Capstone Project - Udacity Machine Learning Engineer Nanodegree

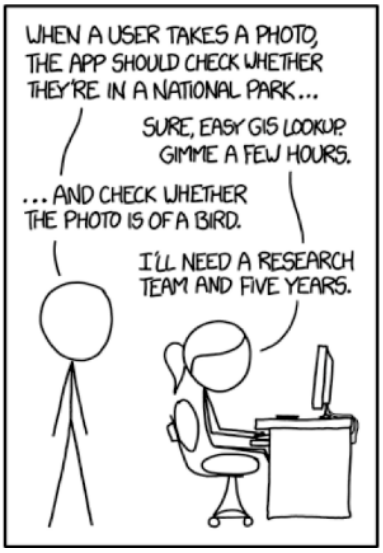
Definition

Project Overview

Deep learning is the new big trend in machine learning. It had many recent successes in computer vision, automatic speech recognition and natural language processing. The goal of this project is to give you a hands-on introduction to deep learning. To do this, we will build a Cat/Dog image classifier using a deep learning algorithm called convolutional neural network (CNN) and a ImageNet dataset.

Deep Learning attempts to use large volumes of unstructured data, such as images and audio clips, to learn hierarchical models which capture the complex structure in the data and then use these models to predict properties of previously unseen data. For example, DL has proven extremely successful at learning hierarchical models of the visual features and concepts represented in handheld camera images and then using those models to automatically label previously unseen images with the objects present in them.

Humans are very good at recognizing visual patterns--including the faces of people you know--while it's maddeningly complex for a computer to "understand" something visually, dot-by-dot. We will build a Cat/Dog image classifier inspired from Kaggle competition (https://www.kaggle.com/c/dogs-vs-cats).



Problem Statement

Our goal is to build a machine learning algorithm capable of detecting the correct animal (cat or dog) in new unseen images.

The objectives of this project are to complete the following tasks with Deep Learning:

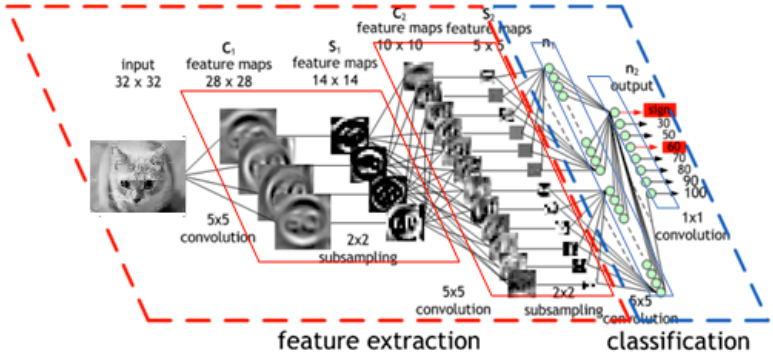
1. Dwwnload and preprocess dog/cat images from ImageNet and move them to a database.
2. Build and train a convolutional neural network (CNN) using Caffe for classifying images. Here is an intuitive explanation of the CNN (https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/).
3. Evaluate the classification performance of a trained CNN under different training parameter configurations.
4. Modify the network configuration to improve classification performance.
5. Visualize the features that a trained network has learned.

The intended solution for this problem is to classify new unseen images with highest possible accuracy using a convolutional neural network completely trained in the project from scratch.

Metrics

In deep learning, the final layer of a neural network used for classification can often be interpreted as a logistic regression. In this context, one can see a deep learning algorithm as multiple feature learning stages, which then pass their features into a logistic regression that classifies an input.

In the next sections of this project, we will look carefully at models we create. We will define two closely related networks in one file: the network used for training and the network used for testing. Both networks output the softmax_loss layer, which in training is used to compute the loss function and to initialize the backpropagation, while in validation this loss is simply reported. The testing network also has a second output layer, accuracy, which is used to report the accuracy on the test set.



Because each training example belongs to one class, we assign the training example to the class whose probability is the highest and generate a vector of binary values. Then, we compare the class assigned by the classifier and its true class to evaluate the accuracy. The official accuracy metric used:

$$acc = \frac{NumCorrectPredictions}{numExamples}$$

This process is repeated many thousands of times until the network converges to a stable average classification accuracy across all the training images.

Analysis

Data Exploration

For this project, a subset of the ImageNet (http://www.image-net.org/) dataset has been used. ImageNet, a largescale ontology of images built upon the backbone of the WordNet structure. ImageNet aims to populate the majority of the 80,000 synsets of WordNet with an average of 500-1000 clean and full resolution images. This will result in tens of millions of annotated images organized by the semantic hierarchy of WordNet.

The images used in this project are from two different categories, cats and dogs, downloaded from Imagenet. The training data and validation data are downloaded from ImageNet and saved as `jpg` files, and they are stored on the disk drive like:

- /home/ubuntu/Data/dog_cat_32/train
- /home/ubuntu/Data/dog_cat_32/test
- /home/ubuntu/Data/dog_cat_32/val

To minimize the training time, all of the images have been resized to 32x32 pixels. The training and validation input are described in `train.txt` and `val.txt` as text listing all the files and their labels. These files simply list the relative filename of each image tab separated from a natural number representing the class the image belongs to. For example, `train.txt` contains the following rows:

cat/cat_0_32.jpg 0 cat/cat_1000_32.jpg 0 dog/dog_9990_32.jpg 1 dog/dog_9991_32.jpg 1 ...

Exploratory Visualization

Below are sample images from both categories. The cats category includes domestic cats as well as large breeds like lions and tigers. The dog category is comprised of domestic dogs including pugs, basenji and great pyrenees. There are approximately 13,000 images in total.

	Image name	Label

Visualization of class labels

In [14]:
```
import pandas as pd

%matplotlib inline

user_cols = ['Image name','Label']
data_dir = "/home/ubuntu/Data/dog_cat_32/train/train.txt"

data = pd.read_csv(data_dir, sep = ' ', names = user_cols, header = None)
print "Data read successfully!"
```

```
Data read successfully!
```

In [15]:
```
data.shape
```

Out[15]:
```
(12904, 2)
```

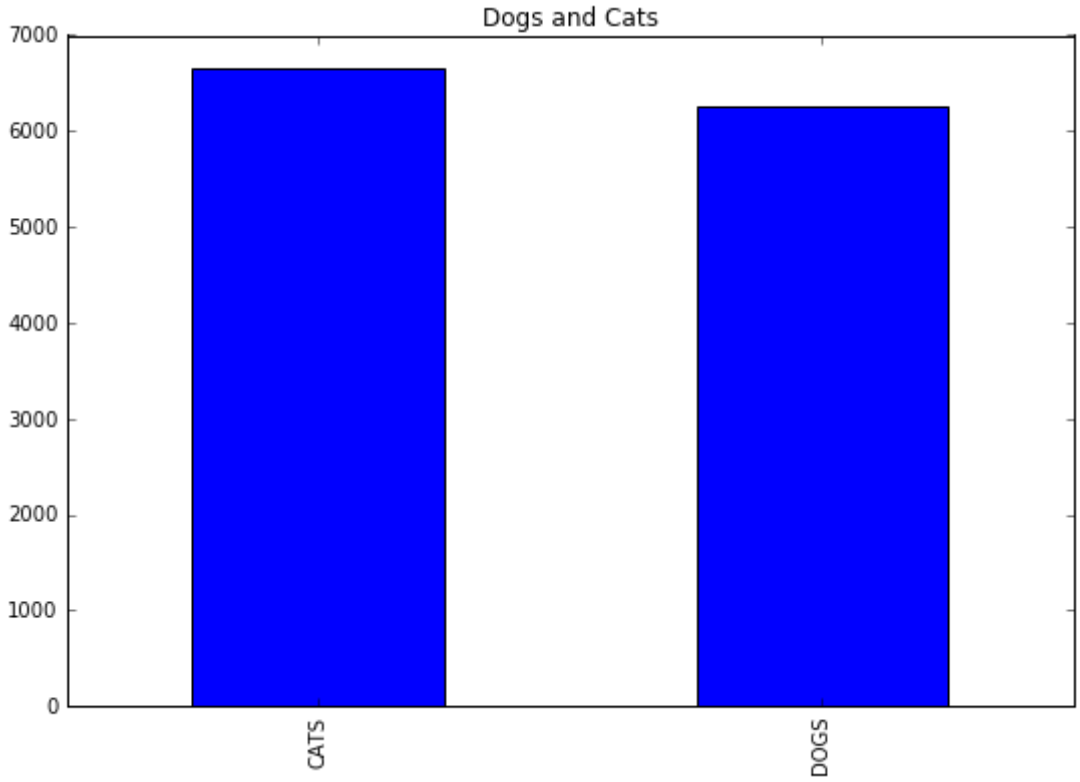
In [16]:
```
data.head()
```

Out[16]:

	Image name	Label
0	cat/cat_0_32.jpg	0
1	cat/cat_1000_32.jpg	0
2	cat/cat_1001_32.jpg	0
3	cat/cat_1002_32.jpg	0
4	cat/cat_1003_32.jpg	0

In [17]:
```
data['Label'].replace([0,1],['CATS','DOGS'],inplace=True)
data['Label'].value_counts().plot(figsize=(9,6),kind='bar', title = 'Dogs and
 Cats')
```

Out[17]: <matplotlib.axes._subplots.AxesSubplot at 0x7f3378cf3c10>



Visualization of separation between the classes

Another way to represent images in a numerical format is to convert each image to a series of RGB pixels. For example, if we had a 32x32 px image, we'd convert it to a series of 1,024 RGB pixels. You could also explore other numerical measures like hue and saturation, but let's keep it simple for now.

In [18]:
```
from PIL import Image
import numpy as np

#setup a standard image size; this will distort some images but will get every
thing into the same shape
#STANDARD_SIZE = (32, 32)
def img_to_matrix(filename, verbose=False):
    """
    takes a filename and turns it into a numpy array of RGB pixels
    """
    img = Image.open(filename)

    #No need to resize the images, they are already 32x32
    #if verbose==True:
      #  print "changing size from %s to %s" % (str(img.size), str(STANDARD_SI
ZE))
    #img = img.resize(STANDARD_SIZE)

    img = list(img.getdata())
    img = map(list, img)
    img = np.array(img)
    return img

def flatten_image(img):
    """
    takes in an (m, n) numpy array and flattens it
    into an array of shape (1, m * n)
    """
    s = img.shape[0] * img.shape[1]
    img_wide = img.reshape(1, s)
    return img_wide[0]
```

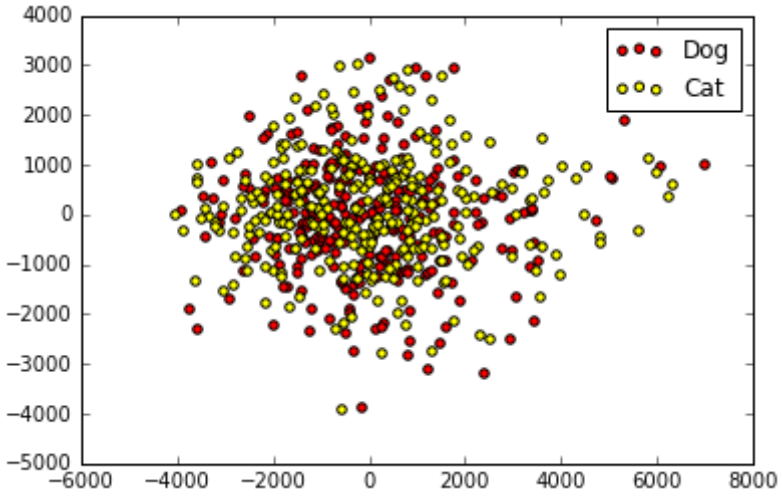
In [19]:
```
# Read images from the directory and flatten the images.
#To speed up the process we just use 100 images from each category (Dog and Ca
t)
import os
dog_img_dir = "/home/ubuntu/Data/dog_cat_32/train/dog/"
dog_images = [dog_img_dir+ f for f in os.listdir(dog_img_dir)]

cat_img_dir = "/home/ubuntu/Data/dog_cat_32/train/cat/"
cat_images = [cat_img_dir+ f for f in os.listdir(cat_img_dir)]

data = []
labels = []

for dog_image in dog_images[:300]:
    img = img_to_matrix(dog_image)
    img = flatten_image(img)
    data.append(img)
    labels.append("Dog")

for cat_image in cat_images[:300]:
    img = img_to_matrix(cat_image)
    img = flatten_image(img)
    data.append(img)
    labels.append("Cats")

data = np.array(data)
```

1,024 features is a lot to deal with for many algorithms, so we need to reduce the number of dimensions somehow. For this we can use an unsupervised learning technique called Randomized PCA to derive a smaller number of features from the raw pixel data.

Let's transform the dataset into just 2 components which we can easily plot in 2 dimensions.

Reducing the data to 2 features maintains and allows us to visualize separation between the classes.

In [20]:
```
from sklearn.decomposition import RandomizedPCA
import pylab as pl

pca = RandomizedPCA(n_components=2)
X = pca.fit_transform(data)
y = np.where(np.array(labels)=="Dog", 1, 0)

df = pd.DataFrame({"x": X[:, 0], "y": X[:, 1], "label":np.where(y==1, "Dog", "

colors = ["red", "yellow"]
for label, color in zip(df['label'].unique(), colors):
    mask = df['label']==label
    pl.scatter(df[mask]['x'], df[mask]['y'], c=color, label=label)
pl.legend()
pl.show()
```



Based on the plot above, it looks like there's not a strong split b/w Dogs and Cats images. The images are all mixed up, therefore, we won't be able to use supervised classification techniques like K-Nearest Neighbors to classify Dogs and Cats images. For this reason, we will be using Deep Learning and Convolutional Neural Networks to classify the images.

Algorithms and Techniques

We will use Python and a popular open source deep learning framework called Caffe (http://caffe.berkeleyvision.org/) to build the image classifier.

Caffe (http://caffe.berkeleyvision.org/) is a deep learning framework developed by the Berkeley Vision and Learning Center (BVLC). It is written in C++ and has Python and Matlab bindings.

- Caffe is fast due to it's highly optimized C/CUDA backend which integrates GPU acceleration
- Caffe is still very accessible due to command line, Python and Matlab interfaces
- The wrapper interfaces make it very easy to integrate DNN training and deployment into larger data analytics workflows
- Caffe has a large open-source development community adding new features all time
- Caffe has an associated model-zoo where researchers can upload trained models for others to fine tune or use for inference using their own data

Convolutional Neural Networks are a powerful artificial neural network technique.

These networks preserve the spatial structure of the problem and were developed for object recognition tasks such as handwritten digit recognition. They are popular because people are achieving state-of-the-art results on difficult computer vision and natural language processing tasks.

Given a dataset of gray scale images with the standardized size of 32×32 pixels each, a traditional feedforward neural network would require 1024 input weights (plus one bias).

This is fair enough, but the flattening of the image matrix of pixels to a long vector of pixel values loses all of the spatial structure in the image. Unless all of the images are perfectly resized, the neural network will have great difficulty with the problem.

Convolutional Neural Networks expect and preserve the spatial relationship between pixels by learning internal feature representations using small squares of input data. Feature are learned and used across the whole image, allowing for the objects in the images to be shifted or translated in the scene and still detectable by the network.

There are three types of layers in a Convolutional Neural Network:

1. Convolutional Layers.
2. Pooling Layers.
3. Fully-Connected Layers.

Convolutional Layers

Convolutional layers are comprised of filters and feature maps.

- The **filters** are the “neurons” of the layer. The have input weights and output a value.
- The **feature map** is the output of one filter applied to the previous layer.

Pooling Layers

The pooling layers down-sample the previous layers feature map.

- Pooling layers follow a sequence of one or more convolutional layers and are intended to consolidate the features learned and expressed in the previous layers feature map. As such, pooling may be consider a technique to compress or generalize feature representations and generally reduce the overfitting of the training data by the model.

Fully Connected Layers

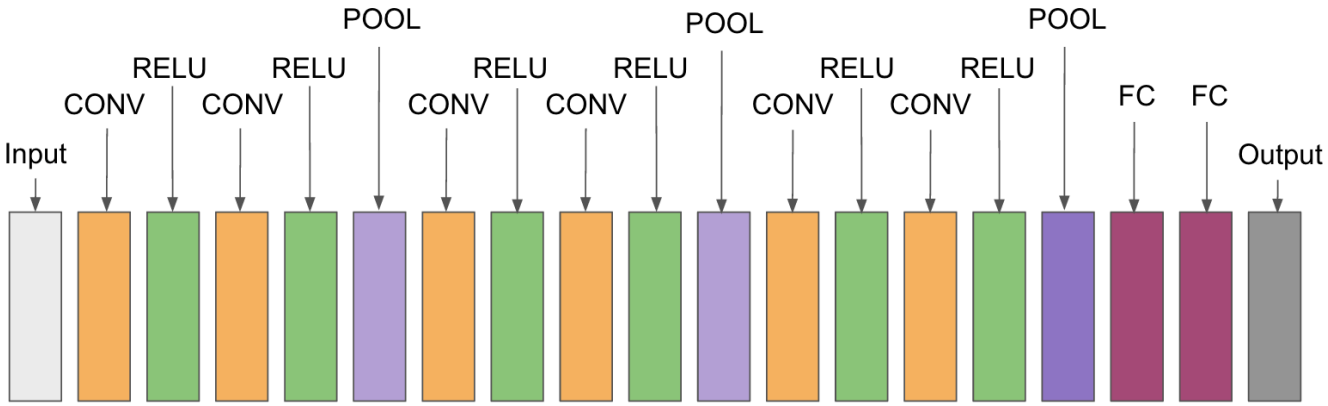
- Fully connected layers are the normal flat feed-forward neural network layer.These layers may have a non-linear activation function or a softmax activation in order to output probabilities of class predictions.

Convolutional Neural Networks Architecture

The simplest architecture of a convolutional neural networks starts with an input layer (images) followed by a sequence of convolutional layers and pooling layers, and ends with fully-connected layers. The convolutional layers are usually followed by one layer of ReLU activation functions.

The convolutional, pooling and ReLU layers act as learnable features extractors, while the fully connected layers acts as a machine learning classifier. Furthermore, the early layers of the network encode generic patterns of the images, while later layers encode the details patterns of the images.

Note that only the convolutional layers and fully-connected layers have weights. These weights are learned in the training phase.



Example of a CNN Architecture

There are 4 steps in training a CNN using Caffe:

- Step 1 - Data preparation: In this step, we clean the images and store them in a format that can be used by Caffe. We use tools that comes with Caffe to move the training and validation images into **lmdb databases**. One for train (train.lmdb) and one for validation (val.lmdb)
- Step 2 - Model definition: In this step, we choose a CNN architecture and we define its parameters in a configuration file with extension **.prototxt**.
- Step 3 - Solver definition: The solver is responsible for model optimization. We define the solver parameters in a configuration file with extension .prototxt.
- Step 4 - Model training: We train the model by executing one Caffe command from the terminal. After training the model, we will get the trained model in a file with extension **.caffemodel**.

After the training phase, we will use the .caffemodel trained model to make predictions of new unseen data. We will write Python script to do this.

Machine Setup

Images, videos, and other graphics are represented as matrices. GPUs, compared to CPUs, are faster at performing matrix operations and other advanced mathematical transformations. More detailed information regarding Hardware performance configurations can be found here (http://caffe.berkeleyvision.org/performance_hardware.html)

For this project, AWS (https://aws.amazon.com/) EC2 instance of type g2.2xlarge is used. This instance has a high-performance NVIDIA GPU with 1,536 CUDA cores and 4GB of video memory, 15GB of RAM and 8 vCPUs. The machine costs $0.65/hour.

You can skip the hassle of setting up deep learning frameworks from scratch by choosing an Amazon Machine Image (AMI) that comes pre-installed with the libraries and their dependencies. The Stanford class, CS231n: Convolutional Neural Networksfor Visual Recognition (http://cs231n.stanford.edu/), has provided a public AMI with these specs:

For step by step instruction on how to setup a AWS server with Jupyter, please follow this tutorial Start deep learning with Jupyter notebooks in the cloud. (http://efavdb.com/deep-learning-with-jupyter-on-aws/)

Let's execute the cell below to display information about the GPUs running on the server.

In [21]:
```
!nvidia-smi
```

```
Wed Sep  7 18:03:25 2016
+-----------------------------------------------------+

| NVIDIA-SMI 352.63     Driver Version: 352.63         |

|-------------------------------+----------------------+----------------------
-+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC
 |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M.
 |
|===============================+======================+======================
=|
|   0  GRID K520           Off  | 0000:00:03.0     Off |                  N/A
 |
| N/A   30C    P8    17W / 125W |    156MiB /  4095MiB |      0%      Default
 |
+-------------------------------+----------------------+----------------------
-+


+-----------------------------------------------------------------------------
-+
| Processes:                                                       GPU Memory
 |
|  GPU       PID  Type  Process name                               Usage
 |
|=============================================================================
=|
|    0      1985    C   /home/ubuntu/anaconda2/bin/python             144MiB
 |
+-----------------------------------------------------------------------------
-+
```

Benchmark

A similar problem was defined at Kaggle dogs-vs-cats (https://www.kaggle.com/c/dogs-vs-cats) competition in 2014 where contestant were asked to write an algorithm to classify whether images contain either a dog or a cat. A Deep learning expert wins Kaggle Dogs vs Cats image competition with an almost perfect result of %99.

Although the data set used in this project is from ImageNet and not from Kaggle, I use Kaggle dags vs cats competition accuracy scores as a benchmark to evaluate the performance of the models presented in the project.

Methodology

Data Preprocessing

Before we train a neural network using Caffe we will move the training and validation images into a database. The database allows Caffe to efficiently iterate over the image data during training. The training and validation datasets are independent subsets of the original image dataset. We will train the network using the training dataset and then test the networks performance using the validation dataset; that way we can be sure that the network performs well for images that it has never been trained on.

mages, videos, and other graphics are represented as matrices, and when you perform a certain operation, such as a camera rotation or a zoom in effect, all you are doing is applying some mathematical transformation to a matrix.

We complete each of these tasks using command line tools that come with Caffe. A number of useful utilities for data pre-processing, network training and network deployment can be found in the Caffe installation folder in `$CAFFE_ROOT/build/tools`

(*Note*: you can ignore the "Failed to initialize libdc1394" warning messages in the output). libdc1394 is a library for controlling camera hardware.

In [22]:
```
%%bash

#Delet database files if they are already exist.
rm -rf train_lmdb val_lmdb

#Setup environment variables
TOOLS=/home/ubuntu/caffe/build/tools
TRAIN_DATA_ROOT=/home/ubuntu/Data/dog_cat_32/train/
VAL_DATA_ROOT=/home/ubuntu/Data/dog_cat_32/val/

#Create the training database
$TOOLS/convert_imageset \
--shuffle \
$TRAIN_DATA_ROOT \
$TRAIN_DATA_ROOT/train.txt \
train_lmdb

#Create the validation database
$TOOLS/convert_imageset \
--shuffle \
$VAL_DATA_ROOT \
$VAL_DATA_ROOT/val.txt \
val_lmdb
```

```
I0907 18:03:28.444833  2063 convert_imageset.cpp:83] Shuffling data
I0907 18:03:28.580814  2063 convert_imageset.cpp:86] A total of 12904 images.
I0907 18:03:28.581188  2063 db_lmdb.cpp:38] Opened lmdb train_lmdb
I0907 18:03:28.667840  2063 convert_imageset.cpp:144] Processed 1000 files.
I0907 18:03:28.750430  2063 convert_imageset.cpp:144] Processed 2000 files.
I0907 18:03:28.839179  2063 convert_imageset.cpp:144] Processed 3000 files.
I0907 18:03:28.921123  2063 convert_imageset.cpp:144] Processed 4000 files.
I0907 18:03:29.011930  2063 convert_imageset.cpp:144] Processed 5000 files.
I0907 18:03:29.093874  2063 convert_imageset.cpp:144] Processed 6000 files.
I0907 18:03:29.176609  2063 convert_imageset.cpp:144] Processed 7000 files.
I0907 18:03:29.267001  2063 convert_imageset.cpp:144] Processed 8000 files.
I0907 18:03:29.353193  2063 convert_imageset.cpp:144] Processed 9000 files.
I0907 18:03:29.436241  2063 convert_imageset.cpp:144] Processed 10000 files.
I0907 18:03:29.519358  2063 convert_imageset.cpp:144] Processed 11000 files.
I0907 18:03:29.603425  2063 convert_imageset.cpp:144] Processed 12000 files.
I0907 18:03:29.685639  2063 convert_imageset.cpp:150] Processed 12904 files.
I0907 18:03:29.953532  2066 convert_imageset.cpp:83] Shuffling data
I0907 18:03:30.081560  2066 convert_imageset.cpp:86] A total of 1138 images.
I0907 18:03:30.081914  2066 db_lmdb.cpp:38] Opened lmdb val_lmdb
I0907 18:03:30.168824  2066 convert_imageset.cpp:144] Processed 1000 files.
I0907 18:03:30.186359  2066 convert_imageset.cpp:150] Processed 1138 files.
```

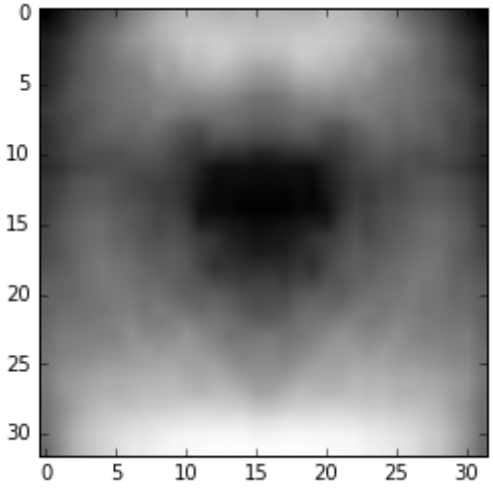
We also create a mean image from the training data. This is the image obtained by taking the mean value of each pixel across all of the training dataset images. We do this so that we can extract that mean image from each training and validation image before it is fed into the neural network. This is an important pre-processing and refinement step for achieving fast and effective training. It has the effect of removing the average brightness (intensity) of each point in the image so that the network learns about image content rather than illumination conditions.

Execute the cell below to create a mean image of the training data.

In [23]:
```
%%bash

#Setup environment variables
TOOLS=/home/ubuntu/caffe/build/tools

#Create the mean image database
$TOOLS/compute_image_mean train_lmdb mean.binaryproto
```

Run the cell below to see what the mean image looks like. Strangely, it looks a little like a mouse...

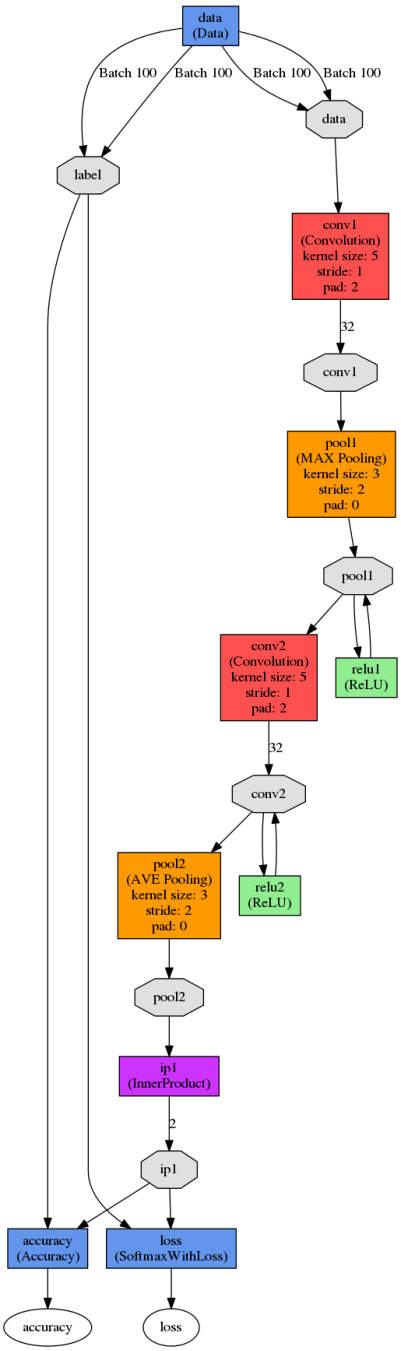
In [24]:
```
import caffe
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.cm as cm
%matplotlib inline

blob = caffe.proto.caffe_pb2.BlobProto()
data = open('/home/ubuntu/caffe/examples/Dog_Cat_Classification/mean.binarypro
to','rb').read()
blob.ParseFromString(data)
arr = np.array(caffe.io.blobproto_to_array(blob))[0,:,:,:].mean(0)
plt.imshow(arr, cmap=cm.Greys_r)
plt.show()
```



Implementation

We are now going to configure our network and start training. For this project we will borrow our network architecture from the cifar10 design that is provided with Caffe. You can find the original network in `$CAFFE_ROOT/examples/cifar10`. The only modification we will make initially is to change the final output softmax layer, which does the actual classification, to only have two classes for our dogs and cats problem rather than the ten classes needed for the CIFAR data.

Below we see an image of the cifar10 network architecture. As we can see the cifar10 network is a convolutional neural network (CNN) with three convolutional layers (each with pooling, ReLU activation and normalization) followed by a single fully-connected layer performing the final classification.


data
(Data)
Batch 100
Batch 100
Batch 100
Batch 100
data
label
conv1
(Convolution)
kernel size: 5
stride: 1
pad: 2
32
conv1
pool1
(MAX Pooling)
kernel size: 3
stride: 2
pad: 0
pool1
conv2
(Convolution)
kernel size: 5
stride: 1
pad: 2
relu1
(ReLU)
32
conv2
pool2
(AVE Pooling)
kernel size: 3
stride: 2
pad: 0
relu2
(ReLU)
pool2
ip1
(InnerProduct)
2
ip1
accuracy
(Accuracy)
loss
(SoftmaxWithLoss)
accuracy
loss

We see that the network input data has dimensions 100x3x32x32. This means that we have batches of 100 training images each with three-channels, i.e. it's color, and 32x32 pixels.

The first convolutional layer applies 32 5x5 filters. As the filters have size 5x5 we must pad the edges of the input images with zeros to ensure that the output of the layer has the same size.

After the convolutional layer max pooling is applied to reduce the size of the output images by half and a rectified linear unit (ReLU) activation function is applied.

Layers 2 is also a convolutional layer repeating this pattern but using average pooling.

The final layer is a fully connected layer with two neurons which together with the image labels feeds into a softmax layer which performs the classification and computes the classification loss. The two final neurons in the output layer correspond to our two classes, dogs and cats.

Caffe encodes deep neural network architectures like this in text files called prototxt files. You can get more information about the types of layers that can be defined in a network prototxt file here (http://caffe.berkeleyvision.org/tutorial/layers.html).

We now have our datasets and a network configuration we are nearly ready to train the network. The final thing that Caffe needs before we can train is a specification of the learning algorithm parameters. This specification is also made in a prototxt file but with a simpler structure. You can get more information about the range of parameters that can be set in a solver prototxt file here (http://caffe.berkeleyvision.org/tutorial/solver.html).

The network configuration and solver files are located at src directory.

We are now ready to train the network. Again, this is carried out using a command line tool that comes with Caffe, this time it is the binary `caffe` itself with the `train` option. Execute the cell below to begin training - it should take just under a minute to train - be sure to scroll down through the complete Caffe output.

In [8]:
```
%%bash
#Set the location of the caffe tools folder
TOOLS=/home/ubuntu/caffe/build/tools
#Train the network
$TOOLS/caffe train -gpu 0 -solver src/solver.prototxt
```

```
I0907 17:49:10.599663  2018 upgrade_proto.cpp:990] Attempting to upgrade inpu
t file specified using deprecated 'solver_type' field (enum)': src/solver.pro
totxt
I0907 17:49:10.599895  2018 upgrade_proto.cpp:997] Successfully upgraded file
 specified using deprecated 'solver_type' field (enum) to 'type' field (strin
g).
W0907 17:49:10.599907  2018 upgrade_proto.cpp:999] Note that future Caffe rel
eases will only support 'type' field (string) for a solver's type.
I0907 17:49:10.600075  2018 caffe.cpp:185] Using GPUs 0
I0907 17:49:10.734774  2018 solver.cpp:48] Initializing solver from parameter
s:
test_iter: 100
test_interval: 250
base_lr: 0.01
display: 20
max_iter: 750
lr_policy: "step"
gamma: 0.1
momentum: 0.9
weight_decay: 0.0005
stepsize: 500
snapshot: 250
snapshot_prefix: "checkpoints/snapshot"
solver_mode: GPU
device_id: 0
random_seed: 1234
net: "src/train_val.prototxt"
type: "SGD"
I0907 17:49:10.734944  2018 solver.cpp:91] Creating training net from net fil
e: src/train_val.prototxt
I0907 17:49:10.735399  2018 net.cpp:322] The NetState phase (0) differed from
 the phase (1) specified by a rule in layer data
I0907 17:49:10.735427  2018 net.cpp:322] The NetState phase (0) differed from
 the phase (1) specified by a rule in layer accuracy
I0907 17:49:10.735538  2018 net.cpp:49] Initializing net from parameters:
state {
  phase: TRAIN
}
layer {
  name: "data"
  type: "Data"
  top: "data"
  top: "label"
  include {
    phase: TRAIN
  }
  transform_param {
    mirror: true
    crop_size: 32
    mean_file: "mean.binaryproto"
  }
  data_param {
    source: "train_lmdb"
    batch_size: 100
    backend: LMDB
  }
}
```

```
layer {
  name: "conv1"
  type: "Convolution"
  bottom: "data"
  top: "conv1"
  param {
    lr_mult: 1
  }
  param {
    lr_mult: 2
  }
  convolution_param {
    num_output: 32
    pad: 2
    kernel_size: 5
    stride: 1
    weight_filler {
      type: "gaussian"
      std: 0.0001
    }
    bias_filler {
      type: "constant"
    }
  }
}
layer {
  name: "pool1"
  type: "Pooling"
  bottom: "conv1"
  top: "pool1"
  pooling_param {
    pool: MAX
    kernel_size: 3
    stride: 2
  }
}
layer {
  name: "relu1"
  type: "ReLU"
  bottom: "pool1"
  top: "pool1"
}
layer {
  name: "conv2"
  type: "Convolution"
  bottom: "pool1"
  top: "conv2"
  param {
    lr_mult: 1
  }
  param {
    lr_mult: 2
  }
  convolution_param {
    num_output: 32
    pad: 2
    kernel_size: 5
```

```
    stride: 1
    weight_filler {
      type: "gaussian"
      std: 0.01
    }
    bias_filler {
      type: "constant"
    }
  }
}
layer {
  name: "relu2"
  type: "ReLU"
  bottom: "conv2"
  top: "conv2"
}
layer {
  name: "pool2"
  type: "Pooling"
  bottom: "conv2"
  top: "pool2"
  pooling_param {
    pool: AVE
    kernel_size: 3
    stride: 2
  }
}
layer {
  name: "ip1"
  type: "InnerProduct"
  bottom: "pool2"
  top: "ip1"
  param {
    lr_mult: 1
    decay_mult: 250
  }
  param {
    lr_mult: 2
    decay_mult: 0
  }
  inner_product_param {
    num_output: 2
    weight_filler {
      type: "gaussian"
      std: 0.01
    }
    bias_filler {
      type: "constant"
    }
  }
}
layer {
  name: "loss"
  type: "SoftmaxWithLoss"
  bottom: "ip1"
  bottom: "label"
  top: "loss"
```

```
}
I0907 17:49:10.735671  2018 layer_factory.hpp:77] Creating layer data
I0907 17:49:10.735867  2018 net.cpp:106] Creating Layer data
I0907 17:49:10.735889  2018 net.cpp:411] data -> data
I0907 17:49:10.735931  2018 net.cpp:411] data -> label
I0907 17:49:10.735960  2018 data_transformer.cpp:25] Loading mean file from:
 mean.binaryproto
I0907 17:49:10.736557  2021 db_lmdb.cpp:38] Opened lmdb train_lmdb
I0907 17:49:10.747678  2018 data_layer.cpp:41] output data size: 100,3,32,32
I0907 17:49:10.750541  2018 net.cpp:150] Setting up data
I0907 17:49:10.750619  2018 net.cpp:157] Top shape: 100 3 32 32 (307200)
I0907 17:49:10.750636  2018 net.cpp:157] Top shape: 100 (100)
I0907 17:49:10.750641  2018 net.cpp:165] Memory required for data: 1229200
I0907 17:49:10.750655  2018 layer_factory.hpp:77] Creating layer conv1
I0907 17:49:10.750687  2018 net.cpp:106] Creating Layer conv1
I0907 17:49:10.750704  2018 net.cpp:454] conv1 <- data
I0907 17:49:10.750721  2018 net.cpp:411] conv1 -> conv1
I0907 17:49:10.869427  2018 net.cpp:150] Setting up conv1
I0907 17:49:10.869477  2018 net.cpp:157] Top shape: 100 32 32 32 (3276800)
I0907 17:49:10.869514  2018 net.cpp:165] Memory required for data: 14336400
I0907 17:49:10.869545  2018 layer_factory.hpp:77] Creating layer pool1
I0907 17:49:10.869575  2018 net.cpp:106] Creating Layer pool1
I0907 17:49:10.869588  2018 net.cpp:454] pool1 <- conv1
I0907 17:49:10.869598  2018 net.cpp:411] pool1 -> pool1
I0907 17:49:10.869664  2018 net.cpp:150] Setting up pool1
I0907 17:49:10.869681  2018 net.cpp:157] Top shape: 100 32 16 16 (819200)
I0907 17:49:10.869686  2018 net.cpp:165] Memory required for data: 17613200
I0907 17:49:10.869701  2018 layer_factory.hpp:77] Creating layer relu1
I0907 17:49:10.869710  2018 net.cpp:106] Creating Layer relu1
I0907 17:49:10.869720  2018 net.cpp:454] relu1 <- pool1
I0907 17:49:10.869729  2018 net.cpp:397] relu1 -> pool1 (in-place)
I0907 17:49:10.869884  2018 net.cpp:150] Setting up relu1
I0907 17:49:10.869904  2018 net.cpp:157] Top shape: 100 32 16 16 (819200)
I0907 17:49:10.869909  2018 net.cpp:165] Memory required for data: 20890000
I0907 17:49:10.869915  2018 layer_factory.hpp:77] Creating layer conv2
I0907 17:49:10.869933  2018 net.cpp:106] Creating Layer conv2
I0907 17:49:10.869943  2018 net.cpp:454] conv2 <- pool1
I0907 17:49:10.869952  2018 net.cpp:411] conv2 -> conv2
I0907 17:49:10.872194  2018 net.cpp:150] Setting up conv2
I0907 17:49:10.872220  2018 net.cpp:157] Top shape: 100 32 16 16 (819200)
I0907 17:49:10.872225  2018 net.cpp:165] Memory required for data: 24166800
I0907 17:49:10.872241  2018 layer_factory.hpp:77] Creating layer relu2
I0907 17:49:10.872254  2018 net.cpp:106] Creating Layer relu2
I0907 17:49:10.872261  2018 net.cpp:454] relu2 <- conv2
I0907 17:49:10.872272  2018 net.cpp:397] relu2 -> conv2 (in-place)
I0907 17:49:10.872520  2018 net.cpp:150] Setting up relu2
I0907 17:49:10.872550  2018 net.cpp:157] Top shape: 100 32 16 16 (819200)
I0907 17:49:10.872555  2018 net.cpp:165] Memory required for data: 27443600
I0907 17:49:10.872566  2018 layer_factory.hpp:77] Creating layer pool2
I0907 17:49:10.872580  2018 net.cpp:106] Creating Layer pool2
I0907 17:49:10.872591  2018 net.cpp:454] pool2 <- conv2
I0907 17:49:10.872609  2018 net.cpp:411] pool2 -> pool2
I0907 17:49:10.872788  2018 net.cpp:150] Setting up pool2
I0907 17:49:10.872807  2018 net.cpp:157] Top shape: 100 32 8 8 (204800)
I0907 17:49:10.872812  2018 net.cpp:165] Memory required for data: 28262800
I0907 17:49:10.872819  2018 layer_factory.hpp:77] Creating layer ip1
I0907 17:49:10.872833  2018 net.cpp:106] Creating Layer ip1
```

```
I0907 17:49:10.872844  2018 net.cpp:454] ip1 <- pool2
I0907 17:49:10.872856  2018 net.cpp:411] ip1 -> ip1
I0907 17:49:10.873124  2018 net.cpp:150] Setting up ip1
I0907 17:49:10.873142  2018 net.cpp:157] Top shape: 100 2 (200)
I0907 17:49:10.873147  2018 net.cpp:165] Memory required for data: 28263600
I0907 17:49:10.873160  2018 layer_factory.hpp:77] Creating layer loss
I0907 17:49:10.873183  2018 net.cpp:106] Creating Layer loss
I0907 17:49:10.873194  2018 net.cpp:454] loss <- ip1
I0907 17:49:10.873201  2018 net.cpp:454] loss <- label
I0907 17:49:10.873214  2018 net.cpp:411] loss -> loss
I0907 17:49:10.873237  2018 layer_factory.hpp:77] Creating layer loss
I0907 17:49:10.873580  2018 net.cpp:150] Setting up loss
I0907 17:49:10.873600  2018 net.cpp:157] Top shape: (1)
I0907 17:49:10.873606  2018 net.cpp:160]     with loss weight 1
I0907 17:49:10.873637  2018 net.cpp:165] Memory required for data: 28263604
I0907 17:49:10.873643  2018 net.cpp:226] loss needs backward computation.
I0907 17:49:10.873649  2018 net.cpp:226] ip1 needs backward computation.
I0907 17:49:10.873656  2018 net.cpp:226] pool2 needs backward computation.
I0907 17:49:10.873659  2018 net.cpp:226] relu2 needs backward computation.
I0907 17:49:10.873666  2018 net.cpp:226] conv2 needs backward computation.
I0907 17:49:10.873669  2018 net.cpp:226] relu1 needs backward computation.
I0907 17:49:10.873675  2018 net.cpp:226] pool1 needs backward computation.
I0907 17:49:10.873679  2018 net.cpp:226] conv1 needs backward computation.
I0907 17:49:10.873685  2018 net.cpp:228] data does not need backward computat
ion.
I0907 17:49:10.873709  2018 net.cpp:270] This network produces output loss
I0907 17:49:10.873724  2018 net.cpp:283] Network initialization done.
I0907 17:49:10.874142  2018 solver.cpp:181] Creating test net (#0) specified
 by net file: src/train_val.prototxt
I0907 17:49:10.874191  2018 net.cpp:322] The NetState phase (1) differed from
 the phase (0) specified by a rule in layer data
I0907 17:49:10.874311  2018 net.cpp:49] Initializing net from parameters:
state {
  phase: TEST
}
layer {
  name: "data"
  type: "Data"
  top: "data"
  top: "label"
  include {
    phase: TEST
  }
  transform_param {
    crop_size: 32
    mean_file: "mean.binaryproto"
  }
  data_param {
    source: "val_lmdb"
    batch_size: 100
    backend: LMDB
  }
}
layer {
  name: "conv1"
  type: "Convolution"
  bottom: "data"
```

```
    top: "conv1"
    param {
      lr_mult: 1
    }
    param {
      lr_mult: 2
    }
    convolution_param {
      num_output: 32
      pad: 2
      kernel_size: 5
      stride: 1
      weight_filler {
        type: "gaussian"
        std: 0.0001
      }
      bias_filler {
        type: "constant"
      }
    }
  }
  layer {
    name: "pool1"
    type: "Pooling"
    bottom: "conv1"
    top: "pool1"
    pooling_param {
      pool: MAX
      kernel_size: 3
      stride: 2
    }
  }
  layer {
    name: "relu1"
    type: "ReLU"
    bottom: "pool1"
    top: "pool1"
  }
  layer {
    name: "conv2"
    type: "Convolution"
    bottom: "pool1"
    top: "conv2"
    param {
      lr_mult: 1
    }
    param {
      lr_mult: 2
    }
    convolution_param {
      num_output: 32
      pad: 2
      kernel_size: 5
      stride: 1
      weight_filler {
        type: "gaussian"
        std: 0.01
```

```
    }
    bias_filler {
      type: "constant"
    }
  }
}
layer {
  name: "relu2"
  type: "ReLU"
  bottom: "conv2"
  top: "conv2"
}
layer {
  name: "pool2"
  type: "Pooling"
  bottom: "conv2"
  top: "pool2"
  pooling_param {
    pool: AVE
    kernel_size: 3
    stride: 2
  }
}
layer {
  name: "ip1"
  type: "InnerProduct"
  bottom: "pool2"
  top: "ip1"
  param {
    lr_mult: 1
    decay_mult: 250
  }
  param {
    lr_mult: 2
    decay_mult: 0
  }
  inner_product_param {
    num_output: 2
    weight_filler {
      type: "gaussian"
      std: 0.01
    }
    bias_filler {
      type: "constant"
    }
  }
}
layer {
  name: "loss"
  type: "SoftmaxWithLoss"
  bottom: "ip1"
  bottom: "label"
  top: "loss"
}
layer {
  name: "accuracy"
  type: "Accuracy"
```

```
  bottom: "ip1"
  bottom: "label"
  top: "accuracy"
  include {
    phase: TEST
  }
}
I0907 17:49:10.874408  2018 layer_factory.hpp:77] Creating layer data
I0907 17:49:10.874550  2018 net.cpp:106] Creating Layer data
I0907 17:49:10.874567  2018 net.cpp:411] data -> data
I0907 17:49:10.874579  2018 net.cpp:411] data -> label
I0907 17:49:10.874595  2018 data_transformer.cpp:25] Loading mean file from:
 mean.binaryproto
I0907 17:49:10.875272  2023 db_lmdb.cpp:38] Opened lmdb val_lmdb
I0907 17:49:10.875392  2018 data_layer.cpp:41] output data size: 100,3,32,32
I0907 17:49:10.878393  2018 net.cpp:150] Setting up data
I0907 17:49:10.878414  2018 net.cpp:157] Top shape: 100 3 32 32 (307200)
I0907 17:49:10.878422  2018 net.cpp:157] Top shape: 100 (100)
I0907 17:49:10.878433  2018 net.cpp:165] Memory required for data: 1229200
I0907 17:49:10.878439  2018 layer_factory.hpp:77] Creating layer label_data_1
_split
I0907 17:49:10.878448  2018 net.cpp:106] Creating Layer label_data_1_split
I0907 17:49:10.878459  2018 net.cpp:454] label_data_1_split <- label
I0907 17:49:10.878466  2018 net.cpp:411] label_data_1_split -> label_data_1_s
plit_0
I0907 17:49:10.878481  2018 net.cpp:411] label_data_1_split -> label_data_1_s
plit_1
I0907 17:49:10.878564  2018 net.cpp:150] Setting up label_data_1_split
I0907 17:49:10.878581  2018 net.cpp:157] Top shape: 100 (100)
I0907 17:49:10.878587  2018 net.cpp:157] Top shape: 100 (100)
I0907 17:49:10.878597  2018 net.cpp:165] Memory required for data: 1230000
I0907 17:49:10.878602  2018 layer_factory.hpp:77] Creating layer conv1
I0907 17:49:10.878646  2018 net.cpp:106] Creating Layer conv1
I0907 17:49:10.878660  2018 net.cpp:454] conv1 <- data
I0907 17:49:10.878669  2018 net.cpp:411] conv1 -> conv1
I0907 17:49:10.879997  2018 net.cpp:150] Setting up conv1
I0907 17:49:10.880022  2018 net.cpp:157] Top shape: 100 32 32 32 (3276800)
I0907 17:49:10.880028  2018 net.cpp:165] Memory required for data: 14337200
I0907 17:49:10.880043  2018 layer_factory.hpp:77] Creating layer pool1
I0907 17:49:10.880056  2018 net.cpp:106] Creating Layer pool1
I0907 17:49:10.880069  2018 net.cpp:454] pool1 <- conv1
I0907 17:49:10.880076  2018 net.cpp:411] pool1 -> pool1
I0907 17:49:10.880125  2018 net.cpp:150] Setting up pool1
I0907 17:49:10.880141  2018 net.cpp:157] Top shape: 100 32 16 16 (819200)
I0907 17:49:10.880146  2018 net.cpp:165] Memory required for data: 17614000
I0907 17:49:10.880153  2018 layer_factory.hpp:77] Creating layer relu1
I0907 17:49:10.880162  2018 net.cpp:106] Creating Layer relu1
I0907 17:49:10.880172  2018 net.cpp:454] relu1 <- pool1
I0907 17:49:10.880178  2018 net.cpp:397] relu1 -> pool1 (in-place)
I0907 17:49:10.880343  2018 net.cpp:150] Setting up relu1
I0907 17:49:10.880365  2018 net.cpp:157] Top shape: 100 32 16 16 (819200)
I0907 17:49:10.880370  2018 net.cpp:165] Memory required for data: 20890800
I0907 17:49:10.880378  2018 layer_factory.hpp:77] Creating layer conv2
I0907 17:49:10.880393  2018 net.cpp:106] Creating Layer conv2
I0907 17:49:10.880404  2018 net.cpp:454] conv2 <- pool1
I0907 17:49:10.880422  2018 net.cpp:411] conv2 -> conv2
I0907 17:49:10.882477  2018 net.cpp:150] Setting up conv2
```

```
I0907 17:49:10.882499  2018 net.cpp:157] Top shape: 100 32 16 16 (819200)
I0907 17:49:10.882505  2018 net.cpp:165] Memory required for data: 24167600
I0907 17:49:10.882519  2018 layer_factory.hpp:77] Creating layer relu2
I0907 17:49:10.882536  2018 net.cpp:106] Creating Layer relu2
I0907 17:49:10.882545  2018 net.cpp:454] relu2 <- conv2
I0907 17:49:10.882552  2018 net.cpp:397] relu2 -> conv2 (in-place)
I0907 17:49:10.882721  2018 net.cpp:150] Setting up relu2
I0907 17:49:10.882740  2018 net.cpp:157] Top shape: 100 32 16 16 (819200)
I0907 17:49:10.882745  2018 net.cpp:165] Memory required for data: 27444400
I0907 17:49:10.882752  2018 layer_factory.hpp:77] Creating layer pool2
I0907 17:49:10.882761  2018 net.cpp:106] Creating Layer pool2
I0907 17:49:10.882771  2018 net.cpp:454] pool2 <- conv2
I0907 17:49:10.882781  2018 net.cpp:411] pool2 -> pool2
I0907 17:49:10.883066  2018 net.cpp:150] Setting up pool2
I0907 17:49:10.883086  2018 net.cpp:157] Top shape: 100 32 8 8 (204800)
I0907 17:49:10.883091  2018 net.cpp:165] Memory required for data: 28263600
I0907 17:49:10.883098  2018 layer_factory.hpp:77] Creating layer ip1
I0907 17:49:10.883110  2018 net.cpp:106] Creating Layer ip1
I0907 17:49:10.883119  2018 net.cpp:454] ip1 <- pool2
I0907 17:49:10.883134  2018 net.cpp:411] ip1 -> ip1
I0907 17:49:10.883389  2018 net.cpp:150] Setting up ip1
I0907 17:49:10.883407  2018 net.cpp:157] Top shape: 100 2 (200)
I0907 17:49:10.883412  2018 net.cpp:165] Memory required for data: 28264400
I0907 17:49:10.883425  2018 layer_factory.hpp:77] Creating layer ip1_ip1_0_sp
lit
I0907 17:49:10.883440  2018 net.cpp:106] Creating Layer ip1_ip1_0_split
I0907 17:49:10.883445  2018 net.cpp:454] ip1_ip1_0_split <- ip1
I0907 17:49:10.883457  2018 net.cpp:411] ip1_ip1_0_split -> ip1_ip1_0_split_0
I0907 17:49:10.883471  2018 net.cpp:411] ip1_ip1_0_split -> ip1_ip1_0_split_1
I0907 17:49:10.883517  2018 net.cpp:150] Setting up ip1_ip1_0_split
I0907 17:49:10.883533  2018 net.cpp:157] Top shape: 100 2 (200)
I0907 17:49:10.883539  2018 net.cpp:157] Top shape: 100 2 (200)
I0907 17:49:10.883544  2018 net.cpp:165] Memory required for data: 28266000
I0907 17:49:10.883550  2018 layer_factory.hpp:77] Creating layer loss
I0907 17:49:10.883558  2018 net.cpp:106] Creating Layer loss
I0907 17:49:10.883569  2018 net.cpp:454] loss <- ip1_ip1_0_split_0
I0907 17:49:10.883575  2018 net.cpp:454] loss <- label_data_1_split_0
I0907 17:49:10.883592  2018 net.cpp:411] loss -> loss
I0907 17:49:10.883626  2018 layer_factory.hpp:77] Creating layer loss
I0907 17:49:10.883978  2018 net.cpp:150] Setting up loss
I0907 17:49:10.884004  2018 net.cpp:157] Top shape: (1)
I0907 17:49:10.884016  2018 net.cpp:160]     with loss weight 1
I0907 17:49:10.884065  2018 net.cpp:165] Memory required for data: 28266004
I0907 17:49:10.884078  2018 layer_factory.hpp:77] Creating layer accuracy
I0907 17:49:10.884096  2018 net.cpp:106] Creating Layer accuracy
I0907 17:49:10.884116  2018 net.cpp:454] accuracy <- ip1_ip1_0_split_1
I0907 17:49:10.884130  2018 net.cpp:454] accuracy <- label_data_1_split_1
I0907 17:49:10.884152  2018 net.cpp:411] accuracy -> accuracy
I0907 17:49:10.884187  2018 net.cpp:150] Setting up accuracy
I0907 17:49:10.884209  2018 net.cpp:157] Top shape: (1)
I0907 17:49:10.884219  2018 net.cpp:165] Memory required for data: 28266008
I0907 17:49:10.884232  2018 net.cpp:228] accuracy does not need backward comp
utation.
I0907 17:49:10.884244  2018 net.cpp:226] loss needs backward computation.
I0907 17:49:10.884263  2018 net.cpp:226] ip1_ip1_0_split needs backward compu
tation.
I0907 17:49:10.884274  2018 net.cpp:226] ip1 needs backward computation.
```

```
I0907 17:49:10.884290  2018 net.cpp:226] pool2 needs backward computation.
I0907 17:49:10.884300  2018 net.cpp:226] relu2 needs backward computation.
I0907 17:49:10.884312  2018 net.cpp:226] conv2 needs backward computation.
I0907 17:49:10.884322  2018 net.cpp:226] relu1 needs backward computation.
I0907 17:49:10.884340  2018 net.cpp:226] pool1 needs backward computation.
I0907 17:49:10.884349  2018 net.cpp:226] conv1 needs backward computation.
I0907 17:49:10.884364  2018 net.cpp:228] label_data_1_split does not need bac
kward computation.
I0907 17:49:10.884374  2018 net.cpp:228] data does not need backward computat
ion.
I0907 17:49:10.884390  2018 net.cpp:270] This network produces output accurac
y
I0907 17:49:10.884402  2018 net.cpp:270] This network produces output loss
I0907 17:49:10.884426  2018 net.cpp:283] Network initialization done.
I0907 17:49:10.884502  2018 solver.cpp:60] Solver scaffolding done.
I0907 17:49:10.884750  2018 caffe.cpp:213] Starting Optimization
I0907 17:49:10.884773  2018 solver.cpp:280] Solving
I0907 17:49:10.884781  2018 solver.cpp:281] Learning Rate Policy: step
I0907 17:49:10.885170  2018 solver.cpp:338] Iteration 0, Testing net (#0)
I0907 17:49:11.375325  2018 solver.cpp:406]     Test net output #0: accuracy
 = 0.5206
I0907 17:49:11.375389  2018 solver.cpp:406]     Test net output #1: loss = 0.
693017 (* 1 = 0.693017 loss)
I0907 17:49:11.381693  2018 solver.cpp:229] Iteration 0, loss = 0.693084
I0907 17:49:11.381729  2018 solver.cpp:245]     Train net output #0: loss =
 0.693084 (* 1 = 0.693084 loss)
I0907 17:49:11.381758  2018 sgd_solver.cpp:106] Iteration 0, lr = 0.01
I0907 17:49:11.658129  2018 solver.cpp:229] Iteration 20, loss = 0.696418
I0907 17:49:11.658182  2018 solver.cpp:245]     Train net output #0: loss =
 0.696418 (* 1 = 0.696418 loss)
I0907 17:49:11.658197  2018 sgd_solver.cpp:106] Iteration 20, lr = 0.01
I0907 17:49:11.934586  2018 solver.cpp:229] Iteration 40, loss = 0.702814
I0907 17:49:11.934641  2018 solver.cpp:245]     Train net output #0: loss =
 0.702814 (* 1 = 0.702814 loss)
I0907 17:49:11.934656  2018 sgd_solver.cpp:106] Iteration 40, lr = 0.01
I0907 17:49:12.211274  2018 solver.cpp:229] Iteration 60, loss = 0.693217
I0907 17:49:12.211331  2018 solver.cpp:245]     Train net output #0: loss =
 0.693217 (* 1 = 0.693217 loss)
I0907 17:49:12.211347  2018 sgd_solver.cpp:106] Iteration 60, lr = 0.01
I0907 17:49:12.487753  2018 solver.cpp:229] Iteration 80, loss = 0.692173
I0907 17:49:12.487807  2018 solver.cpp:245]     Train net output #0: loss =
 0.692173 (* 1 = 0.692173 loss)
I0907 17:49:12.487823  2018 sgd_solver.cpp:106] Iteration 80, lr = 0.01
I0907 17:49:12.764281  2018 solver.cpp:229] Iteration 100, loss = 0.693186
I0907 17:49:12.764334  2018 solver.cpp:245]     Train net output #0: loss =
 0.693186 (* 1 = 0.693186 loss)
I0907 17:49:12.764351  2018 sgd_solver.cpp:106] Iteration 100, lr = 0.01
I0907 17:49:13.040750  2018 solver.cpp:229] Iteration 120, loss = 0.693745
I0907 17:49:13.040812  2018 solver.cpp:245]     Train net output #0: loss =
 0.693745 (* 1 = 0.693745 loss)
I0907 17:49:13.040828  2018 sgd_solver.cpp:106] Iteration 120, lr = 0.01
I0907 17:49:13.318050  2018 solver.cpp:229] Iteration 140, loss = 0.688764
I0907 17:49:13.318100  2018 solver.cpp:245]     Train net output #0: loss =
 0.688764 (* 1 = 0.688764 loss)
I0907 17:49:13.318116  2018 sgd_solver.cpp:106] Iteration 140, lr = 0.01
I0907 17:49:13.595294  2018 solver.cpp:229] Iteration 160, loss = 0.694766
I0907 17:49:13.595350  2018 solver.cpp:245]     Train net output #0: loss =
```

```
 0.694766 (* 1 = 0.694766 loss)
I0907 17:49:13.595366  2018 sgd_solver.cpp:106] Iteration 160, lr = 0.01
I0907 17:49:13.872601  2018 solver.cpp:229] Iteration 180, loss = 0.68917
I0907 17:49:13.872658  2018 solver.cpp:245]     Train net output #0: loss =
 0.68917 (* 1 = 0.68917 loss)
I0907 17:49:13.872673  2018 sgd_solver.cpp:106] Iteration 180, lr = 0.01
I0907 17:49:14.149952  2018 solver.cpp:229] Iteration 200, loss = 0.694095
I0907 17:49:14.150007  2018 solver.cpp:245]     Train net output #0: loss =
 0.694095 (* 1 = 0.694095 loss)
I0907 17:49:14.150022  2018 sgd_solver.cpp:106] Iteration 200, lr = 0.01
I0907 17:49:14.427142  2018 solver.cpp:229] Iteration 220, loss = 0.694301
I0907 17:49:14.427196  2018 solver.cpp:245]     Train net output #0: loss =
 0.694301 (* 1 = 0.694301 loss)
I0907 17:49:14.427211  2018 sgd_solver.cpp:106] Iteration 220, lr = 0.01
I0907 17:49:14.704120  2018 solver.cpp:229] Iteration 240, loss = 0.691377
I0907 17:49:14.704172  2018 solver.cpp:245]     Train net output #0: loss =
 0.691377 (* 1 = 0.691377 loss)
I0907 17:49:14.704187  2018 sgd_solver.cpp:106] Iteration 240, lr = 0.01
I0907 17:49:14.828528  2018 solver.cpp:456] Snapshotting to binary proto file
 checkpoints/snapshot_iter_250.caffemodel
I0907 17:49:14.838659  2018 sgd_solver.cpp:273] Snapshotting solver state to
 binary proto file checkpoints/snapshot_iter_250.solverstate
I0907 17:49:14.839365  2018 solver.cpp:338] Iteration 250, Testing net (#0)
I0907 17:49:15.329273  2018 solver.cpp:406]     Test net output #0: accuracy
 = 0.5062
I0907 17:49:15.329335  2018 solver.cpp:406]     Test net output #1: loss = 0.
694212 (* 1 = 0.694212 loss)
I0907 17:49:15.472729  2018 solver.cpp:229] Iteration 260, loss = 0.696516
I0907 17:49:15.472792  2018 solver.cpp:245]     Train net output #0: loss =
 0.696516 (* 1 = 0.696516 loss)
I0907 17:49:15.472807  2018 sgd_solver.cpp:106] Iteration 260, lr = 0.01
I0907 17:49:15.749964  2018 solver.cpp:229] Iteration 280, loss = 0.698534
I0907 17:49:15.750020  2018 solver.cpp:245]     Train net output #0: loss =
 0.698534 (* 1 = 0.698534 loss)
I0907 17:49:15.750036  2018 sgd_solver.cpp:106] Iteration 280, lr = 0.01
I0907 17:49:16.026904  2018 solver.cpp:229] Iteration 300, loss = 0.690629
I0907 17:49:16.026959  2018 solver.cpp:245]     Train net output #0: loss =
 0.690629 (* 1 = 0.690629 loss)
I0907 17:49:16.026974  2018 sgd_solver.cpp:106] Iteration 300, lr = 0.01
I0907 17:49:16.304231  2018 solver.cpp:229] Iteration 320, loss = 0.691888
I0907 17:49:16.304287  2018 solver.cpp:245]     Train net output #0: loss =
 0.691888 (* 1 = 0.691888 loss)
I0907 17:49:16.304303  2018 sgd_solver.cpp:106] Iteration 320, lr = 0.01
I0907 17:49:16.580847  2018 solver.cpp:229] Iteration 340, loss = 0.69386
I0907 17:49:16.580900  2018 solver.cpp:245]     Train net output #0: loss =
 0.69386 (* 1 = 0.69386 loss)
I0907 17:49:16.580915  2018 sgd_solver.cpp:106] Iteration 340, lr = 0.01
I0907 17:49:16.857722  2018 solver.cpp:229] Iteration 360, loss = 0.690541
I0907 17:49:16.857780  2018 solver.cpp:245]     Train net output #0: loss =
 0.69054 (* 1 = 0.69054 loss)
I0907 17:49:16.857796  2018 sgd_solver.cpp:106] Iteration 360, lr = 0.01
I0907 17:49:17.134531  2018 solver.cpp:229] Iteration 380, loss = 0.699985
I0907 17:49:17.134583  2018 solver.cpp:245]     Train net output #0: loss =
 0.699985 (* 1 = 0.699985 loss)
I0907 17:49:17.134598  2018 sgd_solver.cpp:106] Iteration 380, lr = 0.01
I0907 17:49:17.411090  2018 solver.cpp:229] Iteration 400, loss = 0.698098
I0907 17:49:17.411149  2018 solver.cpp:245]     Train net output #0: loss =
```

```
 0.698098 (* 1 = 0.698098 loss)
I0907 17:49:17.411164  2018 sgd_solver.cpp:106] Iteration 400, lr = 0.01
I0907 17:49:17.687440  2018 solver.cpp:229] Iteration 420, loss = 0.695981
I0907 17:49:17.687492  2018 solver.cpp:245]     Train net output #0: loss =
 0.695981 (* 1 = 0.695981 loss)
I0907 17:49:17.687508  2018 sgd_solver.cpp:106] Iteration 420, lr = 0.01
I0907 17:49:17.963994  2018 solver.cpp:229] Iteration 440, loss = 0.693345
I0907 17:49:17.964051  2018 solver.cpp:245]     Train net output #0: loss =
 0.693345 (* 1 = 0.693345 loss)
I0907 17:49:17.964074  2018 sgd_solver.cpp:106] Iteration 440, lr = 0.01
I0907 17:49:18.241008  2018 solver.cpp:229] Iteration 460, loss = 0.6923
I0907 17:49:18.241071  2018 solver.cpp:245]     Train net output #0: loss =
 0.6923 (* 1 = 0.6923 loss)
I0907 17:49:18.241087  2018 sgd_solver.cpp:106] Iteration 460, lr = 0.01
I0907 17:49:18.517881  2018 solver.cpp:229] Iteration 480, loss = 0.691581
I0907 17:49:18.517935  2018 solver.cpp:245]     Train net output #0: loss =
 0.691581 (* 1 = 0.691581 loss)
I0907 17:49:18.517951  2018 sgd_solver.cpp:106] Iteration 480, lr = 0.01
I0907 17:49:18.780949  2018 solver.cpp:456] Snapshotting to binary proto file
 checkpoints/snapshot_iter_500.caffemodel
I0907 17:49:18.790834  2018 sgd_solver.cpp:273] Snapshotting solver state to
 binary proto file checkpoints/snapshot_iter_500.solverstate
I0907 17:49:18.791682  2018 solver.cpp:338] Iteration 500, Testing net (#0)
I0907 17:49:19.280625  2018 solver.cpp:406]     Test net output #0: accuracy
 = 0.5071
I0907 17:49:19.280684  2018 solver.cpp:406]     Test net output #1: loss = 0.
693794 (* 1 = 0.693794 loss)
I0907 17:49:19.285815  2018 solver.cpp:229] Iteration 500, loss = 0.694545
I0907 17:49:19.285861  2018 solver.cpp:245]     Train net output #0: loss =
 0.694545 (* 1 = 0.694545 loss)
I0907 17:49:19.285878  2018 sgd_solver.cpp:106] Iteration 500, lr = 0.001
I0907 17:49:19.563009  2018 solver.cpp:229] Iteration 520, loss = 0.688978
I0907 17:49:19.563076  2018 solver.cpp:245]     Train net output #0: loss =
 0.688978 (* 1 = 0.688978 loss)
I0907 17:49:19.563091  2018 sgd_solver.cpp:106] Iteration 520, lr = 0.001
I0907 17:49:19.840301  2018 solver.cpp:229] Iteration 540, loss = 0.689023
I0907 17:49:19.840355  2018 solver.cpp:245]     Train net output #0: loss =
 0.689023 (* 1 = 0.689023 loss)
I0907 17:49:19.840370  2018 sgd_solver.cpp:106] Iteration 540, lr = 0.001
I0907 17:49:20.117620  2018 solver.cpp:229] Iteration 560, loss = 0.691357
I0907 17:49:20.117674  2018 solver.cpp:245]     Train net output #0: loss =
 0.691357 (* 1 = 0.691357 loss)
I0907 17:49:20.117691  2018 sgd_solver.cpp:106] Iteration 560, lr = 0.001
I0907 17:49:20.395112  2018 solver.cpp:229] Iteration 580, loss = 0.686913
I0907 17:49:20.395167  2018 solver.cpp:245]     Train net output #0: loss =
 0.686913 (* 1 = 0.686913 loss)
I0907 17:49:20.395184  2018 sgd_solver.cpp:106] Iteration 580, lr = 0.001
I0907 17:49:20.672291  2018 solver.cpp:229] Iteration 600, loss = 0.688692
I0907 17:49:20.672343  2018 solver.cpp:245]     Train net output #0: loss =
 0.688692 (* 1 = 0.688692 loss)
I0907 17:49:20.672363  2018 sgd_solver.cpp:106] Iteration 600, lr = 0.001
I0907 17:49:20.949833  2018 solver.cpp:229] Iteration 620, loss = 0.690929
I0907 17:49:20.949885  2018 solver.cpp:245]     Train net output #0: loss =
 0.690929 (* 1 = 0.690929 loss)
I0907 17:49:20.949900  2018 sgd_solver.cpp:106] Iteration 620, lr = 0.001
I0907 17:49:21.227128  2018 solver.cpp:229] Iteration 640, loss = 0.696944
I0907 17:49:21.227190  2018 solver.cpp:245]     Train net output #0: loss =
```

```
 0.696944 (* 1 = 0.696944 loss)
I0907 17:49:21.227205  2018 sgd_solver.cpp:106] Iteration 640, lr = 0.001
I0907 17:49:21.504741  2018 solver.cpp:229] Iteration 660, loss = 0.690737
I0907 17:49:21.504804  2018 solver.cpp:245]     Train net output #0: loss =
 0.690737 (* 1 = 0.690737 loss)
I0907 17:49:21.504820  2018 sgd_solver.cpp:106] Iteration 660, lr = 0.001
I0907 17:49:21.781639  2018 solver.cpp:229] Iteration 680, loss = 0.696515
I0907 17:49:21.781697  2018 solver.cpp:245]     Train net output #0: loss =
 0.696515 (* 1 = 0.696515 loss)
I0907 17:49:21.781713  2018 sgd_solver.cpp:106] Iteration 680, lr = 0.001
I0907 17:49:22.058897  2018 solver.cpp:229] Iteration 700, loss = 0.696229
I0907 17:49:22.058950  2018 solver.cpp:245]     Train net output #0: loss =
 0.696229 (* 1 = 0.696229 loss)
I0907 17:49:22.058965  2018 sgd_solver.cpp:106] Iteration 700, lr = 0.001
I0907 17:49:22.336318  2018 solver.cpp:229] Iteration 720, loss = 0.692999
I0907 17:49:22.336371  2018 solver.cpp:245]     Train net output #0: loss =
 0.692999 (* 1 = 0.692999 loss)
I0907 17:49:22.336386  2018 sgd_solver.cpp:106] Iteration 720, lr = 0.001
I0907 17:49:22.613425  2018 solver.cpp:229] Iteration 740, loss = 0.690792
I0907 17:49:22.613484  2018 solver.cpp:245]     Train net output #0: loss =
 0.690792 (* 1 = 0.690792 loss)
I0907 17:49:22.613499  2018 sgd_solver.cpp:106] Iteration 740, lr = 0.001
I0907 17:49:22.738293  2018 solver.cpp:456] Snapshotting to binary proto file
 checkpoints/snapshot_iter_750.caffemodel
I0907 17:49:22.748927  2018 sgd_solver.cpp:273] Snapshotting solver state to
 binary proto file checkpoints/snapshot_iter_750.solverstate
I0907 17:49:22.750164  2018 solver.cpp:338] Iteration 750, Testing net (#0)
I0907 17:49:23.240234  2018 solver.cpp:406]     Test net output #0: accuracy
 = 0.5054
I0907 17:49:23.240293  2018 solver.cpp:406]     Test net output #1: loss = 0.
693245 (* 1 = 0.693245 loss)
I0907 17:49:23.240309  2018 solver.cpp:323] Optimization Done.
I0907 17:49:23.240315  2018 caffe.cpp:216] Optimization Done.
```

After 750 training iterations the training accuracy is about 50%. Not impressive!

We can achieve much higher accuracy and a lower loss against both the training and validation datasets for this task. Ir seems our network is underfitting the data.

Refinements

Many network configurations such as Alexnet, GoogLeNet and VGG are significantly larger than the three layer architecture used above and have proven to be very accurate at classifying the ImageNet images. Now, we are going to increase the complexity of this network to improve the accuracy.

There are many knobs that one can turn in choosing a neural network architecture. For example, add layers, increased the number of learned weights, change the learning rate or introduce a more complex policy to modify the learning rate as training progresses. We will experiment with some of these modifications to see the effect on classification accuracy.

Refinement #1

We will increase the number of outputs in the convolutional layers to 64 for layer 1 and 128 for layer 2. Also add a third convolutional layer that is identical to the second one. Just make sure to have the right layer outputs feeding into this new layer and the final fully-connected layer.

This time it will take about 90 seconds to train due to the increased network size.

In [9]:
```
%%bash
TOOLS=/home/ubuntu/caffe/build/tools
#Train your modified network configuration
$TOOLS/caffe train -gpu 0 -solver src/solver2.prototxt
```

```
I0907 17:49:56.266579  2026 upgrade_proto.cpp:990] Attempting to upgrade inpu
t file specified using deprecated 'solver_type' field (enum)': src/solver2.pr
ototxt
I0907 17:49:56.266804  2026 upgrade_proto.cpp:997] Successfully upgraded file
 specified using deprecated 'solver_type' field (enum) to 'type' field (strin
g).
W0907 17:49:56.266813  2026 upgrade_proto.cpp:999] Note that future Caffe rel
eases will only support 'type' field (string) for a solver's type.
I0907 17:49:56.266942  2026 caffe.cpp:185] Using GPUs 0
I0907 17:49:56.397658  2026 solver.cpp:48] Initializing solver from parameter
s:
test_iter: 100
test_interval: 250
base_lr: 0.01
display: 20
max_iter: 750
lr_policy: "step"
gamma: 0.1
momentum: 0.9
weight_decay: 0.0005
stepsize: 500
snapshot: 250
snapshot_prefix: "checkpoints/snapshot"
solver_mode: GPU
device_id: 0
random_seed: 1234
net: "src/train_val2.answer.prototxt"
type: "SGD"
I0907 17:49:56.397824  2026 solver.cpp:91] Creating training net from net fil
e: src/train_val2.answer.prototxt
I0907 17:49:56.398375  2026 net.cpp:322] The NetState phase (0) differed from
 the phase (1) specified by a rule in layer data
I0907 17:49:56.398409  2026 net.cpp:322] The NetState phase (0) differed from
 the phase (1) specified by a rule in layer accuracy
I0907 17:49:56.398548  2026 net.cpp:49] Initializing net from parameters:
state {
  phase: TRAIN
}
layer {
  name: "data"
  type: "Data"
  top: "data"
  top: "label"
  include {
    phase: TRAIN
  }
  transform_param {
    mirror: true
    crop_size: 32
    mean_file: "mean.binaryproto"
  }
  data_param {
    source: "train_lmdb"
    batch_size: 100
    backend: LMDB
  }
}
```

```
layer {
  name: "conv1"
  type: "Convolution"
  bottom: "data"
  top: "conv1"
  param {
    lr_mult: 1
  }
  param {
    lr_mult: 2
  }
  convolution_param {
    num_output: 64
    pad: 2
    kernel_size: 5
    stride: 1
    weight_filler {
      type: "gaussian"
      std: 0.0001
    }
    bias_filler {
      type: "constant"
    }
  }
}
layer {
  name: "pool1"
  type: "Pooling"
  bottom: "conv1"
  top: "pool1"
  pooling_param {
    pool: MAX
    kernel_size: 3
    stride: 2
  }
}
layer {
  name: "relu1"
  type: "ReLU"
  bottom: "pool1"
  top: "pool1"
}
layer {
  name: "conv2"
  type: "Convolution"
  bottom: "pool1"
  top: "conv2"
  param {
    lr_mult: 1
  }
  param {
    lr_mult: 2
  }
  convolution_param {
    num_output: 128
    pad: 2
    kernel_size: 5
```

```
    stride: 1
    weight_filler {
      type: "gaussian"
      std: 0.01
    }
    bias_filler {
      type: "constant"
    }
  }
}
layer {
  name: "relu2"
  type: "ReLU"
  bottom: "conv2"
  top: "conv2"
}
layer {
  name: "pool2"
  type: "Pooling"
  bottom: "conv2"
  top: "pool2"
  pooling_param {
    pool: AVE
    kernel_size: 3
    stride: 2
  }
}
layer {
  name: "conv3"
  type: "Convolution"
  bottom: "pool2"
  top: "conv3"
  param {
    lr_mult: 1
  }
  param {
    lr_mult: 2
  }
  convolution_param {
    num_output: 128
    pad: 2
    kernel_size: 5
    stride: 1
    weight_filler {
      type: "gaussian"
      std: 0.01
    }
    bias_filler {
      type: "constant"
    }
  }
}
layer {
  name: "relu3"
  type: "ReLU"
  bottom: "conv3"
  top: "conv3"
```

```
}
layer {
  name: "pool3"
  type: "Pooling"
  bottom: "conv3"
  top: "pool3"
  pooling_param {
    pool: AVE
    kernel_size: 3
    stride: 2
  }
}
layer {
  name: "ip1"
  type: "InnerProduct"
  bottom: "pool3"
  top: "ip1"
  param {
    lr_mult: 1
    decay_mult: 250
  }
  param {
    lr_mult: 2
    decay_mult: 0
  }
  inner_product_param {
    num_output: 2
    weight_filler {
      type: "gaussian"
      std: 0.01
    }
    bias_filler {
      type: "constant"
    }
  }
}
layer {
  name: "loss"
  type: "SoftmaxWithLoss"
  bottom: "ip1"
  bottom: "label"
  top: "loss"
}
I0907 17:49:56.398650  2026 layer_factory.hpp:77] Creating layer data
I0907 17:49:56.398857  2026 net.cpp:106] Creating Layer data
I0907 17:49:56.398879  2026 net.cpp:411] data -> data
I0907 17:49:56.398928  2026 net.cpp:411] data -> label
I0907 17:49:56.398957  2026 data_transformer.cpp:25] Loading mean file from:
 mean.binaryproto
I0907 17:49:56.399595  2029 db_lmdb.cpp:38] Opened lmdb train_lmdb
I0907 17:49:56.411839  2026 data_layer.cpp:41] output data size: 100,3,32,32
I0907 17:49:56.414516  2026 net.cpp:150] Setting up data
I0907 17:49:56.414572  2026 net.cpp:157] Top shape: 100 3 32 32 (307200)
I0907 17:49:56.414580  2026 net.cpp:157] Top shape: 100 (100)
I0907 17:49:56.414584  2026 net.cpp:165] Memory required for data: 1229200
I0907 17:49:56.414595  2026 layer_factory.hpp:77] Creating layer conv1
I0907 17:49:56.414625  2026 net.cpp:106] Creating Layer conv1
```

```
I0907 17:49:56.414636  2026 net.cpp:454] conv1 <- data
I0907 17:49:56.414657  2026 net.cpp:411] conv1 -> conv1
I0907 17:49:56.531949  2026 net.cpp:150] Setting up conv1
I0907 17:49:56.531997  2026 net.cpp:157] Top shape: 100 64 32 32 (6553600)
I0907 17:49:56.532004  2026 net.cpp:165] Memory required for data: 27443600
I0907 17:49:56.532033  2026 layer_factory.hpp:77] Creating layer pool1
I0907 17:49:56.532063  2026 net.cpp:106] Creating Layer pool1
I0907 17:49:56.532078  2026 net.cpp:454] pool1 <- conv1
I0907 17:49:56.532088  2026 net.cpp:411] pool1 -> pool1
I0907 17:49:56.532163  2026 net.cpp:150] Setting up pool1
I0907 17:49:56.532181  2026 net.cpp:157] Top shape: 100 64 16 16 (1638400)
I0907 17:49:56.532186  2026 net.cpp:165] Memory required for data: 33997200
I0907 17:49:56.532194  2026 layer_factory.hpp:77] Creating layer relu1
I0907 17:49:56.532203  2026 net.cpp:106] Creating Layer relu1
I0907 17:49:56.532213  2026 net.cpp:454] relu1 <- pool1
I0907 17:49:56.532222  2026 net.cpp:397] relu1 -> pool1 (in-place)
I0907 17:49:56.532369  2026 net.cpp:150] Setting up relu1
I0907 17:49:56.532388  2026 net.cpp:157] Top shape: 100 64 16 16 (1638400)
I0907 17:49:56.532394  2026 net.cpp:165] Memory required for data: 40550800
I0907 17:49:56.532399  2026 layer_factory.hpp:77] Creating layer conv2
I0907 17:49:56.532420  2026 net.cpp:106] Creating Layer conv2
I0907 17:49:56.532433  2026 net.cpp:454] conv2 <- pool1
I0907 17:49:56.532441  2026 net.cpp:411] conv2 -> conv2
I0907 17:49:56.540318  2026 net.cpp:150] Setting up conv2
I0907 17:49:56.540345  2026 net.cpp:157] Top shape: 100 128 16 16 (3276800)
I0907 17:49:56.540357  2026 net.cpp:165] Memory required for data: 53658000
I0907 17:49:56.540370  2026 layer_factory.hpp:77] Creating layer relu2
I0907 17:49:56.540385  2026 net.cpp:106] Creating Layer relu2
I0907 17:49:56.540390  2026 net.cpp:454] relu2 <- conv2
I0907 17:49:56.540407  2026 net.cpp:397] relu2 -> conv2 (in-place)
I0907 17:49:56.540650  2026 net.cpp:150] Setting up relu2
I0907 17:49:56.540671  2026 net.cpp:157] Top shape: 100 128 16 16 (3276800)
I0907 17:49:56.540676  2026 net.cpp:165] Memory required for data: 66765200
I0907 17:49:56.540689  2026 layer_factory.hpp:77] Creating layer pool2
I0907 17:49:56.540701  2026 net.cpp:106] Creating Layer pool2
I0907 17:49:56.540712  2026 net.cpp:454] pool2 <- conv2
I0907 17:49:56.540720  2026 net.cpp:411] pool2 -> pool2
I0907 17:49:56.540904  2026 net.cpp:150] Setting up pool2
I0907 17:49:56.540925  2026 net.cpp:157] Top shape: 100 128 8 8 (819200)
I0907 17:49:56.540930  2026 net.cpp:165] Memory required for data: 70042000
I0907 17:49:56.540940  2026 layer_factory.hpp:77] Creating layer conv3
I0907 17:49:56.540956  2026 net.cpp:106] Creating Layer conv3
I0907 17:49:56.540966  2026 net.cpp:454] conv3 <- pool2
I0907 17:49:56.540978  2026 net.cpp:411] conv3 -> conv3
I0907 17:49:56.555501  2026 net.cpp:150] Setting up conv3
I0907 17:49:56.555526  2026 net.cpp:157] Top shape: 100 128 8 8 (819200)
I0907 17:49:56.555532  2026 net.cpp:165] Memory required for data: 73318800
I0907 17:49:56.555548  2026 layer_factory.hpp:77] Creating layer relu3
I0907 17:49:56.555560  2026 net.cpp:106] Creating Layer relu3
I0907 17:49:56.555567  2026 net.cpp:454] relu3 <- conv3
I0907 17:49:56.555577  2026 net.cpp:397] relu3 -> conv3 (in-place)
I0907 17:49:56.555745  2026 net.cpp:150] Setting up relu3
I0907 17:49:56.555764  2026 net.cpp:157] Top shape: 100 128 8 8 (819200)
I0907 17:49:56.555770  2026 net.cpp:165] Memory required for data: 76595600
I0907 17:49:56.555778  2026 layer_factory.hpp:77] Creating layer pool3
I0907 17:49:56.555788  2026 net.cpp:106] Creating Layer pool3
I0907 17:49:56.555799  2026 net.cpp:454] pool3 <- conv3
```

```
I0907 17:49:56.555843  2026 net.cpp:411] pool3 -> pool3
I0907 17:49:56.556097  2026 net.cpp:150] Setting up pool3
I0907 17:49:56.556119  2026 net.cpp:157] Top shape: 100 128 4 4 (204800)
I0907 17:49:56.556125  2026 net.cpp:165] Memory required for data: 77414800
I0907 17:49:56.556131  2026 layer_factory.hpp:77] Creating layer ip1
I0907 17:49:56.556148  2026 net.cpp:106] Creating Layer ip1
I0907 17:49:56.556159  2026 net.cpp:454] ip1 <- pool3
I0907 17:49:56.556174  2026 net.cpp:411] ip1 -> ip1
I0907 17:49:56.556427  2026 net.cpp:150] Setting up ip1
I0907 17:49:56.556444  2026 net.cpp:157] Top shape: 100 2 (200)
I0907 17:49:56.556449  2026 net.cpp:165] Memory required for data: 77415600
I0907 17:49:56.556462  2026 layer_factory.hpp:77] Creating layer loss
I0907 17:49:56.556475  2026 net.cpp:106] Creating Layer loss
I0907 17:49:56.556486  2026 net.cpp:454] loss <- ip1
I0907 17:49:56.556493  2026 net.cpp:454] loss <- label
I0907 17:49:56.556509  2026 net.cpp:411] loss -> loss
I0907 17:49:56.556526  2026 layer_factory.hpp:77] Creating layer loss
I0907 17:49:56.556879  2026 net.cpp:150] Setting up loss
I0907 17:49:56.556900  2026 net.cpp:157] Top shape: (1)
I0907 17:49:56.556905  2026 net.cpp:160]     with loss weight 1
I0907 17:49:56.556936  2026 net.cpp:165] Memory required for data: 77415604
I0907 17:49:56.556942  2026 net.cpp:226] loss needs backward computation.
I0907 17:49:56.556948  2026 net.cpp:226] ip1 needs backward computation.
I0907 17:49:56.556954  2026 net.cpp:226] pool3 needs backward computation.
I0907 17:49:56.556958  2026 net.cpp:226] relu3 needs backward computation.
I0907 17:49:56.556964  2026 net.cpp:226] conv3 needs backward computation.
I0907 17:49:56.556969  2026 net.cpp:226] pool2 needs backward computation.
I0907 17:49:56.556979  2026 net.cpp:226] relu2 needs backward computation.
I0907 17:49:56.556984  2026 net.cpp:226] conv2 needs backward computation.
I0907 17:49:56.556988  2026 net.cpp:226] relu1 needs backward computation.
I0907 17:49:56.556993  2026 net.cpp:226] pool1 needs backward computation.
I0907 17:49:56.556998  2026 net.cpp:226] conv1 needs backward computation.
I0907 17:49:56.557004  2026 net.cpp:228] data does not need backward computat
ion.
I0907 17:49:56.557009  2026 net.cpp:270] This network produces output loss
I0907 17:49:56.557026  2026 net.cpp:283] Network initialization done.
I0907 17:49:56.557523  2026 solver.cpp:181] Creating test net (#0) specified
 by net file: src/train_val2.answer.prototxt
I0907 17:49:56.557579  2026 net.cpp:322] The NetState phase (1) differed from
 the phase (0) specified by a rule in layer data
I0907 17:49:56.557720  2026 net.cpp:49] Initializing net from parameters:
state {
  phase: TEST
}
layer {
  name: "data"
  type: "Data"
  top: "data"
  top: "label"
  include {
    phase: TEST
  }
  transform_param {
    crop_size: 32
    mean_file: "mean.binaryproto"
  }
  data_param {
```

```
    source: "val_lmdb"
    batch_size: 100
    backend: LMDB
  }
}
layer {
  name: "conv1"
  type: "Convolution"
  bottom: "data"
  top: "conv1"
  param {
    lr_mult: 1
  }
  param {
    lr_mult: 2
  }
  convolution_param {
    num_output: 64
    pad: 2
    kernel_size: 5
    stride: 1
    weight_filler {
      type: "gaussian"
      std: 0.0001
    }
    bias_filler {
      type: "constant"
    }
  }
}
layer {
  name: "pool1"
  type: "Pooling"
  bottom: "conv1"
  top: "pool1"
  pooling_param {
    pool: MAX
    kernel_size: 3
    stride: 2
  }
}
layer {
  name: "relu1"
  type: "ReLU"
  bottom: "pool1"
  top: "pool1"
}
layer {
  name: "conv2"
  type: "Convolution"
  bottom: "pool1"
  top: "conv2"
  param {
    lr_mult: 1
  }
  param {
    lr_mult: 2
```

```
    }
    convolution_param {
      num_output: 128
      pad: 2
      kernel_size: 5
      stride: 1
      weight_filler {
        type: "gaussian"
        std: 0.01
      }
      bias_filler {
        type: "constant"
      }
    }
}
layer {
  name: "relu2"
  type: "ReLU"
  bottom: "conv2"
  top: "conv2"
}
layer {
  name: "pool2"
  type: "Pooling"
  bottom: "conv2"
  top: "pool2"
  pooling_param {
    pool: AVE
    kernel_size: 3
    stride: 2
  }
}
layer {
  name: "conv3"
  type: "Convolution"
  bottom: "pool2"
  top: "conv3"
  param {
    lr_mult: 1
  }
  param {
    lr_mult: 2
  }
  convolution_param {
    num_output: 128
    pad: 2
    kernel_size: 5
    stride: 1
    weight_filler {
      type: "gaussian"
      std: 0.01
    }
    bias_filler {
      type: "constant"
    }
  }
}
```

```
layer {
  name: "relu3"
  type: "ReLU"
  bottom: "conv3"
  top: "conv3"
}
layer {
  name: "pool3"
  type: "Pooling"
  bottom: "conv3"
  top: "pool3"
  pooling_param {
    pool: AVE
    kernel_size: 3
    stride: 2
  }
}
layer {
  name: "ip1"
  type: "InnerProduct"
  bottom: "pool3"
  top: "ip1"
  param {
    lr_mult: 1
    decay_mult: 250
  }
  param {
    lr_mult: 2
    decay_mult: 0
  }
  inner_product_param {
    num_output: 2
    weight_filler {
      type: "gaussian"
      std: 0.01
    }
    bias_filler {
      type: "constant"
    }
  }
}
layer {
  name: "loss"
  type: "SoftmaxWithLoss"
  bottom: "ip1"
  bottom: "label"
  top: "loss"
}
layer {
  name: "accuracy"
  type: "Accuracy"
  bottom: "ip1"
  bottom: "label"
  top: "accuracy"
  include {
    phase: TEST
  }
```

```
}
I0907 17:49:56.557842  2026 layer_factory.hpp:77] Creating layer data
I0907 17:49:56.557986  2026 net.cpp:106] Creating Layer data
I0907 17:49:56.558002  2026 net.cpp:411] data -> data
I0907 17:49:56.558014  2026 net.cpp:411] data -> label
I0907 17:49:56.558037  2026 data_transformer.cpp:25] Loading mean file from:
 mean.binaryproto
I0907 17:49:56.558761  2031 db_lmdb.cpp:38] Opened lmdb val_lmdb
I0907 17:49:56.558881  2026 data_layer.cpp:41] output data size: 100,3,32,32
I0907 17:49:56.561842  2026 net.cpp:150] Setting up data
I0907 17:49:56.561864  2026 net.cpp:157] Top shape: 100 3 32 32 (307200)
I0907 17:49:56.561872  2026 net.cpp:157] Top shape: 100 (100)
I0907 17:49:56.561877  2026 net.cpp:165] Memory required for data: 1229200
I0907 17:49:56.561882  2026 layer_factory.hpp:77] Creating layer label_data_1
_split
I0907 17:49:56.561890  2026 net.cpp:106] Creating Layer label_data_1_split
I0907 17:49:56.561897  2026 net.cpp:454] label_data_1_split <- label
I0907 17:49:56.561904  2026 net.cpp:411] label_data_1_split -> label_data_1_s
plit_0
I0907 17:49:56.561918  2026 net.cpp:411] label_data_1_split -> label_data_1_s
plit_1
I0907 17:49:56.561998  2026 net.cpp:150] Setting up label_data_1_split
I0907 17:49:56.562016  2026 net.cpp:157] Top shape: 100 (100)
I0907 17:49:56.562022  2026 net.cpp:157] Top shape: 100 (100)
I0907 17:49:56.562026  2026 net.cpp:165] Memory required for data: 1230000
I0907 17:49:56.562031  2026 layer_factory.hpp:77] Creating layer conv1
I0907 17:49:56.562055  2026 net.cpp:106] Creating Layer conv1
I0907 17:49:56.562062  2026 net.cpp:454] conv1 <- data
I0907 17:49:56.562073  2026 net.cpp:411] conv1 -> conv1
I0907 17:49:56.563552  2026 net.cpp:150] Setting up conv1
I0907 17:49:56.563575  2026 net.cpp:157] Top shape: 100 64 32 32 (6553600)
I0907 17:49:56.563580  2026 net.cpp:165] Memory required for data: 27444400
I0907 17:49:56.563621  2026 layer_factory.hpp:77] Creating layer pool1
I0907 17:49:56.563632  2026 net.cpp:106] Creating Layer pool1
I0907 17:49:56.563639  2026 net.cpp:454] pool1 <- conv1
I0907 17:49:56.563649  2026 net.cpp:411] pool1 -> pool1
I0907 17:49:56.563697  2026 net.cpp:150] Setting up pool1
I0907 17:49:56.563714  2026 net.cpp:157] Top shape: 100 64 16 16 (1638400)
I0907 17:49:56.563719  2026 net.cpp:165] Memory required for data: 33998000
I0907 17:49:56.563724  2026 layer_factory.hpp:77] Creating layer relu1
I0907 17:49:56.563735  2026 net.cpp:106] Creating Layer relu1
I0907 17:49:56.563742  2026 net.cpp:454] relu1 <- pool1
I0907 17:49:56.563750  2026 net.cpp:397] relu1 -> pool1 (in-place)
I0907 17:49:56.564000  2026 net.cpp:150] Setting up relu1
I0907 17:49:56.564020  2026 net.cpp:157] Top shape: 100 64 16 16 (1638400)
I0907 17:49:56.564028  2026 net.cpp:165] Memory required for data: 40551600
I0907 17:49:56.564034  2026 layer_factory.hpp:77] Creating layer conv2
I0907 17:49:56.564045  2026 net.cpp:106] Creating Layer conv2
I0907 17:49:56.564050  2026 net.cpp:454] conv2 <- pool1
I0907 17:49:56.564061  2026 net.cpp:411] conv2 -> conv2
I0907 17:49:56.571995  2026 net.cpp:150] Setting up conv2
I0907 17:49:56.572021  2026 net.cpp:157] Top shape: 100 128 16 16 (3276800)
I0907 17:49:56.572026  2026 net.cpp:165] Memory required for data: 53658800
I0907 17:49:56.572038  2026 layer_factory.hpp:77] Creating layer relu2
I0907 17:49:56.572053  2026 net.cpp:106] Creating Layer relu2
I0907 17:49:56.572064  2026 net.cpp:454] relu2 <- conv2
I0907 17:49:56.572075  2026 net.cpp:397] relu2 -> conv2 (in-place)
```

```
I0907 17:49:56.572238  2026 net.cpp:150] Setting up relu2
I0907 17:49:56.572257  2026 net.cpp:157] Top shape: 100 128 16 16 (3276800)
I0907 17:49:56.572263  2026 net.cpp:165] Memory required for data: 66766000
I0907 17:49:56.572274  2026 layer_factory.hpp:77] Creating layer pool2
I0907 17:49:56.572283  2026 net.cpp:106] Creating Layer pool2
I0907 17:49:56.572295  2026 net.cpp:454] pool2 <- conv2
I0907 17:49:56.572304  2026 net.cpp:411] pool2 -> pool2
I0907 17:49:56.572567  2026 net.cpp:150] Setting up pool2
I0907 17:49:56.572588  2026 net.cpp:157] Top shape: 100 128 8 8 (819200)
I0907 17:49:56.572593  2026 net.cpp:165] Memory required for data: 70042800
I0907 17:49:56.572600  2026 layer_factory.hpp:77] Creating layer conv3
I0907 17:49:56.572618  2026 net.cpp:106] Creating Layer conv3
I0907 17:49:56.572629  2026 net.cpp:454] conv3 <- pool2
I0907 17:49:56.572638  2026 net.cpp:411] conv3 -> conv3
I0907 17:49:56.588001  2026 net.cpp:150] Setting up conv3
I0907 17:49:56.588032  2026 net.cpp:157] Top shape: 100 128 8 8 (819200)
I0907 17:49:56.588037  2026 net.cpp:165] Memory required for data: 73319600
I0907 17:49:56.588058  2026 layer_factory.hpp:77] Creating layer relu3
I0907 17:49:56.588073  2026 net.cpp:106] Creating Layer relu3
I0907 17:49:56.588079  2026 net.cpp:454] relu3 <- conv3
I0907 17:49:56.588091  2026 net.cpp:397] relu3 -> conv3 (in-place)
I0907 17:49:56.588265  2026 net.cpp:150] Setting up relu3
I0907 17:49:56.588284  2026 net.cpp:157] Top shape: 100 128 8 8 (819200)
I0907 17:49:56.588290  2026 net.cpp:165] Memory required for data: 76596400
I0907 17:49:56.588295  2026 layer_factory.hpp:77] Creating layer pool3
I0907 17:49:56.588307  2026 net.cpp:106] Creating Layer pool3
I0907 17:49:56.588317  2026 net.cpp:454] pool3 <- conv3
I0907 17:49:56.588328  2026 net.cpp:411] pool3 -> pool3
I0907 17:49:56.588639  2026 net.cpp:150] Setting up pool3
I0907 17:49:56.588657  2026 net.cpp:157] Top shape: 100 128 4 4 (204800)
I0907 17:49:56.588663  2026 net.cpp:165] Memory required for data: 77415600
I0907 17:49:56.588668  2026 layer_factory.hpp:77] Creating layer ip1
I0907 17:49:56.588683  2026 net.cpp:106] Creating Layer ip1
I0907 17:49:56.588691  2026 net.cpp:454] ip1 <- pool3
I0907 17:49:56.588701  2026 net.cpp:411] ip1 -> ip1
I0907 17:49:56.588956  2026 net.cpp:150] Setting up ip1
I0907 17:49:56.588976  2026 net.cpp:157] Top shape: 100 2 (200)
I0907 17:49:56.588981  2026 net.cpp:165] Memory required for data: 77416400
I0907 17:49:56.588999  2026 layer_factory.hpp:77] Creating layer ip1_ip1_0_sp
lit
I0907 17:49:56.589013  2026 net.cpp:106] Creating Layer ip1_ip1_0_split
I0907 17:49:56.589020  2026 net.cpp:454] ip1_ip1_0_split <- ip1
I0907 17:49:56.589027  2026 net.cpp:411] ip1_ip1_0_split -> ip1_ip1_0_split_0
I0907 17:49:56.589035  2026 net.cpp:411] ip1_ip1_0_split -> ip1_ip1_0_split_1
I0907 17:49:56.589088  2026 net.cpp:150] Setting up ip1_ip1_0_split
I0907 17:49:56.589103  2026 net.cpp:157] Top shape: 100 2 (200)
I0907 17:49:56.589110  2026 net.cpp:157] Top shape: 100 2 (200)
I0907 17:49:56.589117  2026 net.cpp:165] Memory required for data: 77418000
I0907 17:49:56.589121  2026 layer_factory.hpp:77] Creating layer loss
I0907 17:49:56.589129  2026 net.cpp:106] Creating Layer loss
I0907 17:49:56.589139  2026 net.cpp:454] loss <- ip1_ip1_0_split_0
I0907 17:49:56.589145  2026 net.cpp:454] loss <- label_data_1_split_0
I0907 17:49:56.589156  2026 net.cpp:411] loss -> loss
I0907 17:49:56.589169  2026 layer_factory.hpp:77] Creating layer loss
I0907 17:49:56.589413  2026 net.cpp:150] Setting up loss
I0907 17:49:56.589433  2026 net.cpp:157] Top shape: (1)
I0907 17:49:56.589462  2026 net.cpp:160]     with loss weight 1
```

```
I0907 17:49:56.589478  2026 net.cpp:165] Memory required for data: 77418004
I0907 17:49:56.589483  2026 layer_factory.hpp:77] Creating layer accuracy
I0907 17:49:56.589498  2026 net.cpp:106] Creating Layer accuracy
I0907 17:49:56.589504  2026 net.cpp:454] accuracy <- ip1_ip1_0_split_1
I0907 17:49:56.589510  2026 net.cpp:454] accuracy <- label_data_1_split_1
I0907 17:49:56.589521  2026 net.cpp:411] accuracy -> accuracy
I0907 17:49:56.589540  2026 net.cpp:150] Setting up accuracy
I0907 17:49:56.589552  2026 net.cpp:157] Top shape: (1)
I0907 17:49:56.589557  2026 net.cpp:165] Memory required for data: 77418008
I0907 17:49:56.589562  2026 net.cpp:228] accuracy does not need backward comp
utation.
I0907 17:49:56.589570  2026 net.cpp:226] loss needs backward computation.
I0907 17:49:56.589576  2026 net.cpp:226] ip1_ip1_0_split needs backward compu
tation.
I0907 17:49:56.589579  2026 net.cpp:226] ip1 needs backward computation.
I0907 17:49:56.589586  2026 net.cpp:226] pool3 needs backward computation.
I0907 17:49:56.589591  2026 net.cpp:226] relu3 needs backward computation.
I0907 17:49:56.589594  2026 net.cpp:226] conv3 needs backward computation.
I0907 17:49:56.589601  2026 net.cpp:226] pool2 needs backward computation.
I0907 17:49:56.589606  2026 net.cpp:226] relu2 needs backward computation.
I0907 17:49:56.589610  2026 net.cpp:226] conv2 needs backward computation.
I0907 17:49:56.589617  2026 net.cpp:226] relu1 needs backward computation.
I0907 17:49:56.589622  2026 net.cpp:226] pool1 needs backward computation.
I0907 17:49:56.589625  2026 net.cpp:226] conv1 needs backward computation.
I0907 17:49:56.589633  2026 net.cpp:228] label_data_1_split does not need bac
kward computation.
I0907 17:49:56.589638  2026 net.cpp:228] data does not need backward computat
ion.
I0907 17:49:56.589643  2026 net.cpp:270] This network produces output accurac
y
I0907 17:49:56.589648  2026 net.cpp:270] This network produces output loss
I0907 17:49:56.589663  2026 net.cpp:283] Network initialization done.
I0907 17:49:56.589745  2026 solver.cpp:60] Solver scaffolding done.
I0907 17:49:56.590039  2026 caffe.cpp:213] Starting Optimization
I0907 17:49:56.590062  2026 solver.cpp:280] Solving
I0907 17:49:56.590067  2026 solver.cpp:281] Learning Rate Policy: step
I0907 17:49:56.590569  2026 solver.cpp:338] Iteration 0, Testing net (#0)
I0907 17:49:58.996317  2026 solver.cpp:406]     Test net output #0: accuracy
 = 0.4942
I0907 17:49:58.996376  2026 solver.cpp:406]     Test net output #1: loss = 0.
693116 (* 1 = 0.693116 loss)
I0907 17:49:59.022239  2026 solver.cpp:229] Iteration 0, loss = 0.692804
I0907 17:49:59.022272  2026 solver.cpp:245]     Train net output #0: loss =
 0.692804 (* 1 = 0.692804 loss)
I0907 17:49:59.022292  2026 sgd_solver.cpp:106] Iteration 0, lr = 0.01
I0907 17:50:01.016228  2026 solver.cpp:229] Iteration 20, loss = 0.681038
I0907 17:50:01.016283  2026 solver.cpp:245]     Train net output #0: loss =
 0.681038 (* 1 = 0.681038 loss)
I0907 17:50:01.016294  2026 sgd_solver.cpp:106] Iteration 20, lr = 0.01
I0907 17:50:03.008795  2026 solver.cpp:229] Iteration 40, loss = 0.67226
I0907 17:50:03.008852  2026 solver.cpp:245]     Train net output #0: loss =
 0.67226 (* 1 = 0.67226 loss)
I0907 17:50:03.008862  2026 sgd_solver.cpp:106] Iteration 40, lr = 0.01
I0907 17:50:05.001580  2026 solver.cpp:229] Iteration 60, loss = 0.673935
I0907 17:50:05.001646  2026 solver.cpp:245]     Train net output #0: loss =
 0.673935 (* 1 = 0.673935 loss)
I0907 17:50:05.001655  2026 sgd_solver.cpp:106] Iteration 60, lr = 0.01
```

```
I0907 17:50:06.994484  2026 solver.cpp:229] Iteration 80, loss = 0.692611
I0907 17:50:06.994542  2026 solver.cpp:245]     Train net output #0: loss =
 0.692611 (* 1 = 0.692611 loss)
I0907 17:50:06.994554  2026 sgd_solver.cpp:106] Iteration 80, lr = 0.01
I0907 17:50:08.988448  2026 solver.cpp:229] Iteration 100, loss = 0.674448
I0907 17:50:08.988512  2026 solver.cpp:245]     Train net output #0: loss =
 0.674448 (* 1 = 0.674448 loss)
I0907 17:50:08.988523  2026 sgd_solver.cpp:106] Iteration 100, lr = 0.01
I0907 17:50:10.983147  2026 solver.cpp:229] Iteration 120, loss = 0.683734
I0907 17:50:10.983203  2026 solver.cpp:245]     Train net output #0: loss =
 0.683734 (* 1 = 0.683734 loss)
I0907 17:50:10.983213  2026 sgd_solver.cpp:106] Iteration 120, lr = 0.01
I0907 17:50:12.977910  2026 solver.cpp:229] Iteration 140, loss = 0.63254
I0907 17:50:12.977974  2026 solver.cpp:245]     Train net output #0: loss =
 0.63254 (* 1 = 0.63254 loss)
I0907 17:50:12.977984  2026 sgd_solver.cpp:106] Iteration 140, lr = 0.01
I0907 17:50:14.970306  2026 solver.cpp:229] Iteration 160, loss = 0.624842
I0907 17:50:14.970360  2026 solver.cpp:245]     Train net output #0: loss =
 0.624842 (* 1 = 0.624842 loss)
I0907 17:50:14.970371  2026 sgd_solver.cpp:106] Iteration 160, lr = 0.01
I0907 17:50:16.963953  2026 solver.cpp:229] Iteration 180, loss = 0.597834
I0907 17:50:16.964012  2026 solver.cpp:245]     Train net output #0: loss =
 0.597834 (* 1 = 0.597834 loss)
I0907 17:50:16.964023  2026 sgd_solver.cpp:106] Iteration 180, lr = 0.01
I0907 17:50:18.958662  2026 solver.cpp:229] Iteration 200, loss = 0.59141
I0907 17:50:18.958729  2026 solver.cpp:245]     Train net output #0: loss =
 0.59141 (* 1 = 0.59141 loss)
I0907 17:50:18.958745  2026 sgd_solver.cpp:106] Iteration 200, lr = 0.01
I0907 17:50:20.951462  2026 solver.cpp:229] Iteration 220, loss = 0.66259
I0907 17:50:20.951515  2026 solver.cpp:245]     Train net output #0: loss =
 0.66259 (* 1 = 0.66259 loss)
I0907 17:50:20.951526  2026 sgd_solver.cpp:106] Iteration 220, lr = 0.01
I0907 17:50:22.946105  2026 solver.cpp:229] Iteration 240, loss = 0.6401
I0907 17:50:22.946171  2026 solver.cpp:245]     Train net output #0: loss =
 0.6401 (* 1 = 0.6401 loss)
I0907 17:50:22.946182  2026 sgd_solver.cpp:106] Iteration 240, lr = 0.01
I0907 17:50:23.843701  2026 solver.cpp:456] Snapshotting to binary proto file
 checkpoints/snapshot_iter_250.caffemodel
I0907 17:50:23.933241  2026 sgd_solver.cpp:273] Snapshotting solver state to
 binary proto file checkpoints/snapshot_iter_250.solverstate
I0907 17:50:23.938874  2026 solver.cpp:338] Iteration 250, Testing net (#0)
I0907 17:50:26.336319  2026 solver.cpp:406]     Test net output #0: accuracy
 = 0.6662
I0907 17:50:26.336613  2026 solver.cpp:406]     Test net output #1: loss = 0.
623816 (* 1 = 0.623816 loss)
I0907 17:50:27.358484  2026 solver.cpp:229] Iteration 260, loss = 0.69391
I0907 17:50:27.358542  2026 solver.cpp:245]     Train net output #0: loss =
 0.69391 (* 1 = 0.69391 loss)
I0907 17:50:27.358553  2026 sgd_solver.cpp:106] Iteration 260, lr = 0.01
I0907 17:50:29.352551  2026 solver.cpp:229] Iteration 280, loss = 0.637233
I0907 17:50:29.352607  2026 solver.cpp:245]     Train net output #0: loss =
 0.637233 (* 1 = 0.637233 loss)
I0907 17:50:29.352617  2026 sgd_solver.cpp:106] Iteration 280, lr = 0.01
I0907 17:50:31.346138  2026 solver.cpp:229] Iteration 300, loss = 0.537481
I0907 17:50:31.346195  2026 solver.cpp:245]     Train net output #0: loss =
 0.537481 (* 1 = 0.537481 loss)
I0907 17:50:31.346206  2026 sgd_solver.cpp:106] Iteration 300, lr = 0.01
```

```
I0907 17:50:33.340416  2026 solver.cpp:229] Iteration 320, loss = 0.630526
I0907 17:50:33.340476  2026 solver.cpp:245]     Train net output #0: loss =
 0.630526 (* 1 = 0.630526 loss)
I0907 17:50:33.340489  2026 sgd_solver.cpp:106] Iteration 320, lr = 0.01
I0907 17:50:35.330626  2026 solver.cpp:229] Iteration 340, loss = 0.591059
I0907 17:50:35.330682  2026 solver.cpp:245]     Train net output #0: loss =
 0.591059 (* 1 = 0.591059 loss)
I0907 17:50:35.330693  2026 sgd_solver.cpp:106] Iteration 340, lr = 0.01
I0907 17:50:37.322973  2026 solver.cpp:229] Iteration 360, loss = 0.593058
I0907 17:50:37.323031  2026 solver.cpp:245]     Train net output #0: loss =
 0.593058 (* 1 = 0.593058 loss)
I0907 17:50:37.323050  2026 sgd_solver.cpp:106] Iteration 360, lr = 0.01
I0907 17:50:39.315917  2026 solver.cpp:229] Iteration 380, loss = 0.61844
I0907 17:50:39.315976  2026 solver.cpp:245]     Train net output #0: loss =
 0.61844 (* 1 = 0.61844 loss)
I0907 17:50:39.315986  2026 sgd_solver.cpp:106] Iteration 380, lr = 0.01
I0907 17:50:41.309324  2026 solver.cpp:229] Iteration 400, loss = 0.690648
I0907 17:50:41.309386  2026 solver.cpp:245]     Train net output #0: loss =
 0.690648 (* 1 = 0.690648 loss)
I0907 17:50:41.309396  2026 sgd_solver.cpp:106] Iteration 400, lr = 0.01
I0907 17:50:43.301913  2026 solver.cpp:229] Iteration 420, loss = 0.62693
I0907 17:50:43.301976  2026 solver.cpp:245]     Train net output #0: loss =
 0.62693 (* 1 = 0.62693 loss)
I0907 17:50:43.301990  2026 sgd_solver.cpp:106] Iteration 420, lr = 0.01
I0907 17:50:45.294443  2026 solver.cpp:229] Iteration 440, loss = 0.600443
I0907 17:50:45.294498  2026 solver.cpp:245]     Train net output #0: loss =
 0.600443 (* 1 = 0.600443 loss)
I0907 17:50:45.294508  2026 sgd_solver.cpp:106] Iteration 440, lr = 0.01
I0907 17:50:47.288780  2026 solver.cpp:229] Iteration 460, loss = 0.700381
I0907 17:50:47.288833  2026 solver.cpp:245]     Train net output #0: loss =
 0.700381 (* 1 = 0.700381 loss)
I0907 17:50:47.288843  2026 sgd_solver.cpp:106] Iteration 460, lr = 0.01
I0907 17:50:49.281090  2026 solver.cpp:229] Iteration 480, loss = 0.555135
I0907 17:50:49.281150  2026 solver.cpp:245]     Train net output #0: loss =
 0.555135 (* 1 = 0.555135 loss)
I0907 17:50:49.281162  2026 sgd_solver.cpp:106] Iteration 480, lr = 0.01
I0907 17:50:51.175283  2026 solver.cpp:456] Snapshotting to binary proto file
 checkpoints/snapshot_iter_500.caffemodel
I0907 17:50:51.262532  2026 sgd_solver.cpp:273] Snapshotting solver state to
 binary proto file checkpoints/snapshot_iter_500.solverstate
I0907 17:50:51.267627  2026 solver.cpp:338] Iteration 500, Testing net (#0)
I0907 17:50:53.665534  2026 solver.cpp:406]     Test net output #0: accuracy
 = 0.661
I0907 17:50:53.665586  2026 solver.cpp:406]     Test net output #1: loss = 0.
628322 (* 1 = 0.628322 loss)
I0907 17:50:53.689847  2026 solver.cpp:229] Iteration 500, loss = 0.584232
I0907 17:50:53.689877  2026 solver.cpp:245]     Train net output #0: loss =
 0.584232 (* 1 = 0.584232 loss)
I0907 17:50:53.689888  2026 sgd_solver.cpp:106] Iteration 500, lr = 0.001
I0907 17:50:55.685356  2026 solver.cpp:229] Iteration 520, loss = 0.621351
I0907 17:50:55.685413  2026 solver.cpp:245]     Train net output #0: loss =
 0.621351 (* 1 = 0.621351 loss)
I0907 17:50:55.685462  2026 sgd_solver.cpp:106] Iteration 520, lr = 0.001
I0907 17:50:57.679355  2026 solver.cpp:229] Iteration 540, loss = 0.587776
I0907 17:50:57.679518  2026 solver.cpp:245]     Train net output #0: loss =
 0.587776 (* 1 = 0.587776 loss)
I0907 17:50:57.679530  2026 sgd_solver.cpp:106] Iteration 540, lr = 0.001
```

```
I0907 17:50:59.672078  2026 solver.cpp:229] Iteration 560, loss = 0.547018
I0907 17:50:59.672145  2026 solver.cpp:245]     Train net output #0: loss =
 0.547018 (* 1 = 0.547018 loss)
I0907 17:50:59.672155  2026 sgd_solver.cpp:106] Iteration 560, lr = 0.001
I0907 17:51:01.665143  2026 solver.cpp:229] Iteration 580, loss = 0.612046
I0907 17:51:01.665199  2026 solver.cpp:245]     Train net output #0: loss =
 0.612046 (* 1 = 0.612046 loss)
I0907 17:51:01.665210  2026 sgd_solver.cpp:106] Iteration 580, lr = 0.001
I0907 17:51:03.659373  2026 solver.cpp:229] Iteration 600, loss = 0.651342
I0907 17:51:03.659430  2026 solver.cpp:245]     Train net output #0: loss =
 0.651342 (* 1 = 0.651342 loss)
I0907 17:51:03.659441  2026 sgd_solver.cpp:106] Iteration 600, lr = 0.001
I0907 17:51:05.654263  2026 solver.cpp:229] Iteration 620, loss = 0.5456
I0907 17:51:05.654319  2026 solver.cpp:245]     Train net output #0: loss =
 0.5456 (* 1 = 0.5456 loss)
I0907 17:51:05.654328  2026 sgd_solver.cpp:106] Iteration 620, lr = 0.001
I0907 17:51:07.647342  2026 solver.cpp:229] Iteration 640, loss = 0.616289
I0907 17:51:07.647402  2026 solver.cpp:245]     Train net output #0: loss =
 0.616289 (* 1 = 0.616289 loss)
I0907 17:51:07.647413  2026 sgd_solver.cpp:106] Iteration 640, lr = 0.001
I0907 17:51:09.638375  2026 solver.cpp:229] Iteration 660, loss = 0.570817
I0907 17:51:09.638432  2026 solver.cpp:245]     Train net output #0: loss =
 0.570817 (* 1 = 0.570817 loss)
I0907 17:51:09.638442  2026 sgd_solver.cpp:106] Iteration 660, lr = 0.001
I0907 17:51:11.631057  2026 solver.cpp:229] Iteration 680, loss = 0.550111
I0907 17:51:11.631124  2026 solver.cpp:245]     Train net output #0: loss =
 0.550111 (* 1 = 0.550111 loss)
I0907 17:51:11.631135  2026 sgd_solver.cpp:106] Iteration 680, lr = 0.001
I0907 17:51:13.623296  2026 solver.cpp:229] Iteration 700, loss = 0.54983
I0907 17:51:13.623353  2026 solver.cpp:245]     Train net output #0: loss =
 0.54983 (* 1 = 0.54983 loss)
I0907 17:51:13.623364  2026 sgd_solver.cpp:106] Iteration 700, lr = 0.001
I0907 17:51:15.616344  2026 solver.cpp:229] Iteration 720, loss = 0.565444
I0907 17:51:15.616401  2026 solver.cpp:245]     Train net output #0: loss =
 0.565444 (* 1 = 0.565444 loss)
I0907 17:51:15.616411  2026 sgd_solver.cpp:106] Iteration 720, lr = 0.001
I0907 17:51:17.612784  2026 solver.cpp:229] Iteration 740, loss = 0.522849
I0907 17:51:17.612841  2026 solver.cpp:245]     Train net output #0: loss =
 0.522849 (* 1 = 0.522849 loss)
I0907 17:51:17.612851  2026 sgd_solver.cpp:106] Iteration 740, lr = 0.001
I0907 17:51:18.510428  2026 solver.cpp:456] Snapshotting to binary proto file
 checkpoints/snapshot_iter_750.caffemodel
I0907 17:51:18.597537  2026 sgd_solver.cpp:273] Snapshotting solver state to
 binary proto file checkpoints/snapshot_iter_750.solverstate
I0907 17:51:18.602501  2026 solver.cpp:338] Iteration 750, Testing net (#0)
I0907 17:51:21.004096  2026 solver.cpp:406]     Test net output #0: accuracy
 = 0.7128
I0907 17:51:21.004165  2026 solver.cpp:406]     Test net output #1: loss = 0.
571076 (* 1 = 0.571076 loss)
I0907 17:51:21.004176  2026 solver.cpp:323] Optimization Done.
I0907 17:51:21.004181  2026 caffe.cpp:216] Optimization Done.
```

We noticed significant performance improvements. The accuracy is now about 71% and we are underfitting less.

Refinement #2

Let's make one more modification to the network before training a final time. In the last modification we increased the number of neurons in our convolutional layers. Another way to increase the number of trainable parameters in our network is to make it deeper by adding more layers. This time we add a new fully-connected layer with 500 outputs - call it `ip2`. This layers should come after the third pooling layer, `pool3`, but **before** the existing fully-connected layer `ip1`. In Caffe, fully-connected layers are implemented using the inner product layer construct. After the new fully-connected (inner product) layer you also need a ReLU activation layer and a dropout layer. The dropout layer will prevent the network from *overfitting*, i.e. getting really good at classifying the training data but not able to classify the validation data.

Again, this model will take slightly longer to train due to the increased size. It's about 3 minutes and 20 seconds.

In [10]:
```
%%bash
TOOLS=/home/ubuntu/caffe/build/tools
#Train your modified network configuration
$TOOLS/caffe train -gpu 0 -solver src/solver3.prototxt
```

```
I0907 17:52:18.155535  2034 upgrade_proto.cpp:990] Attempting to upgrade inpu
t file specified using deprecated 'solver_type' field (enum)': src/solver3.pr
ototxt
I0907 17:52:18.155802  2034 upgrade_proto.cpp:997] Successfully upgraded file
 specified using deprecated 'solver_type' field (enum) to 'type' field (strin
g).
W0907 17:52:18.155814  2034 upgrade_proto.cpp:999] Note that future Caffe rel
eases will only support 'type' field (string) for a solver's type.
I0907 17:52:18.155977  2034 caffe.cpp:185] Using GPUs 0
I0907 17:52:18.290024  2034 solver.cpp:48] Initializing solver from parameter
s:
test_iter: 100
test_interval: 250
base_lr: 0.01
display: 20
max_iter: 750
lr_policy: "step"
gamma: 0.1
momentum: 0.9
stepsize: 500
snapshot: 750
snapshot_prefix: "checkpoints/snapshot"
solver_mode: GPU
device_id: 0
random_seed: 1234
net: "src/train_val3.answer.prototxt"
type: "SGD"
I0907 17:52:18.290211  2034 solver.cpp:91] Creating training net from net fil
e: src/train_val3.answer.prototxt
I0907 17:52:18.290791  2034 net.cpp:322] The NetState phase (0) differed from
 the phase (1) specified by a rule in layer data
I0907 17:52:18.290827  2034 net.cpp:322] The NetState phase (0) differed from
 the phase (1) specified by a rule in layer accuracy
I0907 17:52:18.290977  2034 net.cpp:49] Initializing net from parameters:
state {
  phase: TRAIN
}
layer {
  name: "data"
  type: "Data"
  top: "data"
  top: "label"
  include {
    phase: TRAIN
  }
  transform_param {
    mirror: true
    crop_size: 32
    mean_file: "mean.binaryproto"
  }
  data_param {
    source: "train_lmdb"
    batch_size: 100
    backend: LMDB
  }
}
layer {
```

```
  name: "conv1"
  type: "Convolution"
  bottom: "data"
  top: "conv1"
  param {
    lr_mult: 1
  }
  param {
    lr_mult: 2
  }
  convolution_param {
    num_output: 64
    pad: 2
    kernel_size: 5
    stride: 1
    weight_filler {
      type: "gaussian"
      std: 0.0001
    }
    bias_filler {
      type: "constant"
    }
  }
}
layer {
  name: "pool1"
  type: "Pooling"
  bottom: "conv1"
  top: "pool1"
  pooling_param {
    pool: MAX
    kernel_size: 3
    stride: 2
  }
}
layer {
  name: "relu1"
  type: "ReLU"
  bottom: "pool1"
  top: "pool1"
}
layer {
  name: "conv2"
  type: "Convolution"
  bottom: "pool1"
  top: "conv2"
  param {
    lr_mult: 1
  }
  param {
    lr_mult: 2
  }
  convolution_param {
    num_output: 128
    pad: 2
    kernel_size: 5
    stride: 1
```

```
    weight_filler {
      type: "gaussian"
      std: 0.01
    }
    bias_filler {
      type: "constant"
    }
  }
}
layer {
  name: "relu2"
  type: "ReLU"
  bottom: "conv2"
  top: "conv2"
}
layer {
  name: "pool2"
  type: "Pooling"
  bottom: "conv2"
  top: "pool2"
  pooling_param {
    pool: AVE
    kernel_size: 3
    stride: 2
  }
}
layer {
  name: "conv3"
  type: "Convolution"
  bottom: "pool2"
  top: "conv3"
  convolution_param {
    num_output: 128
    pad: 2
    kernel_size: 5
    stride: 1
    weight_filler {
      type: "gaussian"
      std: 0.01
    }
    bias_filler {
      type: "constant"
    }
  }
}
layer {
  name: "relu3"
  type: "ReLU"
  bottom: "conv3"
  top: "conv3"
}
layer {
  name: "pool3"
  type: "Pooling"
  bottom: "conv3"
  top: "pool3"
  pooling_param {
```

```
    pool: AVE
    kernel_size: 3
    stride: 2
  }
}
layer {
  name: "ip2"
  type: "InnerProduct"
  bottom: "pool3"
  top: "ip2"
  param {
    lr_mult: 1
    decay_mult: 1
  }
  param {
    lr_mult: 2
    decay_mult: 0
  }
  inner_product_param {
    num_output: 500
    weight_filler {
      type: "gaussian"
      std: 0.005
    }
    bias_filler {
      type: "constant"
      value: 0.1
    }
  }
}
layer {
  name: "reluip2"
  type: "ReLU"
  bottom: "ip2"
  top: "ip2"
}
layer {
  name: "dropip2"
  type: "Dropout"
  bottom: "ip2"
  top: "ip2"
  dropout_param {
    dropout_ratio: 0.5
  }
}
layer {
  name: "ip1"
  type: "InnerProduct"
  bottom: "ip2"
  top: "ip1"
  param {
    lr_mult: 1
    decay_mult: 250
  }
  param {
    lr_mult: 2
    decay_mult: 0
```

```
  }
  inner_product_param {
    num_output: 2
    weight_filler {
      type: "gaussian"
      std: 0.01
    }
    bias_filler {
      type: "constant"
    }
  }
}
layer {
  name: "loss"
  type: "SoftmaxWithLoss"
  bottom: "ip1"
  bottom: "label"
  top: "loss"
}
I0907 17:52:18.291097  2034 layer_factory.hpp:77] Creating layer data
I0907 17:52:18.291326  2034 net.cpp:106] Creating Layer data
I0907 17:52:18.291352  2034 net.cpp:411] data -> data
I0907 17:52:18.291399  2034 net.cpp:411] data -> label
I0907 17:52:18.291429  2034 data_transformer.cpp:25] Loading mean file from:
 mean.binaryproto
I0907 17:52:18.292052  2037 db_lmdb.cpp:38] Opened lmdb train_lmdb
I0907 17:52:18.303288  2034 data_layer.cpp:41] output data size: 100,3,32,32
I0907 17:52:18.306143  2034 net.cpp:150] Setting up data
I0907 17:52:18.306174  2034 net.cpp:157] Top shape: 100 3 32 32 (307200)
I0907 17:52:18.306185  2034 net.cpp:157] Top shape: 100 (100)
I0907 17:52:18.306190  2034 net.cpp:165] Memory required for data: 1229200
I0907 17:52:18.306201  2034 layer_factory.hpp:77] Creating layer conv1
I0907 17:52:18.306229  2034 net.cpp:106] Creating Layer conv1
I0907 17:52:18.306246  2034 net.cpp:454] conv1 <- data
I0907 17:52:18.306267  2034 net.cpp:411] conv1 -> conv1
I0907 17:52:18.424154  2034 net.cpp:150] Setting up conv1
I0907 17:52:18.424204  2034 net.cpp:157] Top shape: 100 64 32 32 (6553600)
I0907 17:52:18.424212  2034 net.cpp:165] Memory required for data: 27443600
I0907 17:52:18.424240  2034 layer_factory.hpp:77] Creating layer pool1
I0907 17:52:18.424263  2034 net.cpp:106] Creating Layer pool1
I0907 17:52:18.424278  2034 net.cpp:454] pool1 <- conv1
I0907 17:52:18.424288  2034 net.cpp:411] pool1 -> pool1
I0907 17:52:18.424353  2034 net.cpp:150] Setting up pool1
I0907 17:52:18.424371  2034 net.cpp:157] Top shape: 100 64 16 16 (1638400)
I0907 17:52:18.424376  2034 net.cpp:165] Memory required for data: 33997200
I0907 17:52:18.424381  2034 layer_factory.hpp:77] Creating layer relu1
I0907 17:52:18.424393  2034 net.cpp:106] Creating Layer relu1
I0907 17:52:18.424401  2034 net.cpp:454] relu1 <- pool1
I0907 17:52:18.424408  2034 net.cpp:397] relu1 -> pool1 (in-place)
I0907 17:52:18.424556  2034 net.cpp:150] Setting up relu1
I0907 17:52:18.424574  2034 net.cpp:157] Top shape: 100 64 16 16 (1638400)
I0907 17:52:18.424579  2034 net.cpp:165] Memory required for data: 40550800
I0907 17:52:18.424590  2034 layer_factory.hpp:77] Creating layer conv2
I0907 17:52:18.424609  2034 net.cpp:106] Creating Layer conv2
I0907 17:52:18.424619  2034 net.cpp:454] conv2 <- pool1
I0907 17:52:18.424628  2034 net.cpp:411] conv2 -> conv2
I0907 17:52:18.432421  2034 net.cpp:150] Setting up conv2
```

```
I0907 17:52:18.432446  2034 net.cpp:157] Top shape: 100 128 16 16 (3276800)
I0907 17:52:18.432452  2034 net.cpp:165] Memory required for data: 53658000
I0907 17:52:18.432476  2034 layer_factory.hpp:77] Creating layer relu2
I0907 17:52:18.432487  2034 net.cpp:106] Creating Layer relu2
I0907 17:52:18.432497  2034 net.cpp:454] relu2 <- conv2
I0907 17:52:18.432503  2034 net.cpp:397] relu2 -> conv2 (in-place)
I0907 17:52:18.432729  2034 net.cpp:150] Setting up relu2
I0907 17:52:18.432750  2034 net.cpp:157] Top shape: 100 128 16 16 (3276800)
I0907 17:52:18.432756  2034 net.cpp:165] Memory required for data: 66765200
I0907 17:52:18.432772  2034 layer_factory.hpp:77] Creating layer pool2
I0907 17:52:18.432781  2034 net.cpp:106] Creating Layer pool2
I0907 17:52:18.432792  2034 net.cpp:454] pool2 <- conv2
I0907 17:52:18.432801  2034 net.cpp:411] pool2 -> pool2
I0907 17:52:18.432976  2034 net.cpp:150] Setting up pool2
I0907 17:52:18.432997  2034 net.cpp:157] Top shape: 100 128 8 8 (819200)
I0907 17:52:18.433006  2034 net.cpp:165] Memory required for data: 70042000
I0907 17:52:18.433012  2034 layer_factory.hpp:77] Creating layer conv3
I0907 17:52:18.433025  2034 net.cpp:106] Creating Layer conv3
I0907 17:52:18.433037  2034 net.cpp:454] conv3 <- pool2
I0907 17:52:18.433045  2034 net.cpp:411] conv3 -> conv3
I0907 17:52:18.447526  2034 net.cpp:150] Setting up conv3
I0907 17:52:18.447549  2034 net.cpp:157] Top shape: 100 128 8 8 (819200)
I0907 17:52:18.447556  2034 net.cpp:165] Memory required for data: 73318800
I0907 17:52:18.447571  2034 layer_factory.hpp:77] Creating layer relu3
I0907 17:52:18.447597  2034 net.cpp:106] Creating Layer relu3
I0907 17:52:18.447609  2034 net.cpp:454] relu3 <- conv3
I0907 17:52:18.447655  2034 net.cpp:397] relu3 -> conv3 (in-place)
I0907 17:52:18.447814  2034 net.cpp:150] Setting up relu3
I0907 17:52:18.447834  2034 net.cpp:157] Top shape: 100 128 8 8 (819200)
I0907 17:52:18.447839  2034 net.cpp:165] Memory required for data: 76595600
I0907 17:52:18.447845  2034 layer_factory.hpp:77] Creating layer pool3
I0907 17:52:18.447854  2034 net.cpp:106] Creating Layer pool3
I0907 17:52:18.447865  2034 net.cpp:454] pool3 <- conv3
I0907 17:52:18.447875  2034 net.cpp:411] pool3 -> pool3
I0907 17:52:18.448151  2034 net.cpp:150] Setting up pool3
I0907 17:52:18.448171  2034 net.cpp:157] Top shape: 100 128 4 4 (204800)
I0907 17:52:18.448177  2034 net.cpp:165] Memory required for data: 77414800
I0907 17:52:18.448185  2034 layer_factory.hpp:77] Creating layer ip2
I0907 17:52:18.448199  2034 net.cpp:106] Creating Layer ip2
I0907 17:52:18.448210  2034 net.cpp:454] ip2 <- pool3
I0907 17:52:18.448221  2034 net.cpp:411] ip2 -> ip2
I0907 17:52:18.482381  2034 net.cpp:150] Setting up ip2
I0907 17:52:18.482406  2034 net.cpp:157] Top shape: 100 500 (50000)
I0907 17:52:18.482412  2034 net.cpp:165] Memory required for data: 77614800
I0907 17:52:18.482421  2034 layer_factory.hpp:77] Creating layer reluip2
I0907 17:52:18.482439  2034 net.cpp:106] Creating Layer reluip2
I0907 17:52:18.482455  2034 net.cpp:454] reluip2 <- ip2
I0907 17:52:18.482463  2034 net.cpp:397] reluip2 -> ip2 (in-place)
I0907 17:52:18.482728  2034 net.cpp:150] Setting up reluip2
I0907 17:52:18.482748  2034 net.cpp:157] Top shape: 100 500 (50000)
I0907 17:52:18.482754  2034 net.cpp:165] Memory required for data: 77814800
I0907 17:52:18.482761  2034 layer_factory.hpp:77] Creating layer dropip2
I0907 17:52:18.482774  2034 net.cpp:106] Creating Layer dropip2
I0907 17:52:18.482784  2034 net.cpp:454] dropip2 <- ip2
I0907 17:52:18.482795  2034 net.cpp:397] dropip2 -> ip2 (in-place)
I0907 17:52:18.482841  2034 net.cpp:150] Setting up dropip2
I0907 17:52:18.482857  2034 net.cpp:157] Top shape: 100 500 (50000)
```

```
I0907 17:52:18.482866  2034 net.cpp:165] Memory required for data: 78014800
I0907 17:52:18.482872  2034 layer_factory.hpp:77] Creating layer ip1
I0907 17:52:18.482882  2034 net.cpp:106] Creating Layer ip1
I0907 17:52:18.482892  2034 net.cpp:454] ip1 <- ip2
I0907 17:52:18.482900  2034 net.cpp:411] ip1 -> ip1
I0907 17:52:18.483049  2034 net.cpp:150] Setting up ip1
I0907 17:52:18.483067  2034 net.cpp:157] Top shape: 100 2 (200)
I0907 17:52:18.483072  2034 net.cpp:165] Memory required for data: 78015600
I0907 17:52:18.483088  2034 layer_factory.hpp:77] Creating layer loss
I0907 17:52:18.483114  2034 net.cpp:106] Creating Layer loss
I0907 17:52:18.483125  2034 net.cpp:454] loss <- ip1
I0907 17:52:18.483131  2034 net.cpp:454] loss <- label
I0907 17:52:18.483144  2034 net.cpp:411] loss -> loss
I0907 17:52:18.483165  2034 layer_factory.hpp:77] Creating layer loss
I0907 17:52:18.483433  2034 net.cpp:150] Setting up loss
I0907 17:52:18.483451  2034 net.cpp:157] Top shape: (1)
I0907 17:52:18.483456  2034 net.cpp:160]     with loss weight 1
I0907 17:52:18.483489  2034 net.cpp:165] Memory required for data: 78015604
I0907 17:52:18.483496  2034 net.cpp:226] loss needs backward computation.
I0907 17:52:18.483502  2034 net.cpp:226] ip1 needs backward computation.
I0907 17:52:18.483508  2034 net.cpp:226] dropip2 needs backward computation.
I0907 17:52:18.483512  2034 net.cpp:226] reluip2 needs backward computation.
I0907 17:52:18.483518  2034 net.cpp:226] ip2 needs backward computation.
I0907 17:52:18.483523  2034 net.cpp:226] pool3 needs backward computation.
I0907 17:52:18.483528  2034 net.cpp:226] relu3 needs backward computation.
I0907 17:52:18.483533  2034 net.cpp:226] conv3 needs backward computation.
I0907 17:52:18.483539  2034 net.cpp:226] pool2 needs backward computation.
I0907 17:52:18.483543  2034 net.cpp:226] relu2 needs backward computation.
I0907 17:52:18.483549  2034 net.cpp:226] conv2 needs backward computation.
I0907 17:52:18.483553  2034 net.cpp:226] relu1 needs backward computation.
I0907 17:52:18.483559  2034 net.cpp:226] pool1 needs backward computation.
I0907 17:52:18.483595  2034 net.cpp:226] conv1 needs backward computation.
I0907 17:52:18.483610  2034 net.cpp:228] data does not need backward computat
ion.
I0907 17:52:18.483616  2034 net.cpp:270] This network produces output loss
I0907 17:52:18.483631  2034 net.cpp:283] Network initialization done.
I0907 17:52:18.484194  2034 solver.cpp:181] Creating test net (#0) specified
 by net file: src/train_val3.answer.prototxt
I0907 17:52:18.484259  2034 net.cpp:322] The NetState phase (1) differed from
 the phase (0) specified by a rule in layer data
I0907 17:52:18.484424  2034 net.cpp:49] Initializing net from parameters:
state {
  phase: TEST
}
layer {
  name: "data"
  type: "Data"
  top: "data"
  top: "label"
  include {
    phase: TEST
  }
  transform_param {
    crop_size: 32
    mean_file: "mean.binaryproto"
  }
  data_param {
```

```
    source: "val_lmdb"
    batch_size: 100
    backend: LMDB
  }
}
layer {
  name: "conv1"
  type: "Convolution"
  bottom: "data"
  top: "conv1"
  param {
    lr_mult: 1
  }
  param {
    lr_mult: 2
  }
  convolution_param {
    num_output: 64
    pad: 2
    kernel_size: 5
    stride: 1
    weight_filler {
      type: "gaussian"
      std: 0.0001
    }
    bias_filler {
      type: "constant"
    }
  }
}
layer {
  name: "pool1"
  type: "Pooling"
  bottom: "conv1"
  top: "pool1"
  pooling_param {
    pool: MAX
    kernel_size: 3
    stride: 2
  }
}
layer {
  name: "relu1"
  type: "ReLU"
  bottom: "pool1"
  top: "pool1"
}
layer {
  name: "conv2"
  type: "Convolution"
  bottom: "pool1"
  top: "conv2"
  param {
    lr_mult: 1
  }
  param {
    lr_mult: 2
```

```
    }
    convolution_param {
      num_output: 128
      pad: 2
      kernel_size: 5
      stride: 1
      weight_filler {
        type: "gaussian"
        std: 0.01
      }
      bias_filler {
        type: "constant"
      }
    }
}
layer {
  name: "relu2"
  type: "ReLU"
  bottom: "conv2"
  top: "conv2"
}
layer {
  name: "pool2"
  type: "Pooling"
  bottom: "conv2"
  top: "pool2"
  pooling_param {
    pool: AVE
    kernel_size: 3
    stride: 2
  }
}
layer {
  name: "conv3"
  type: "Convolution"
  bottom: "pool2"
  top: "conv3"
  convolution_param {
    num_output: 128
    pad: 2
    kernel_size: 5
    stride: 1
    weight_filler {
      type: "gaussian"
      std: 0.01
    }
    bias_filler {
      type: "constant"
    }
  }
}
layer {
  name: "relu3"
  type: "ReLU"
  bottom: "conv3"
  top: "conv3"
}
```

```
layer {
  name: "pool3"
  type: "Pooling"
  bottom: "conv3"
  top: "pool3"
  pooling_param {
    pool: AVE
    kernel_size: 3
    stride: 2
  }
}
layer {
  name: "ip2"
  type: "InnerProduct"
  bottom: "pool3"
  top: "ip2"
  param {
    lr_mult: 1
    decay_mult: 1
  }
  param {
    lr_mult: 2
    decay_mult: 0
  }
  inner_product_param {
    num_output: 500
    weight_filler {
      type: "gaussian"
      std: 0.005
    }
    bias_filler {
      type: "constant"
      value: 0.1
    }
  }
}
layer {
  name: "reluip2"
  type: "ReLU"
  bottom: "ip2"
  top: "ip2"
}
layer {
  name: "dropip2"
  type: "Dropout"
  bottom: "ip2"
  top: "ip2"
  dropout_param {
    dropout_ratio: 0.5
  }
}
layer {
  name: "ip1"
  type: "InnerProduct"
  bottom: "ip2"
  top: "ip1"
  param {
```

```
    lr_mult: 1
    decay_mult: 250
  }
  param {
    lr_mult: 2
    decay_mult: 0
  }
  inner_product_param {
    num_output: 2
    weight_filler {
      type: "gaussian"
      std: 0.01
    }
    bias_filler {
      type: "constant"
    }
  }
}
layer {
  name: "loss"
  type: "SoftmaxWithLoss"
  bottom: "ip1"
  bottom: "label"
  top: "loss"
}
layer {
  name: "accuracy"
  type: "Accuracy"
  bottom: "ip1"
  bottom: "label"
  top: "accuracy"
  include {
    phase: TEST
  }
}
I0907 17:52:18.484539  2034 layer_factory.hpp:77] Creating layer data
I0907 17:52:18.484683  2034 net.cpp:106] Creating Layer data
I0907 17:52:18.484700  2034 net.cpp:411] data -> data
I0907 17:52:18.484714  2034 net.cpp:411] data -> label
I0907 17:52:18.484736  2034 data_transformer.cpp:25] Loading mean file from:
 mean.binaryproto
I0907 17:52:18.485389  2039 db_lmdb.cpp:38] Opened lmdb val_lmdb
I0907 17:52:18.485507  2034 data_layer.cpp:41] output data size: 100,3,32,32
I0907 17:52:18.488396  2034 net.cpp:150] Setting up data
I0907 17:52:18.488420  2034 net.cpp:157] Top shape: 100 3 32 32 (307200)
I0907 17:52:18.488427  2034 net.cpp:157] Top shape: 100 (100)
I0907 17:52:18.488432  2034 net.cpp:165] Memory required for data: 1229200
I0907 17:52:18.488437  2034 layer_factory.hpp:77] Creating layer label_data_1
_split
I0907 17:52:18.488450  2034 net.cpp:106] Creating Layer label_data_1_split
I0907 17:52:18.488456  2034 net.cpp:454] label_data_1_split <- label
I0907 17:52:18.488467  2034 net.cpp:411] label_data_1_split -> label_data_1_s
plit_0
I0907 17:52:18.488477  2034 net.cpp:411] label_data_1_split -> label_data_1_s
plit_1
I0907 17:52:18.488661  2034 net.cpp:150] Setting up label_data_1_split
I0907 17:52:18.488688  2034 net.cpp:157] Top shape: 100 (100)
```

```
I0907 17:52:18.488695  2034 net.cpp:157] Top shape: 100 (100)
I0907 17:52:18.488699  2034 net.cpp:165] Memory required for data: 1230000
I0907 17:52:18.488704  2034 layer_factory.hpp:77] Creating layer conv1
I0907 17:52:18.488719  2034 net.cpp:106] Creating Layer conv1
I0907 17:52:18.488731  2034 net.cpp:454] conv1 <- data
I0907 17:52:18.488744  2034 net.cpp:411] conv1 -> conv1
I0907 17:52:18.490237  2034 net.cpp:150] Setting up conv1
I0907 17:52:18.490260  2034 net.cpp:157] Top shape: 100 64 32 32 (6553600)
I0907 17:52:18.490267  2034 net.cpp:165] Memory required for data: 27444400
I0907 17:52:18.490279  2034 layer_factory.hpp:77] Creating layer pool1
I0907 17:52:18.490293  2034 net.cpp:106] Creating Layer pool1
I0907 17:52:18.490299  2034 net.cpp:454] pool1 <- conv1
I0907 17:52:18.490308  2034 net.cpp:411] pool1 -> pool1
I0907 17:52:18.490356  2034 net.cpp:150] Setting up pool1
I0907 17:52:18.490373  2034 net.cpp:157] Top shape: 100 64 16 16 (1638400)
I0907 17:52:18.490378  2034 net.cpp:165] Memory required for data: 33998000
I0907 17:52:18.490384  2034 layer_factory.hpp:77] Creating layer relu1
I0907 17:52:18.490396  2034 net.cpp:106] Creating Layer relu1
I0907 17:52:18.490402  2034 net.cpp:454] relu1 <- pool1
I0907 17:52:18.490409  2034 net.cpp:397] relu1 -> pool1 (in-place)
I0907 17:52:18.490564  2034 net.cpp:150] Setting up relu1
I0907 17:52:18.490583  2034 net.cpp:157] Top shape: 100 64 16 16 (1638400)
I0907 17:52:18.490589  2034 net.cpp:165] Memory required for data: 40551600
I0907 17:52:18.490594  2034 layer_factory.hpp:77] Creating layer conv2
I0907 17:52:18.490608  2034 net.cpp:106] Creating Layer conv2
I0907 17:52:18.490617  2034 net.cpp:454] conv2 <- pool1
I0907 17:52:18.490628  2034 net.cpp:411] conv2 -> conv2
I0907 17:52:18.498924  2034 net.cpp:150] Setting up conv2
I0907 17:52:18.498950  2034 net.cpp:157] Top shape: 100 128 16 16 (3276800)
I0907 17:52:18.498955  2034 net.cpp:165] Memory required for data: 53658800
I0907 17:52:18.498970  2034 layer_factory.hpp:77] Creating layer relu2
I0907 17:52:18.498991  2034 net.cpp:106] Creating Layer relu2
I0907 17:52:18.499001  2034 net.cpp:454] relu2 <- conv2
I0907 17:52:18.499008  2034 net.cpp:397] relu2 -> conv2 (in-place)
I0907 17:52:18.499266  2034 net.cpp:150] Setting up relu2
I0907 17:52:18.499287  2034 net.cpp:157] Top shape: 100 128 16 16 (3276800)
I0907 17:52:18.499294  2034 net.cpp:165] Memory required for data: 66766000
I0907 17:52:18.499300  2034 layer_factory.hpp:77] Creating layer pool2
I0907 17:52:18.499311  2034 net.cpp:106] Creating Layer pool2
I0907 17:52:18.499321  2034 net.cpp:454] pool2 <- conv2
I0907 17:52:18.499330  2034 net.cpp:411] pool2 -> pool2
I0907 17:52:18.499505  2034 net.cpp:150] Setting up pool2
I0907 17:52:18.499524  2034 net.cpp:157] Top shape: 100 128 8 8 (819200)
I0907 17:52:18.499534  2034 net.cpp:165] Memory required for data: 70042800
I0907 17:52:18.499544  2034 layer_factory.hpp:77] Creating layer conv3
I0907 17:52:18.499562  2034 net.cpp:106] Creating Layer conv3
I0907 17:52:18.499573  2034 net.cpp:454] conv3 <- pool2
I0907 17:52:18.499608  2034 net.cpp:411] conv3 -> conv3
I0907 17:52:18.514439  2034 net.cpp:150] Setting up conv3
I0907 17:52:18.514464  2034 net.cpp:157] Top shape: 100 128 8 8 (819200)
I0907 17:52:18.514470  2034 net.cpp:165] Memory required for data: 73319600
I0907 17:52:18.514513  2034 layer_factory.hpp:77] Creating layer relu3
I0907 17:52:18.514526  2034 net.cpp:106] Creating Layer relu3
I0907 17:52:18.514539  2034 net.cpp:454] relu3 <- conv3
I0907 17:52:18.514546  2034 net.cpp:397] relu3 -> conv3 (in-place)
I0907 17:52:18.514808  2034 net.cpp:150] Setting up relu3
I0907 17:52:18.514829  2034 net.cpp:157] Top shape: 100 128 8 8 (819200)
```

```
I0907 17:52:18.514835  2034 net.cpp:165] Memory required for data: 76596400
I0907 17:52:18.514842  2034 layer_factory.hpp:77] Creating layer pool3
I0907 17:52:18.514852  2034 net.cpp:106] Creating Layer pool3
I0907 17:52:18.514861  2034 net.cpp:454] pool3 <- conv3
I0907 17:52:18.514873  2034 net.cpp:411] pool3 -> pool3
I0907 17:52:18.515054  2034 net.cpp:150] Setting up pool3
I0907 17:52:18.515074  2034 net.cpp:157] Top shape: 100 128 4 4 (204800)
I0907 17:52:18.515079  2034 net.cpp:165] Memory required for data: 77415600
I0907 17:52:18.515085  2034 layer_factory.hpp:77] Creating layer ip2
I0907 17:52:18.515095  2034 net.cpp:106] Creating Layer ip2
I0907 17:52:18.515106  2034 net.cpp:454] ip2 <- pool3
I0907 17:52:18.515117  2034 net.cpp:411] ip2 -> ip2
I0907 17:52:18.549113  2034 net.cpp:150] Setting up ip2
I0907 17:52:18.549139  2034 net.cpp:157] Top shape: 100 500 (50000)
I0907 17:52:18.549144  2034 net.cpp:165] Memory required for data: 77615600
I0907 17:52:18.549154  2034 layer_factory.hpp:77] Creating layer reluip2
I0907 17:52:18.549168  2034 net.cpp:106] Creating Layer reluip2
I0907 17:52:18.549175  2034 net.cpp:454] reluip2 <- ip2
I0907 17:52:18.549183  2034 net.cpp:397] reluip2 -> ip2 (in-place)
I0907 17:52:18.549437  2034 net.cpp:150] Setting up reluip2
I0907 17:52:18.549458  2034 net.cpp:157] Top shape: 100 500 (50000)
I0907 17:52:18.549463  2034 net.cpp:165] Memory required for data: 77815600
I0907 17:52:18.549470  2034 layer_factory.hpp:77] Creating layer dropip2
I0907 17:52:18.549479  2034 net.cpp:106] Creating Layer dropip2
I0907 17:52:18.549489  2034 net.cpp:454] dropip2 <- ip2
I0907 17:52:18.549500  2034 net.cpp:397] dropip2 -> ip2 (in-place)
I0907 17:52:18.549540  2034 net.cpp:150] Setting up dropip2
I0907 17:52:18.549558  2034 net.cpp:157] Top shape: 100 500 (50000)
I0907 17:52:18.549563  2034 net.cpp:165] Memory required for data: 78015600
I0907 17:52:18.549576  2034 layer_factory.hpp:77] Creating layer ip1
I0907 17:52:18.549587  2034 net.cpp:106] Creating Layer ip1
I0907 17:52:18.549597  2034 net.cpp:454] ip1 <- ip2
I0907 17:52:18.549608  2034 net.cpp:411] ip1 -> ip1
I0907 17:52:18.549767  2034 net.cpp:150] Setting up ip1
I0907 17:52:18.549784  2034 net.cpp:157] Top shape: 100 2 (200)
I0907 17:52:18.549789  2034 net.cpp:165] Memory required for data: 78016400
I0907 17:52:18.549803  2034 layer_factory.hpp:77] Creating layer ip1_ip1_0_sp
lit
I0907 17:52:18.549818  2034 net.cpp:106] Creating Layer ip1_ip1_0_split
I0907 17:52:18.549824  2034 net.cpp:454] ip1_ip1_0_split <- ip1
I0907 17:52:18.549839  2034 net.cpp:411] ip1_ip1_0_split -> ip1_ip1_0_split_0
I0907 17:52:18.549849  2034 net.cpp:411] ip1_ip1_0_split -> ip1_ip1_0_split_1
I0907 17:52:18.549897  2034 net.cpp:150] Setting up ip1_ip1_0_split
I0907 17:52:18.549916  2034 net.cpp:157] Top shape: 100 2 (200)
I0907 17:52:18.549921  2034 net.cpp:157] Top shape: 100 2 (200)
I0907 17:52:18.549927  2034 net.cpp:165] Memory required for data: 78018000
I0907 17:52:18.549932  2034 layer_factory.hpp:77] Creating layer loss
I0907 17:52:18.549947  2034 net.cpp:106] Creating Layer loss
I0907 17:52:18.549953  2034 net.cpp:454] loss <- ip1_ip1_0_split_0
I0907 17:52:18.549960  2034 net.cpp:454] loss <- label_data_1_split_0
I0907 17:52:18.549968  2034 net.cpp:411] loss -> loss
I0907 17:52:18.549983  2034 layer_factory.hpp:77] Creating layer loss
I0907 17:52:18.550222  2034 net.cpp:150] Setting up loss
I0907 17:52:18.550245  2034 net.cpp:157] Top shape: (1)
I0907 17:52:18.550249  2034 net.cpp:160]     with loss weight 1
I0907 17:52:18.550269  2034 net.cpp:165] Memory required for data: 78018004
I0907 17:52:18.550276  2034 layer_factory.hpp:77] Creating layer accuracy
```

```
I0907 17:52:18.550307  2034 net.cpp:106] Creating Layer accuracy
I0907 17:52:18.550318  2034 net.cpp:454] accuracy <- ip1_ip1_0_split_1
I0907 17:52:18.550325  2034 net.cpp:454] accuracy <- label_data_1_split_1
I0907 17:52:18.550334  2034 net.cpp:411] accuracy -> accuracy
I0907 17:52:18.550349  2034 net.cpp:150] Setting up accuracy
I0907 17:52:18.550362  2034 net.cpp:157] Top shape: (1)
I0907 17:52:18.550367  2034 net.cpp:165] Memory required for data: 78018008
I0907 17:52:18.550372  2034 net.cpp:228] accuracy does not need backward comp
utation.
I0907 17:52:18.550379  2034 net.cpp:226] loss needs backward computation.
I0907 17:52:18.550384  2034 net.cpp:226] ip1_ip1_0_split needs backward compu
tation.
I0907 17:52:18.550390  2034 net.cpp:226] ip1 needs backward computation.
I0907 17:52:18.550395  2034 net.cpp:226] dropip2 needs backward computation.
I0907 17:52:18.550401  2034 net.cpp:226] reluip2 needs backward computation.
I0907 17:52:18.550405  2034 net.cpp:226] ip2 needs backward computation.
I0907 17:52:18.550411  2034 net.cpp:226] pool3 needs backward computation.
I0907 17:52:18.550415  2034 net.cpp:226] relu3 needs backward computation.
I0907 17:52:18.550426  2034 net.cpp:226] conv3 needs backward computation.
I0907 17:52:18.550431  2034 net.cpp:226] pool2 needs backward computation.
I0907 17:52:18.550441  2034 net.cpp:226] relu2 needs backward computation.
I0907 17:52:18.550446  2034 net.cpp:226] conv2 needs backward computation.
I0907 17:52:18.550451  2034 net.cpp:226] relu1 needs backward computation.
I0907 17:52:18.550456  2034 net.cpp:226] pool1 needs backward computation.
I0907 17:52:18.550460  2034 net.cpp:226] conv1 needs backward computation.
I0907 17:52:18.550467  2034 net.cpp:228] label_data_1_split does not need bac
kward computation.
I0907 17:52:18.550475  2034 net.cpp:228] data does not need backward computat
ion.
I0907 17:52:18.550484  2034 net.cpp:270] This network produces output accurac
y
I0907 17:52:18.550490  2034 net.cpp:270] This network produces output loss
I0907 17:52:18.550505  2034 net.cpp:283] Network initialization done.
I0907 17:52:18.550590  2034 solver.cpp:60] Solver scaffolding done.
I0907 17:52:18.550948  2034 caffe.cpp:213] Starting Optimization
I0907 17:52:18.550966  2034 solver.cpp:280] Solving
I0907 17:52:18.550971  2034 solver.cpp:281] Learning Rate Policy: step
I0907 17:52:18.551618  2034 solver.cpp:338] Iteration 0, Testing net (#0)
I0907 17:52:20.988834  2034 solver.cpp:406]     Test net output #0: accuracy
 = 0.4934
I0907 17:52:20.988890  2034 solver.cpp:406]     Test net output #1: loss = 0.
693383 (* 1 = 0.693383 loss)
I0907 17:52:21.030238  2034 solver.cpp:229] Iteration 0, loss = 0.692136
I0907 17:52:21.030270  2034 solver.cpp:245]     Train net output #0: loss =
 0.692136 (* 1 = 0.692136 loss)
I0907 17:52:21.030292  2034 sgd_solver.cpp:106] Iteration 0, lr = 0.01
I0907 17:52:23.047680  2034 solver.cpp:229] Iteration 20, loss = 0.699369
I0907 17:52:23.047734  2034 solver.cpp:245]     Train net output #0: loss =
 0.699369 (* 1 = 0.699369 loss)
I0907 17:52:23.047744  2034 sgd_solver.cpp:106] Iteration 20, lr = 0.01
I0907 17:52:25.063765  2034 solver.cpp:229] Iteration 40, loss = 0.706948
I0907 17:52:25.063819  2034 solver.cpp:245]     Train net output #0: loss =
 0.706948 (* 1 = 0.706948 loss)
I0907 17:52:25.063829  2034 sgd_solver.cpp:106] Iteration 40, lr = 0.01
I0907 17:52:27.077561  2034 solver.cpp:229] Iteration 60, loss = 0.673435
I0907 17:52:27.077615  2034 solver.cpp:245]     Train net output #0: loss =
 0.673435 (* 1 = 0.673435 loss)
```

```
I0907 17:52:27.077625  2034 sgd_solver.cpp:106] Iteration 60, lr = 0.01
I0907 17:52:29.091864  2034 solver.cpp:229] Iteration 80, loss = 0.659732
I0907 17:52:29.091917  2034 solver.cpp:245]     Train net output #0: loss =
 0.659732 (* 1 = 0.659732 loss)
I0907 17:52:29.091928  2034 sgd_solver.cpp:106] Iteration 80, lr = 0.01
I0907 17:52:31.105305  2034 solver.cpp:229] Iteration 100, loss = 0.670091
I0907 17:52:31.105362  2034 solver.cpp:245]     Train net output #0: loss =
 0.670091 (* 1 = 0.670091 loss)
I0907 17:52:31.105372  2034 sgd_solver.cpp:106] Iteration 100, lr = 0.01
I0907 17:52:33.121857  2034 solver.cpp:229] Iteration 120, loss = 0.673423
I0907 17:52:33.121913  2034 solver.cpp:245]     Train net output #0: loss =
 0.673423 (* 1 = 0.673423 loss)
I0907 17:52:33.121923  2034 sgd_solver.cpp:106] Iteration 120, lr = 0.01
I0907 17:52:35.136251  2034 solver.cpp:229] Iteration 140, loss = 0.642096
I0907 17:52:35.136306  2034 solver.cpp:245]     Train net output #0: loss =
 0.642096 (* 1 = 0.642096 loss)
I0907 17:52:35.136315  2034 sgd_solver.cpp:106] Iteration 140, lr = 0.01
I0907 17:52:37.153023  2034 solver.cpp:229] Iteration 160, loss = 0.615465
I0907 17:52:37.153086  2034 solver.cpp:245]     Train net output #0: loss =
 0.615465 (* 1 = 0.615465 loss)
I0907 17:52:37.153096  2034 sgd_solver.cpp:106] Iteration 160, lr = 0.01
I0907 17:52:39.169787  2034 solver.cpp:229] Iteration 180, loss = 0.579013
I0907 17:52:39.169842  2034 solver.cpp:245]     Train net output #0: loss =
 0.579013 (* 1 = 0.579013 loss)
I0907 17:52:39.169852  2034 sgd_solver.cpp:106] Iteration 180, lr = 0.01
I0907 17:52:41.186693  2034 solver.cpp:229] Iteration 200, loss = 0.614775
I0907 17:52:41.186748  2034 solver.cpp:245]     Train net output #0: loss =
 0.614775 (* 1 = 0.614775 loss)
I0907 17:52:41.186759  2034 sgd_solver.cpp:106] Iteration 200, lr = 0.01
I0907 17:52:43.203661  2034 solver.cpp:229] Iteration 220, loss = 0.635389
I0907 17:52:43.203716  2034 solver.cpp:245]     Train net output #0: loss =
 0.635389 (* 1 = 0.635389 loss)
I0907 17:52:43.203725  2034 sgd_solver.cpp:106] Iteration 220, lr = 0.01
I0907 17:52:45.219781  2034 solver.cpp:229] Iteration 240, loss = 0.610933
I0907 17:52:45.219836  2034 solver.cpp:245]     Train net output #0: loss =
 0.610933 (* 1 = 0.610933 loss)
I0907 17:52:45.219846  2034 sgd_solver.cpp:106] Iteration 240, lr = 0.01
I0907 17:52:46.127920  2034 solver.cpp:338] Iteration 250, Testing net (#0)
I0907 17:52:48.642901  2034 solver.cpp:406]     Test net output #0: accuracy
 = 0.6642
I0907 17:52:48.643168  2034 solver.cpp:406]     Test net output #1: loss = 0.
610271 (* 1 = 0.610271 loss)
I0907 17:52:49.677594  2034 solver.cpp:229] Iteration 260, loss = 0.670394
I0907 17:52:49.677649  2034 solver.cpp:245]     Train net output #0: loss =
 0.670394 (* 1 = 0.670394 loss)
I0907 17:52:49.677659  2034 sgd_solver.cpp:106] Iteration 260, lr = 0.01
I0907 17:52:51.695935  2034 solver.cpp:229] Iteration 280, loss = 0.589702
I0907 17:52:51.695987  2034 solver.cpp:245]     Train net output #0: loss =
 0.589702 (* 1 = 0.589702 loss)
I0907 17:52:51.696000  2034 sgd_solver.cpp:106] Iteration 280, lr = 0.01
I0907 17:52:53.710996  2034 solver.cpp:229] Iteration 300, loss = 0.517287
I0907 17:52:53.711046  2034 solver.cpp:245]     Train net output #0: loss =
 0.517287 (* 1 = 0.517287 loss)
I0907 17:52:53.711056  2034 sgd_solver.cpp:106] Iteration 300, lr = 0.01
I0907 17:52:55.727901  2034 solver.cpp:229] Iteration 320, loss = 0.667897
I0907 17:52:55.727957  2034 solver.cpp:245]     Train net output #0: loss =
 0.667897 (* 1 = 0.667897 loss)
```

```
I0907 17:52:55.727967  2034 sgd_solver.cpp:106] Iteration 320, lr = 0.01
I0907 17:52:57.743724  2034 solver.cpp:229] Iteration 340, loss = 0.577245
I0907 17:52:57.743778  2034 solver.cpp:245]     Train net output #0: loss =
 0.577245 (* 1 = 0.577245 loss)
I0907 17:52:57.743788  2034 sgd_solver.cpp:106] Iteration 340, lr = 0.01
I0907 17:52:59.758558  2034 solver.cpp:229] Iteration 360, loss = 0.577936
I0907 17:52:59.758615  2034 solver.cpp:245]     Train net output #0: loss =
 0.577936 (* 1 = 0.577936 loss)
I0907 17:52:59.758625  2034 sgd_solver.cpp:106] Iteration 360, lr = 0.01
I0907 17:53:01.775216  2034 solver.cpp:229] Iteration 380, loss = 0.599117
I0907 17:53:01.775274  2034 solver.cpp:245]     Train net output #0: loss =
 0.599117 (* 1 = 0.599117 loss)
I0907 17:53:01.775286  2034 sgd_solver.cpp:106] Iteration 380, lr = 0.01
I0907 17:53:03.790626  2034 solver.cpp:229] Iteration 400, loss = 0.654552
I0907 17:53:03.790683  2034 solver.cpp:245]     Train net output #0: loss =
 0.654552 (* 1 = 0.654552 loss)
I0907 17:53:03.790693  2034 sgd_solver.cpp:106] Iteration 400, lr = 0.01
I0907 17:53:05.807268  2034 solver.cpp:229] Iteration 420, loss = 0.56827
I0907 17:53:05.807324  2034 solver.cpp:245]     Train net output #0: loss =
 0.56827 (* 1 = 0.56827 loss)
I0907 17:53:05.807334  2034 sgd_solver.cpp:106] Iteration 420, lr = 0.01
I0907 17:53:07.822702  2034 solver.cpp:229] Iteration 440, loss = 0.522814
I0907 17:53:07.822758  2034 solver.cpp:245]     Train net output #0: loss =
 0.522814 (* 1 = 0.522814 loss)
I0907 17:53:07.822768  2034 sgd_solver.cpp:106] Iteration 440, lr = 0.01
I0907 17:53:09.837556  2034 solver.cpp:229] Iteration 460, loss = 0.593677
I0907 17:53:09.837612  2034 solver.cpp:245]     Train net output #0: loss =
 0.593677 (* 1 = 0.593677 loss)
I0907 17:53:09.837623  2034 sgd_solver.cpp:106] Iteration 460, lr = 0.01
I0907 17:53:11.855217  2034 solver.cpp:229] Iteration 480, loss = 0.547089
I0907 17:53:11.855274  2034 solver.cpp:245]     Train net output #0: loss =
 0.547089 (* 1 = 0.547089 loss)
I0907 17:53:11.855284  2034 sgd_solver.cpp:106] Iteration 480, lr = 0.01
I0907 17:53:13.771093  2034 solver.cpp:338] Iteration 500, Testing net (#0)
I0907 17:53:16.279290  2034 solver.cpp:406]     Test net output #0: accuracy
 = 0.6866
I0907 17:53:16.279343  2034 solver.cpp:406]     Test net output #1: loss = 0.
606226 (* 1 = 0.606226 loss)
I0907 17:53:16.304071  2034 solver.cpp:229] Iteration 500, loss = 0.564794
I0907 17:53:16.304103  2034 solver.cpp:245]     Train net output #0: loss =
 0.564794 (* 1 = 0.564794 loss)
I0907 17:53:16.304114  2034 sgd_solver.cpp:106] Iteration 500, lr = 0.001
I0907 17:53:18.319257  2034 solver.cpp:229] Iteration 520, loss = 0.590697
I0907 17:53:18.319314  2034 solver.cpp:245]     Train net output #0: loss =
 0.590697 (* 1 = 0.590697 loss)
I0907 17:53:18.319324  2034 sgd_solver.cpp:106] Iteration 520, lr = 0.001
I0907 17:53:20.334669  2034 solver.cpp:229] Iteration 540, loss = 0.549887
I0907 17:53:20.334854  2034 solver.cpp:245]     Train net output #0: loss =
 0.549887 (* 1 = 0.549887 loss)
I0907 17:53:20.334866  2034 sgd_solver.cpp:106] Iteration 540, lr = 0.001
I0907 17:53:22.349474  2034 solver.cpp:229] Iteration 560, loss = 0.540129
I0907 17:53:22.349531  2034 solver.cpp:245]     Train net output #0: loss =
 0.540129 (* 1 = 0.540129 loss)
I0907 17:53:22.349541  2034 sgd_solver.cpp:106] Iteration 560, lr = 0.001
I0907 17:53:24.365814  2034 solver.cpp:229] Iteration 580, loss = 0.618205
I0907 17:53:24.365869  2034 solver.cpp:245]     Train net output #0: loss =
 0.618205 (* 1 = 0.618205 loss)
```

```
I0907 17:53:24.365880  2034 sgd_solver.cpp:106] Iteration 580, lr = 0.001
I0907 17:53:26.384941  2034 solver.cpp:229] Iteration 600, loss = 0.5916
I0907 17:53:26.384996  2034 solver.cpp:245]     Train net output #0: loss =
 0.5916 (* 1 = 0.5916 loss)
I0907 17:53:26.385006  2034 sgd_solver.cpp:106] Iteration 600, lr = 0.001
I0907 17:53:28.401162  2034 solver.cpp:229] Iteration 620, loss = 0.473342
I0907 17:53:28.401217  2034 solver.cpp:245]     Train net output #0: loss =
 0.473342 (* 1 = 0.473342 loss)
I0907 17:53:28.401227  2034 sgd_solver.cpp:106] Iteration 620, lr = 0.001
I0907 17:53:30.416332  2034 solver.cpp:229] Iteration 640, loss = 0.604995
I0907 17:53:30.416389  2034 solver.cpp:245]     Train net output #0: loss =
 0.604995 (* 1 = 0.604995 loss)
I0907 17:53:30.416400  2034 sgd_solver.cpp:106] Iteration 640, lr = 0.001
I0907 17:53:32.434772  2034 solver.cpp:229] Iteration 660, loss = 0.511369
I0907 17:53:32.434828  2034 solver.cpp:245]     Train net output #0: loss =
 0.511369 (* 1 = 0.511369 loss)
I0907 17:53:32.434837  2034 sgd_solver.cpp:106] Iteration 660, lr = 0.001
I0907 17:53:34.448817  2034 solver.cpp:229] Iteration 680, loss = 0.5145
I0907 17:53:34.448881  2034 solver.cpp:245]     Train net output #0: loss =
 0.5145 (* 1 = 0.5145 loss)
I0907 17:53:34.448900  2034 sgd_solver.cpp:106] Iteration 680, lr = 0.001
I0907 17:53:36.464830  2034 solver.cpp:229] Iteration 700, loss = 0.505878
I0907 17:53:36.464887  2034 solver.cpp:245]     Train net output #0: loss =
 0.505878 (* 1 = 0.505878 loss)
I0907 17:53:36.464898  2034 sgd_solver.cpp:106] Iteration 700, lr = 0.001
I0907 17:53:38.480952  2034 solver.cpp:229] Iteration 720, loss = 0.545721
I0907 17:53:38.481015  2034 solver.cpp:245]     Train net output #0: loss =
 0.545721 (* 1 = 0.545721 loss)
I0907 17:53:38.481025  2034 sgd_solver.cpp:106] Iteration 720, lr = 0.001
I0907 17:53:40.497026  2034 solver.cpp:229] Iteration 740, loss = 0.49553
I0907 17:53:40.497089  2034 solver.cpp:245]     Train net output #0: loss =
 0.49553 (* 1 = 0.49553 loss)
I0907 17:53:40.497100  2034 sgd_solver.cpp:106] Iteration 740, lr = 0.001
I0907 17:53:41.403877  2034 solver.cpp:456] Snapshotting to binary proto file
 checkpoints/snapshot_iter_750.caffemodel
I0907 17:53:41.511615  2034 sgd_solver.cpp:273] Snapshotting solver state to
 binary proto file checkpoints/snapshot_iter_750.solverstate
I0907 17:53:41.525238  2034 solver.cpp:338] Iteration 750, Testing net (#0)
I0907 17:53:43.960336  2034 solver.cpp:406]     Test net output #0: accuracy
 = 0.7109
I0907 17:53:43.960391  2034 solver.cpp:406]     Test net output #1: loss = 0.
553666 (* 1 = 0.553666 loss)
I0907 17:53:43.960400  2034 solver.cpp:323] Optimization Done.
I0907 17:53:43.960404  2034 caffe.cpp:216] Optimization Done.
```

The classification accuracy remains as 71%.

Results

Model Evaluation and Validation

We will now deploy our final trained network to perform classification of new images. For all of the training above we used the Caffe command line interface tools. For classification we are going to use Caffe's Python interface. We will first import the Python libraries we require and create some variables specifying the locations of important files.

In [25]:
```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.cm as cm
%matplotlib inline

plt.rcParams['figure.figsize'] = (6.0, 4.0)

# Make sure that caffe is on the python path:
#caffe_root = '../'  # this file is expected to be in {caffe_root}/examples
import sys
#sys.path.insert(0, caffe_root + 'python')

import caffe

# Set the right path to your model definition file, pretrained model weights,
# and the image you would like to classify.
MODEL_FILE = '/home/ubuntu/caffe/examples/Dog_Cat_Classification/src/deploy3.p
rototxt'
PRETRAINED = '/home/ubuntu/caffe/examples/Dog_Cat_Classification/checkpoints/p
retrained.caffemodel'
IMAGE_FILE1 = '/home/ubuntu/Data/dog_cat_32/test/cat_236.jpg'
IMAGE_FILE2 = '/home/ubuntu/Data/dog_cat_32/test/dog_4987.jpg'
LABELS_FILE = '/home/ubuntu/Data/dog_cat_32/labels.txt'
labels=open(LABELS_FILE,'r').readlines()
```

Loading a network is easy. The `caffe.Classifier` method takes care of everything. Note the arguments for configuring input preprocessing: mean subtraction switched on by giving a mean array, input channel swapping takes care of mapping RGB into the reference ImageNet model's BGR order, and raw scaling multiplies the feature scale from the input [0,1] to the ImageNet model's [0,255].

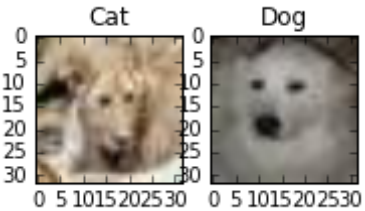
In [26]:
```
# First we must import the mean.binaryproto mean image into a numpy array
blob = caffe.proto.caffe_pb2.BlobProto()
data = open( 'mean.binaryproto' , 'rb' ).read()
blob.ParseFromString(data)
arr = np.array( caffe.io.blobproto_to_array(blob) )
out = arr[0]
```

In [27]:
```
# Load our pretrained model
net = caffe.Classifier(MODEL_FILE, PRETRAINED,
                       mean=out,
                       channel_swap=(2,1,0),
                       raw_scale=255,
                       image_dims=(32, 32))
net2 = caffe.Classifier(MODEL_FILE, PRETRAINED,
                       mean=out,
                       channel_swap=(2,1,0),
                       raw_scale=255,
                       image_dims=(32, 32))
```

Let's take a look at our example images with Caffe's image loading helper. We are going to classify 2 different images, one from each category.

In [28]:
```
# Load two test images
input_image1 = caffe.io.load_image(IMAGE_FILE1)
input_image2 = caffe.io.load_image(IMAGE_FILE2)
# Display the test images
plt.subplot(1,4,1).imshow(input_image1),plt.title('Cat')
plt.subplot(1,4,2).imshow(input_image2),plt.title('Dog')
```

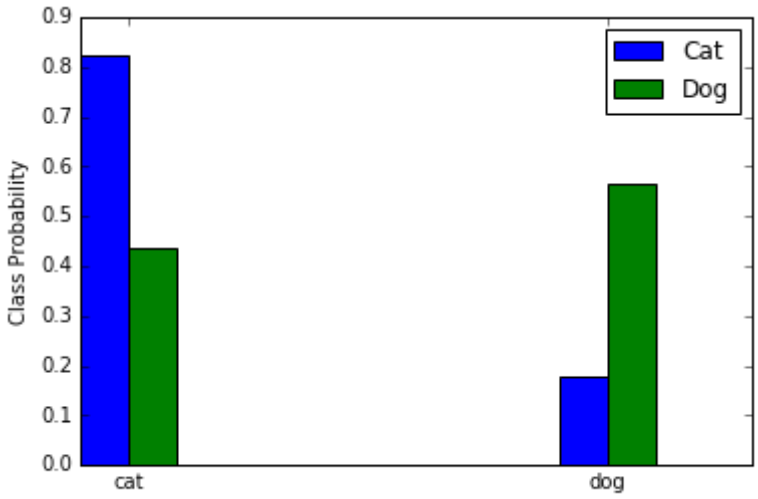
Out[28]:
```
(<matplotlib.image.AxesImage at 0x7f33812ff910>,
 <matplotlib.text.Text at 0x7f3381314d50>)
```



Time to classify. The default is to actually do 10 predictions, cropping the center and corners of the image as well as their mirrored versions, and average over the predictions. This approach typically leads to better classification accuracy as it is more robust to object translation within in the image.

In [29]:
```
prediction1 = net.predict([input_image1])
prediction2 = net2.predict([input_image2])
width=0.1
plt.bar(np.arange(2),prediction1[0],width,color='blue',label='Cat')
plt.bar(np.arange(2)+width,prediction2[0],width,color='green',label='Dog')
plt.xticks(np.arange(2)+width,labels)
plt.ylabel('Class Probability')
plt.legend()
```

Out[29]: <matplotlib.legend.Legend at 0x7f33811d3490>



You can see what class the neural network believes each image is. In the cases above the highest probabilities are given to the correct class for both test images.

Justification

To validate the performance of our model, a good technique would be to utilize the sklearn.cross_validation.KFold module which splits the training set into n-folds and uses n-1 folds for training and 1 fold for testing in a cyclical process that trains and tests every fold. We can compare the n predictions for the folds and see how much they vary.

However, because of the cost associated with training K different models I avoid cross validation. Instead of doing cross validation, we use a random subset of the training data as a hold-out for validation purposes.

A text file containing a list of 20 images of unseen images is provided. By executing the cell below we will classify all of these images with the network we just trained above and calculate the mean accuracy.

In [30]:
```
TEST_FILE=open('/home/ubuntu/Data/dog_cat_32/val/test.txt','r')
TEST_IMAGES=TEST_FILE.readlines()
PredictScore=np.zeros((len(TEST_IMAGES),1))
for i in range(len(TEST_IMAGES)):
    IMAGE_FILE='/home/ubuntu/Data/dog_cat_32/val/' + TEST_IMAGES[i].split()[0]
    CATEGORY=TEST_IMAGES[i].split()[1]
    #print TEST_IMAGES[i]
    input_test = caffe.io.load_image(IMAGE_FILE)
    prediction = net.predict([input_test])
    #print prediction[0]
    if prediction[0].argmax()==int(CATEGORY):
        print 'CORRECT -- predicted class for ', str(IMAGE_FILE[62:]),':', pre
diction[0].argmax(), 'true class:', CATEGORY
    elif prediction[0].argmax()!=int(CATEGORY):
        print 'WRONG -- predicted class ', str(IMAGE_FILE[62:]),':', predictio
n[0].argmax(), 'true class:', CATEGORY
    PredictScore[i]=int(prediction[0].argmax()==int(CATEGORY))
Accuracy=np.sum(PredictScore)/len(PredictScore)
print 'Prediction accuracy with this image set is', np.sum(PredictScore)/len(P
redictScore)
```

```
CORRECT -- predicted class for   : 0 true class: 0
CORRECT -- predicted class for   : 0 true class: 0
CORRECT -- predicted class for   : 0 true class: 0
CORRECT -- predicted class for   : 0 true class: 0
CORRECT -- predicted class for   : 0 true class: 0
CORRECT -- predicted class for   : 0 true class: 0
CORRECT -- predicted class for   : 0 true class: 0
WRONG -- predicted class   : 1 true class: 0
CORRECT -- predicted class for   : 0 true class: 0
CORRECT -- predicted class for   : 0 true class: 0
CORRECT -- predicted class for   : 1 true class: 1
CORRECT -- predicted class for   : 1 true class: 1
CORRECT -- predicted class for   : 1 true class: 1
CORRECT -- predicted class for   : 1 true class: 1
CORRECT -- predicted class for   : 1 true class: 1
WRONG -- predicted class   : 0 true class: 1
WRONG -- predicted class   : 0 true class: 1
CORRECT -- predicted class for   : 1 true class: 1
WRONG -- predicted class   : 0 true class: 1
WRONG -- predicted class   : 0 true class: 1
Prediction accuracy with this image set is 0.75
```

The output above shows our model has correctly classified 75% of our testing set. 75% accuracy is still way below the benchmark (99%) we discussed earlier.

While the solution didn't produce a classifier close to the benchmark given above, it did produce a reasonable classifier which should be very beneficial in classifying Dogs/Cats in images. By evaluating images of the wrong classifications, it appeared to me that those images were either blurry, mix of dogs and cats, an image of big cats (e.g. lions).

It is clear, from output, that this solution model was not sufficient to solve this problem – it seems highly unlikely that even with further refinement this model would perform better than the benchmark. To achieve better accuracy results, more images of different families of cats/dogs required.

Try your own image

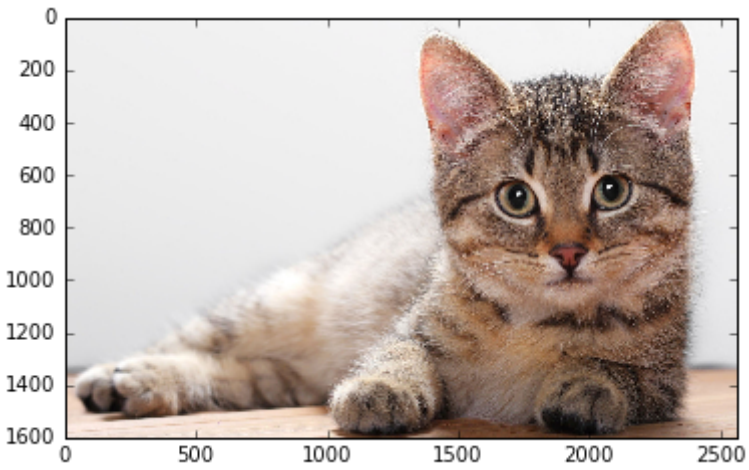
Now we'll grab an image from the web and classify it using the steps above.

- Try setting my_image_url to any JPEG image URL.

In [31]:
```
# download an image
my_image_url = "http://cdn.playbuzz.com/cdn/0079c830-3406-4c05-a5c1-bc43e8f014
79/7dd84d70-768b-492b-88f7-a6c70f2db2e9.jpg"  # paste your URL here
!wget -O image.jpg $my_image_url

# transform it and copy it into the net
image = caffe.io.load_image('image.jpg')
plt.imshow(image)
```

```
--2016-09-07 18:05:05--  http://cdn.playbuzz.com/cdn/0079c830-3406-4c05-a5c1-
bc43e8f01479/7dd84d70-768b-492b-88f7-a6c70f2db2e9.jpg
Resolving cdn.playbuzz.com (cdn.playbuzz.com)... 192.229.163.81
Connecting to cdn.playbuzz.com (cdn.playbuzz.com)|192.229.163.81|:80... conne
cted.
HTTP request sent, awaiting response... 200 OK
Length: 616630 (602K) [image/jpeg]
Saving to: ‘image.jpg’

100%[======================================>] 616,630     --.-K/s   in 0.01s


2016-09-07 18:05:05 (50.0 MB/s) - ‘image.jpg’ saved [616630/616630]
```

Out[31]: <matplotlib.image.AxesImage at 0x7f3381142990>



In [32]:
```
prediction = net.predict([image])
if (prediction[0].argmax() == 1):
    print "This is a DOG!"
else:
    print "This is a CAT!"
```

```
This is a CAT!
```

Conclusion

Free-Form Visualization

This portion of the project follows the filter visualization example provided with Caffe and the DeCAF visualizations originally developed by Yangqing Jia.

In [33]:
```
#View a list of the network layer outputs and their dimensions
[(k, v.data.shape) for k, v in net.blobs.items()]
```

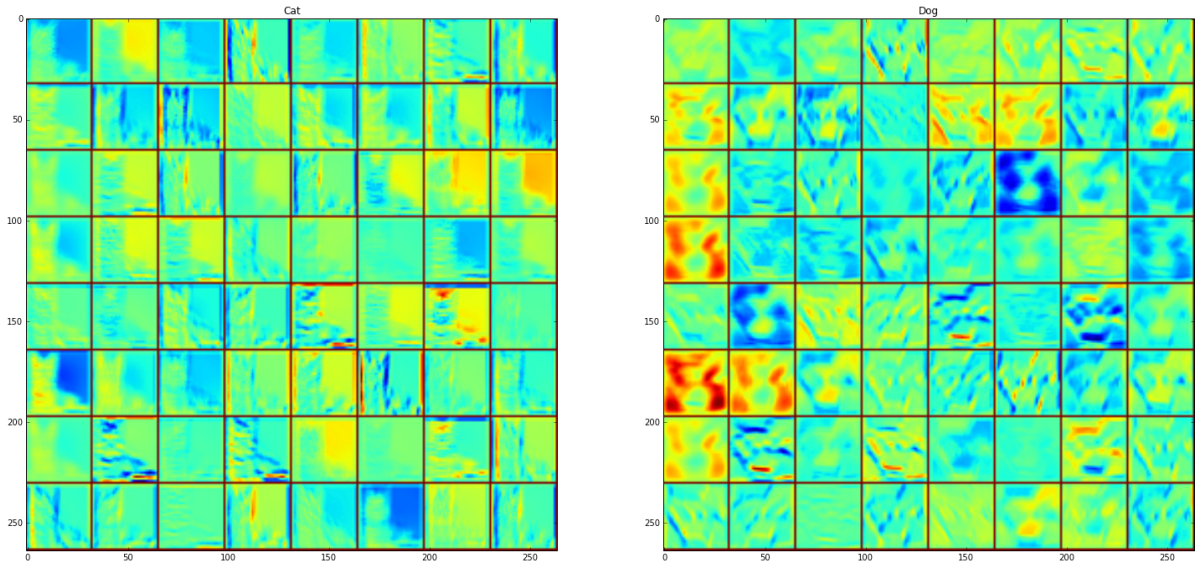
Out[33]:
```
[('data', (1, 3, 32, 32)),
 ('conv1', (1, 64, 32, 32)),
 ('pool1', (1, 64, 16, 16)),
 ('norm1', (1, 64, 16, 16)),
 ('conv2', (1, 128, 16, 16)),
 ('pool2', (1, 128, 8, 8)),
 ('norm2', (1, 128, 8, 8)),
 ('conv3', (1, 128, 8, 8)),
 ('pool3', (1, 128, 4, 4)),
 ('ip2', (1, 500)),
 ('ip3', (1, 500)),
 ('ip1', (1, 2)),
 ('prob', (1, 2))]
```

First you are going to visualize the filters of the first layer.

In [34]:
```
# take an array of shape (n, height, width) or (n, height, width, channels)
# and visualize each (height, width) thing in a grid of size approx. sqrt(n) b
y sqrt(n)
def vis_square(data, padsize=1, padval=0):
    data -= data.min()
    data /= data.max()

    # force the number of filters to be square
    n = int(np.ceil(np.sqrt(data.shape[0])))
    padding = ((0, n ** 2 - data.shape[0]), (0, padsize), (0, padsize)) + ((0,
 0),) * (data.ndim - 3)
    data = np.pad(data, padding, mode='constant', constant_values=(padval, pad
val))

    # tile the filters into an image
    data = data.reshape((n, n) + data.shape[1:]).transpose((0, 2, 1, 3) + tupl
e(range(4, data.ndim + 1)))
    data = data.reshape((n * data.shape[1], n * data.shape[3]) +
data.shape[4:])

    plt.imshow(data)

    # the parameters are a list of [weights, biases]


plt.rcParams['figure.figsize'] = (25.0, 20.0)
filters = net.params['conv1'][0].data
vis_square(filters.transpose(0, 2, 3, 1))
```

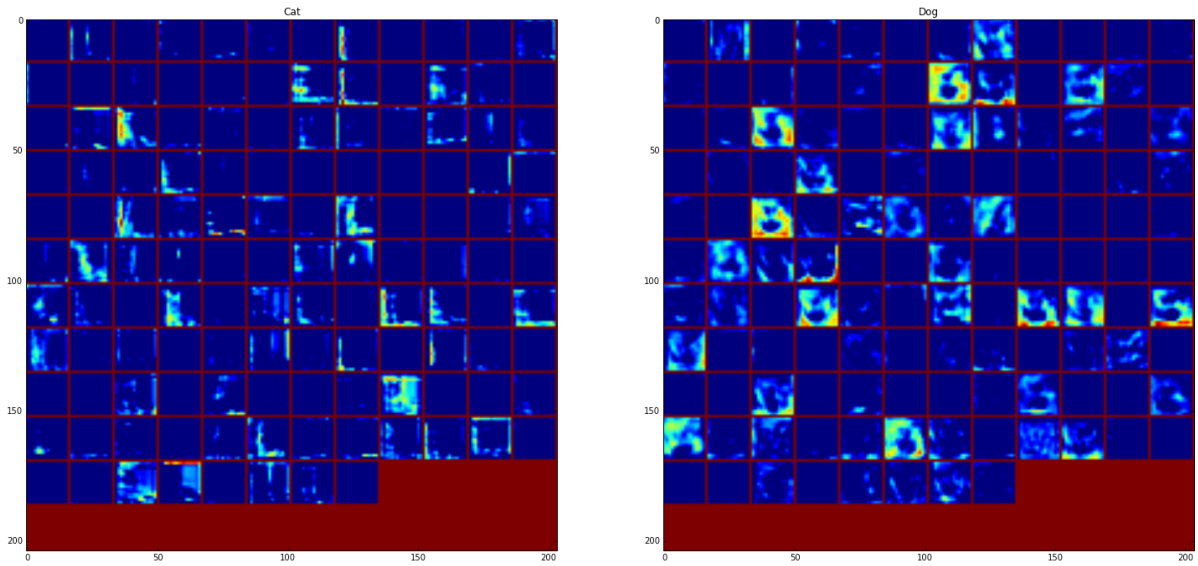
Now you are going to view the feature maps of the two input images after they have been processed by the first convolutional layer. Feel free to modify the feat variables so that you can take a closer look as some of the feature maps more closely. Notice the visual similarities and differences between the features maps of both of these images.

In [35]:
```
feat = net.blobs['conv1'].data[0,:64]
plt.subplot(1,2,1),plt.title('Cat')
vis_square(feat, padval=1)
net.blobs['conv1'].data.shape
feat2 = net2.blobs['conv1'].data[0, :64]
plt.subplot(1,2,2),plt.title('Dog')
vis_square(feat2, padval=1)
```



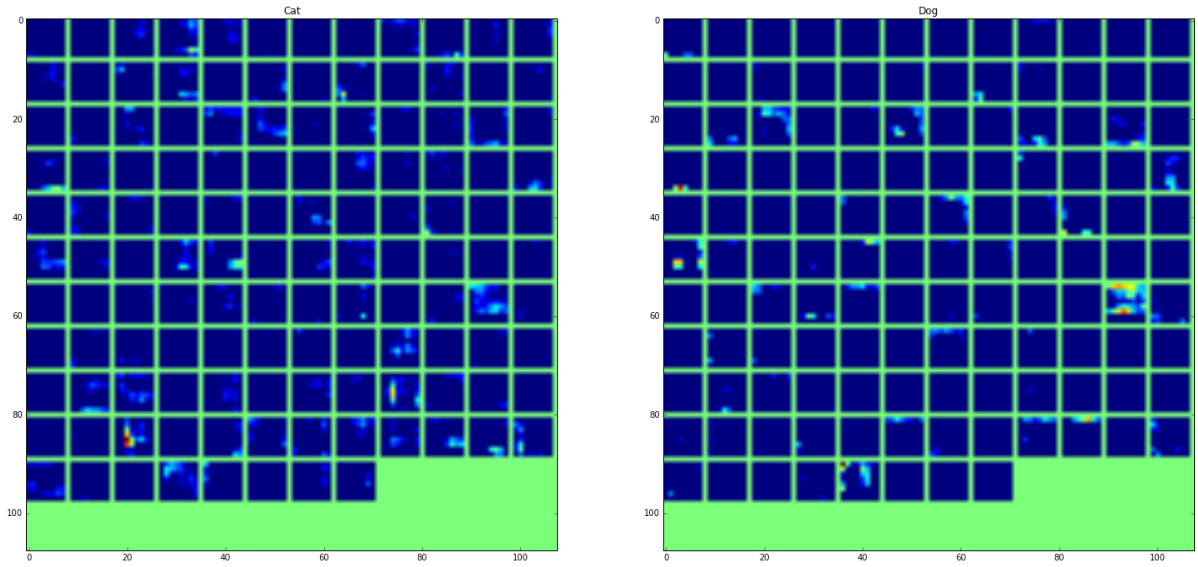
Now view the feature maps from the 2nd convolutional layer.

In [36]:
```
feat = net.blobs['conv2'].data[0]
plt.subplot(1,2,1),plt.title('Cat')
vis_square(feat, padval=1)
feat2 = net2.blobs['conv2'].data[0]
plt.subplot(1,2,2),plt.title('Dog')
vis_square(feat2, padval=1)
```

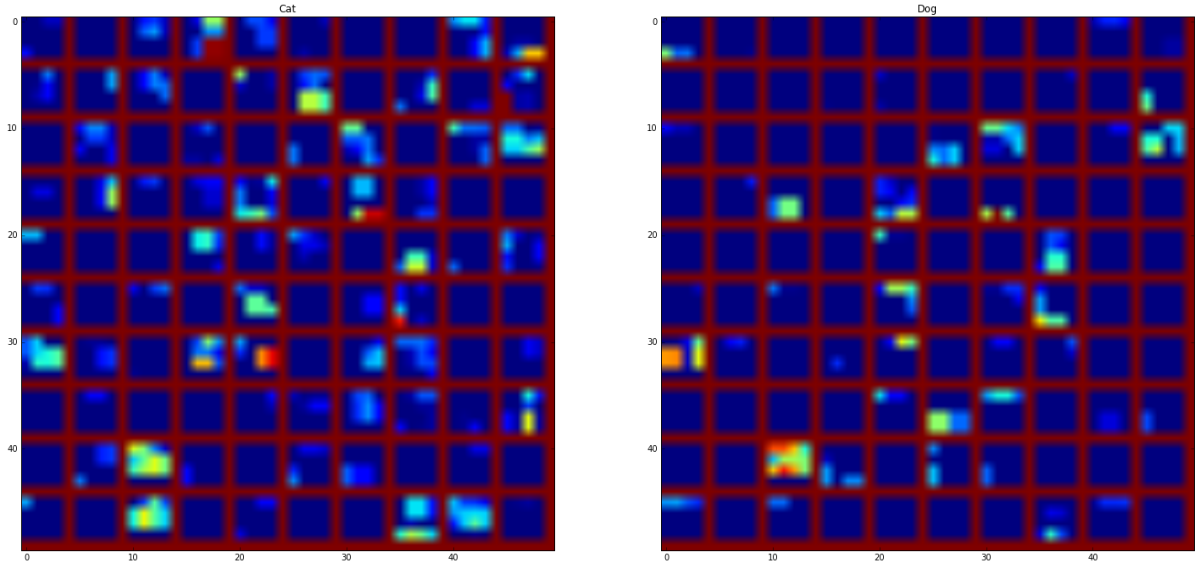


Now view the feature map of the last convolutional layer and then the pooled version.

In [37]:
```
feat = net.blobs['conv3'].data[0]
plt.subplot(1,2,1),plt.title('Cat')
vis_square(feat, padval=0.5)
feat2 = net2.blobs['conv3'].data[0]
plt.subplot(1,2,2),plt.title('Dog')
vis_square(feat2, padval=0.5)
```

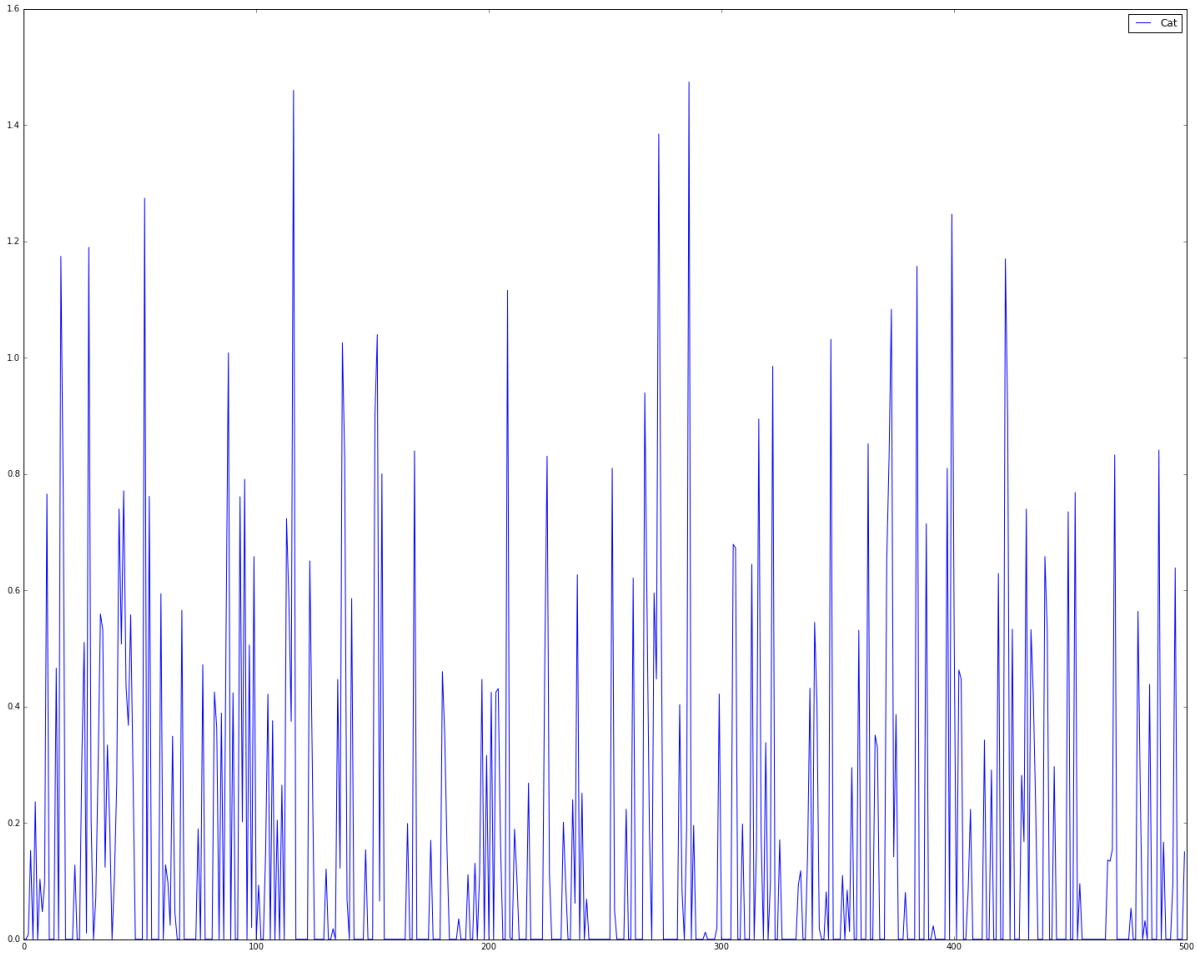


In [38]:
```
feat = net.blobs['pool3'].data[0,:100]
plt.subplot(1,2,1),plt.title('Cat')
vis_square(feat, padval=1)
feat2 = net2.blobs['pool3'].data[0,:100]
plt.subplot(1,2,2),plt.title('Dog')
vis_square(feat2, padval=1)
```

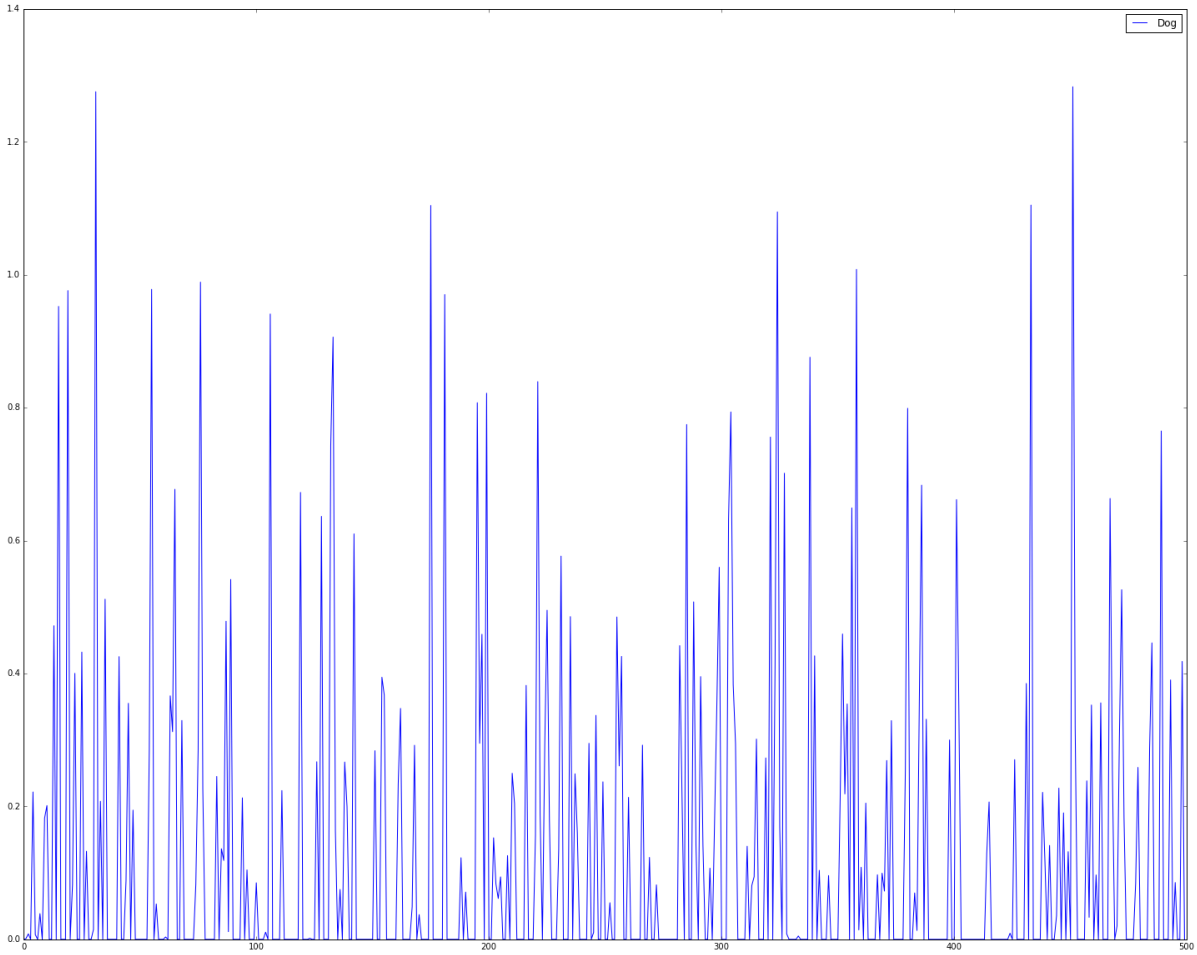


Now view the neuron activations for the fully-connected layer ip2. You will notice that the neurons being activated by the two input images are very different. This is good as it means the network is effectively differentiating the two images at the higher layers in the network.

In [39]:
```
feat = net.blobs['ip2'].data[0]
plt.plot(feat.flat,label='Cat')
plt.legend()
plt.show()
################################################################################
##############################################
# Plot ip2 for the input image of the Dog image. Compare the Differences
feat2 = net2.blobs['ip2'].data[0]
plt.plot(feat2.flat, label='Dog')
plt.legend()
```



Out[39]: <matplotlib.legend.Legend at 0x7f3e888c9ad0>


Dog
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
0
100
200
300
400
500

Reflection

I got interested in object recognition in images when I first watched a PyData video on YouTube. PyData is a gathering of users and developers of data analysis tools in Python, and there is PyData channel on YouTube with lots of videos on different Machine Learning subjects. This particular video that got me interested was from PyData Berlin 2016 and it was about Brand recognition in real-life photos where the presenter talked about the project he worked on to detect brands in photos posted on Instagram. The video link is [here].(https://www.youtube.com/watch?v=0Ho0O1tvcU4 (https://www.youtube.com/watch?v=0Ho0O1tvcU4))

I got the source code for the project but unfortunately the python notebook did not run on my Windows laptop due to missing libraries. Also I did not have access to the dataset used in this project.

I started looking for brand logos dataset and came across FlickrLogos-32_dataset_v2 (http://www.multimedia-computing.de/flickrlogos/). I downloaded the file and started playing with the dataset.

Next, I started studying Caffe. I soon realized that Caffe isn't like scikitlearn: it hasn't been designed so it easily predict using model.fit(X, y) or model.predict(Z) and get a useful result.

I should warn you that Caffe has a steep learning curve, especially if you expect to use it wisely and do some changes and customization. In that level of detail, its documentation is lacking as well (this is of course my personal opinion).

Installation of Caffe is also a challenge since it is required to download the source code and compile the code on a Linux machine. Caffe has several dependencies like CUDA for GPU mode that needs to be installed if you are planning to use GPU. I also learned that there are a lot of vector and matrix operations in deep learning, so it's intuitive that deep learning should run several times faster on a GPU than a CPU just like a game runs faster on a GPU compared to a CPU.

In short, it is a big hassle to install Caffe, and my recommendation is don't even try it to install it on your Windows laptop.

Instead of installing Caffe from scratch, I got a AWS instance with GPU support and started with an AMI that has already installed with all the required software.

I started with Caffe tutorials and used my custom logo dataset. On my first try my jupyter server crashed. I started researching and reading more articles to make this work but Unfortunately it was not a successful experience.

I then decided to switch the project from brand recognition in photos to Dogs/Cats image classifier which I also found it interesting. This was also a competition at Kaggle. I tried to learn as much as I could with this project and then go back to Brand Recognition problem. Hopefully someday you will see this brand recognition project on my Github repo.

The field is moving very fast and new and interesting architectures and techniques are been discussed and used all the time. Because of the success of CNNs in the field of image classification, in many different areas where convolutional neural networks are applied to solve ultra-complex problems, often the input data first is translated to image data.

Understanding ConvNets and learning to use them for the first time can sometimes be an intimidating experience but I found Deep Learning and especially Convolutional Neural Networks very interesting, and there are lots of applications like Image recognition, Video analysis, Natural language processing, Drug discovery, and Playing Go!

Improvement

My Goal in this project was to create Convolutional Neural Network from scratch, however, lots of researchers and engineers have made Caffe models for different tasks with all kinds of architectures and data. These models are learned and applied for problems ranging from simple regression, to large-scale visual classification, to Siamese networks for image similarity, to speech and robotics applications.

To improve the classification accuracy, we might increase network size or use one of the Caffe's predefined models from Caffe Model Zoo (http://caffe.berkeleyvision.org/model_zoo.html), change activation functions, modify training algorithm, and do data augmentation.

Another possibility is using augmented image generators, such as in keras with the ImageDataGenerator class, in order to flip, rotate, and transform your old images to produce new images that can add to your training set.

References

1. Caffe - Deep learning framework by the BVLC (http://caffe.berkeleyvision.org/)
2. Wikipedia - Convolutional Neural Network (https://en.wikipedia.org/wiki/Convolutional_neural_network)
3. An Intuitive Explanation of Convolutional Neural Networks (https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/)
4. Deep Learning in a Nutshell: Core Concepts (https://devblogs.nvidia.com/parallelforall/deep-learning-nutshell-core-concepts/)
5. Start deep learning with Jupyter notebooks in the cloud (http://efavdb.com/deep-learning-with-jupyter-on-aws/)
6. Udacity Deep Learning - Take machine learning to the next level (https://www.udacity.com/course/deep-learning--ud730)
7. Installation Guide - Caffe and Anaconda on AWS EC2 (https://github.com/adilmoujahid/deeplearning-cats-dogs-tutorial/blob/master/aws-ec2-setup.md)
8. CS231n Convolutional Neural Networks for Visual Recognition (http://cs231n.github.io/neural-networks-3/)

In [ ]: