvlc_alexnet/deploy.prototxt (https://raw.githubusercontent.com/BVLC/caffe/master/mo
dels/bvlc_alexnet/deploy.prototxt)
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 151.101.248.133
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|151.101.248.133
|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 3629 (3.5K) [text/plain]
Saving to: ‘deploy.prototxt.1’

deploy.prototxt.1   100%[===================>]   3.54K  --.-KB/s    in 0s

2020-02-24 11:37:03 (108 MB/s) - ‘deploy.prototxt.1’ saved [3629/3629]
```

이들은 우리가 모델을 처음부터 만들 때 DIGITS가 생성한 것과 같은 파일입니다. DIGITS에서 가져온 또다른 파일 하나는 학습에 사용되는 평균 이미지입니다. 아래처럼 다운로드합니다.

In [2]:

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

```
--2020-02-24 11:37:04--  https://github.com/BVLC/caffe/blob/master/python/caffe/imag
enet/ilsvrc_2012_mean.npy?raw=true (https://github.com/BVLC/caffe/blob/master/pytho
n/caffe/imagenet/ilsvrc_2012_mean.npy?raw=true)
Resolving github.com (github.com)... 140.82.114.3
Connecting to github.com (github.com)|140.82.114.3|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://github.com/BVLC/caffe/raw/master/python/caffe/imagenet/ilsvrc_2012
_mean.npy (https://github.com/BVLC/caffe/raw/master/python/caffe/imagenet/ilsvrc_201
2_mean.npy) [following]
--2020-02-24 11:37:04--  https://github.com/BVLC/caffe/raw/master/python/caffe/image
net/ilsvrc_2012_mean.npy (https://github.com/BVLC/caffe/raw/master/python/caffe/imag
enet/ilsvrc_2012_mean.npy)
Reusing existing connection to github.com:443.
HTTP request sent, awaiting response... 302 Found
Location: https://raw.githubusercontent.com/BVLC/caffe/master/python/caffe/imagenet/
ilsvrc_2012_mean.npy (https://raw.githubusercontent.com/BVLC/caffe/master/python/caf
fe/imagenet/ilsvrc_2012_mean.npy) [following]
--2020-02-24 11:37:04--  https://raw.githubusercontent.com/BVLC/caffe/master/python/
caffe/imagenet/ilsvrc_2012_mean.npy (https://raw.githubusercontent.com/BVLC/caffe/ma
ster/python/caffe/imagenet/ilsvrc_2012_mean.npy)
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 151.101.248.133
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|151.101.248.133
|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1572944 (1.5M) [application/octet-stream]
Saving to: ‘ilsvrc_2012_mean.npy?raw=true’

ilsvrc_2012_mean.np 100%[===================>]   1.50M  --.-KB/s    in 0.08s

2020-02-24 11:37:04 (20.0 MB/s) - ‘ilsvrc_2012_mean.npy?raw=true’ saved [1572944/1
572944]
```

이 파일들은 이제 서버에 아래와 같이 저장되었습니다.

```
/dli/tasks/task4/task/deploy.prototxt
/dli/tasks/task4/task/bvlc_alexnet.caffemodel
/dli/tasks/task4/task/ilsvrc_2012_mean.npy
```

이미지 하나를 이 모델에 적용하기 위해 우리가 아는 내용을 이용해 봅시다. 모델을 초기화하는 것으로 시작합니다.

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 = '../../../data/deploy.prototxt' #a locally stored version
WEIGHTS = '../../../data/bvlc_alexnet.caffemodel'#a locally stored version
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
```

다음으로는 신경망이 원하는 형식의 이미지를 만듭니다. 이는 지난번 모델에서 사용한 *선처리* 방식과 다름에 유의하세요. ImageNet이 선처리하는 방식을 알아보기 위하여 Caffe 웹사이트 (http://caffe.berkeleyvision.org/gathered/examples/imagenet.html)를 참고하세요.

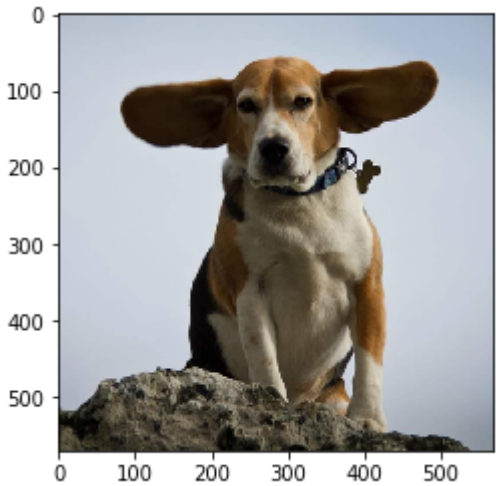
In [4]:

```
#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

# 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
net.blobs['data'].reshape(1,        # batch size
                          3,         # 3-channel (BGR) images
                          227, 227)  # image size is 227x227

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



함수를 실행하고 결과를 시각화하세요.

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
```

```
array([[  9.16413256e-09,   2.83365609e-10,   7.52338247e-10,
          4.40543557e-09,   1.59321656e-09,   2.50845039e-10,
          9.11683351e-09,   7.06857861e-09,   4.30004860e-07,
          1.74467052e-09,   6.42121734e-08,   1.39848586e-08,
          1.75973647e-09,   6.84332702e-10,   7.38009098e-10,
          1.02457343e-09,   5.74614312e-09,   1.74485359e-09,
          1.63354830e-09,   8.89102514e-10,   3.16383941e-09,
          2.19129337e-09,   5.76722226e-10,   5.85071769e-10,
          4.38478809e-09,   1.03755871e-09,   5.34484812e-08,
          2.29038992e-08,   6.19334806e-10,   4.50705162e-09,
          4.61923726e-08,   4.58914629e-10,   1.16998555e-09,
          1.19662875e-08,   2.55975219e-09,   5.51481172e-08,
          3.34089506e-10,   5.35509592e-09,   5.17955456e-10,
          2.10799858e-08,   1.50125228e-08,   5.83575757e-11,
          4.87324847e-09,   1.03735476e-09,   8.35556779e-09,
          9.00073849e-10,   1.13103070e-07,   3.44919222e-08,
          1.39825052e-09,   7.03649583e-09,   9.47259160e-10,
          1.06990488e-08,   7.44618189e-09,   3.81153198e-09,
          1.78574879e-08,   2.32537118e-08,   2.21020464e-08,
```

사용자에게 의미있는 출력 만들기

여러분이 보시는 것은 입력 이미지가 1000 개의 클래스 중 어디에 속할지를 나타내는 확률값 배열입니다. 이것을 좀 더 의미있는 정보로 만듭시다.

In [6]:

```
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
```

더 낫네요. 각 번호가 무엇을 의미하는지 알아보기 위해 ImageNet (https://gist.github.com/yrevar/942d3a0ac09ec9e5eb3a)을 참고하세요. 훨씬 낫죠? 유용한 end-to-end 적용을 위해 이 기능을 추가합시다.

레이블 불이는 데 사용할 사전을 가져오기 위해 wget을 이용합시다.

In [7]:

```
!wget https://raw.githubusercontent.com/HoldenCaulfieldRye/caffe/master/data/ilsvrc12/synset_words
labels_file = 'synset_words.txt'
labels = np.loadtxt(labels_file, str, delimiter='Wt')

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

```
--2020-02-24 11:37:08--  https://raw.githubusercontent.com/HoldenCaulfieldRye/caffe/
master/data/ilsvrc12/synset_words.txt (https://raw.githubusercontent.com/HoldenCaulf
ieldRye/caffe/master/data/ilsvrc12/synset_words.txt)
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 151.101.248.133
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|151.101.248.133
|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 31675 (31K) [text/plain]
Saving to: ‘synset_words.txt.2’

synset_words.txt.2  100%[===================>]  30.93K  --.-KB/s    in 0.01s

2020-02-24 11:37:09 (2.64 MB/s) - ‘synset_words.txt.2’ saved [31675/31675]

output label: n02088364 beagle
```

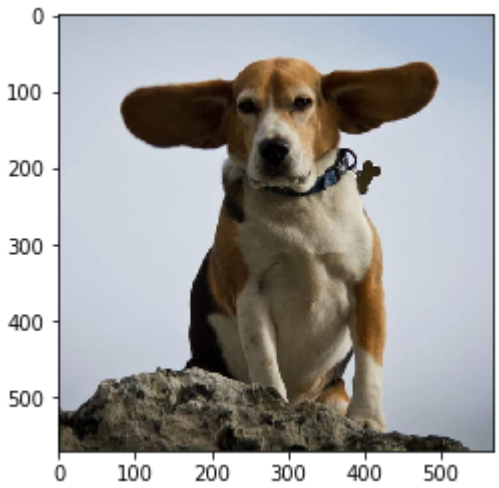
우리의 애플리케이션이 하는 일을 명확히 보여주기 위해, 여기에 입력과 출력이 있습니다.

In [8]:

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

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

```
Input image:
```




```
Output label:n02088364 beagle
```

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