convnet

January 12, 2016

1 Train a ConvNet!

We now have a generic solver and a bunch of modularized layers. It's time to put it all together, and train a ConvNet to recognize the classes in CIFAR-10. In this notebook we will walk you through training a simple two-layer ConvNet and then set you free to build the best net that you can to perform well on CIFAR-10.

Open up the file `cs231n/classifiers/convnet.py`; you will see that the `two_layer_convnet` function computes the loss and gradients for a two-layer ConvNet. Note that this function uses the "sandwich" layers defined in `cs231n/layer_utils.py`.

In [6]:
```
# As usual, a bit of setup

import numpy as np
import matplotlib.pyplot as plt
from cs231n.classifier_trainer import ClassifierTrainer
from cs231n.gradient_check import eval_numerical_gradient
from cs231n.classifiers.convnet import *

%matplotlib inline
plt.rcParams['figure.figsize'] = (10.0, 8.0) # set default size of plots
plt.rcParams['image.interpolation'] = 'nearest'
plt.rcParams['image.cmap'] = 'gray'

# for auto-reloading external modules
# see http://stackoverflow.com/questions/1907993/autoreload-of-modules-in-ipython
%load_ext autoreload
%autoreload 2

def rel_error(x, y):
  """ returns relative error """
  return np.max(np.abs(x - y) / (np.maximum(1e-8, np.abs(x) + np.abs(y))))
```

In [7]:
```
from cs231n.data_utils import load_CIFAR10

def get_CIFAR10_data(num_training=49000, num_validation=1000, num_test=1000):
    """
    Load the CIFAR-10 dataset from disk and perform preprocessing to prepare
    it for the two-layer neural net classifier. These are the same steps as
    we used for the SVM, but condensed to a single function.
    """
    # Load the raw CIFAR-10 data
    cifar10_dir = 'cs231n/datasets/cifar-10-batches-py'
    X_train, y_train, X_test, y_test = load_CIFAR10(cifar10_dir)
```

```
    # Subsample the data
    mask = range(num_training, num_training + num_validation)
    X_val = X_train[mask]
    y_val = y_train[mask]
    mask = range(num_training)
    X_train = X_train[mask]
    y_train = y_train[mask]
    mask = range(num_test)
    X_test = X_test[mask]
    y_test = y_test[mask]

    # Normalize the data: subtract the mean image
    mean_image = np.mean(X_train, axis=0)
    X_train -= mean_image
    X_val -= mean_image
    X_test -= mean_image

    # Transpose so that channels come first
    X_train = X_train.transpose(0, 3, 1, 2).copy()
    X_val = X_val.transpose(0, 3, 1, 2).copy()
    x_test = X_test.transpose(0, 3, 1, 2).copy()

    return X_train, y_train, X_val, y_val, X_test, y_test


# Invoke the above function to get our data.
X_train, y_train, X_val, y_val, X_test, y_test = get_CIFAR10_data()
print 'Train data shape: ', X_train.shape
print 'Train labels shape: ', y_train.shape
print 'Validation data shape: ', X_val.shape
print 'Validation labels shape: ', y_val.shape
print 'Test data shape: ', X_test.shape
print 'Test labels shape: ', y_test.shape
```

```
Train data shape:  (49000, 3, 32, 32)
Train labels shape:  (49000,)
Validation data shape:  (1000, 3, 32, 32)
Validation labels shape:  (1000,)
Test data shape:  (1000, 32, 32, 3)
Test labels shape:  (1000,)
```

2 Sanity check loss

After you build a new network, one of the first things you should do is sanity check the loss. When we use the softmax loss, we expect the loss for random weights (and no regularization) to be about `log(C)` for `C` classes. When we add regularization this should go up.

In [8]:
```
model = init_two_layer_convnet()

X = np.random.randn(100, 3, 32, 32)
y = np.random.randint(10, size=100)

loss, _ = two_layer_convnet(X, model, y, reg=0)
```

```
        # Sanity check: Loss should be about log(10) = 2.3026
        print 'Sanity check loss (no regularization): ', loss

        # Sanity check: Loss should go up when you add regularization
        loss, _ = two_layer_convnet(X, model, y, reg=1)
        print 'Sanity check loss (with regularization): ', loss
```

```
Sanity check loss (no regularization):  2.30261884083
Sanity check loss (with regularization):  2.34457328523
```

3 Gradient check

After the loss looks reasonable, you should always use numeric gradient checking to make sure that your backward pass is correct. When you use numeric gradient checking you should use a small amount of artifical data and a small number of neurons at each layer.

In [9]:
```
num_inputs = 2
input_shape = (3, 16, 16)
reg = 0.0
num_classes = 10
X = np.random.randn(num_inputs, *input_shape)
y = np.random.randint(num_classes, size=num_inputs)

model = init_two_layer_convnet(num_filters=3, filter_size=3, input_shape=input_shape)
loss, grads = two_layer_convnet(X, model, y)
for param_name in sorted(grads):
    f = lambda _: two_layer_convnet(X, model, y)[0]
    param_grad_num = eval_numerical_gradient(f, model[param_name], verbose=False, h=1e-6)
    e = rel_error(param_grad_num, grads[param_name])
    print '%s max relative error: %e' % (param_name, rel_error(param_grad_num, grads[param_name]
```

```
W1 max relative error: 2.695521e-06
W2 max relative error: 1.209239e-05
b1 max relative error: 1.591874e-08
b2 max relative error: 1.399114e-09
```

4 Overfit small data

A nice trick is to train your model with just a few training samples. You should be able to overfit small datasets, which will result in very high training accuracy and comparatively low validation accuracy.

In [10]:
```
# Use a two-layer ConvNet to overfit 50 training examples.

model = init_two_layer_convnet()
trainer = ClassifierTrainer()
best_model, loss_history, train_acc_history, val_acc_history = trainer.train(
          X_train[:50], y_train[:50], X_val, y_val, model, two_layer_convnet,
          reg=0.001, momentum=0.9, learning_rate=0.0001, batch_size=10, num_epochs=10,
          verbose=True)
```

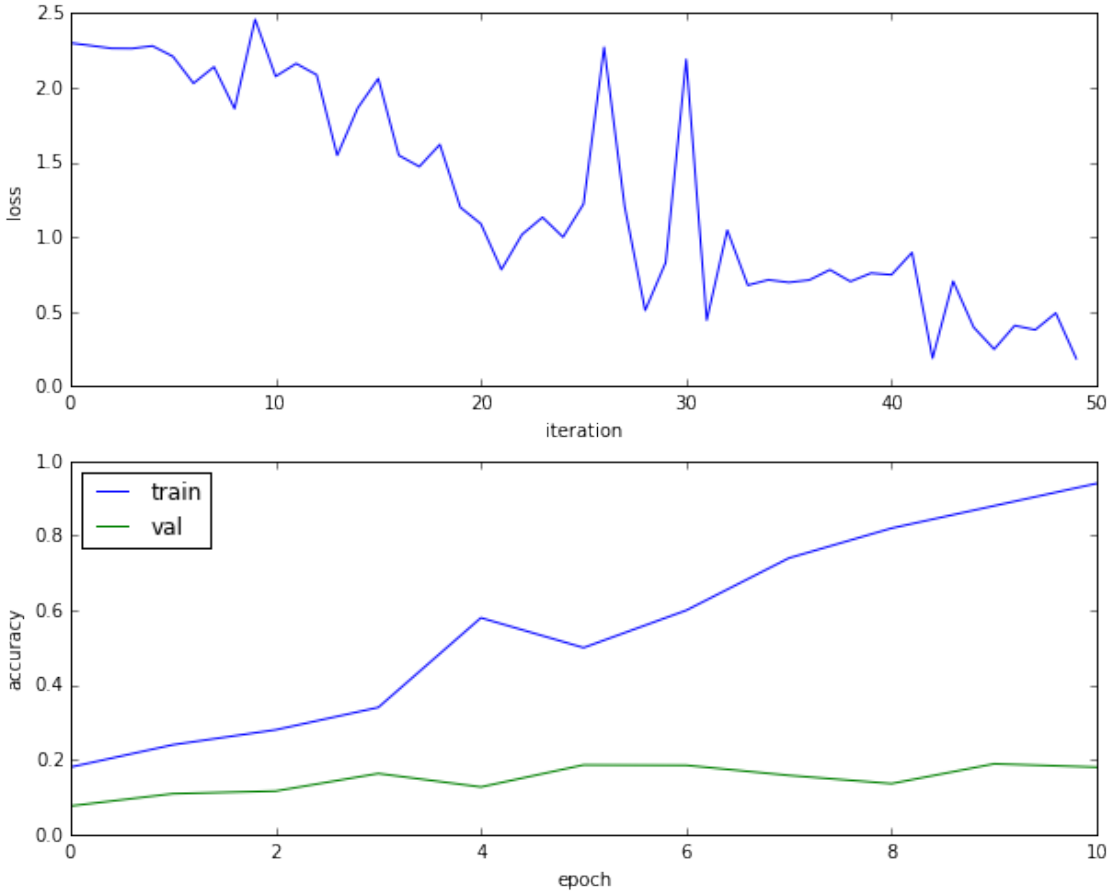
```
starting iteration  0
Finished epoch 0 / 10: cost 2.299339, train: 0.180000, val 0.076000, lr 1.000000e-04
Finished epoch 1 / 10: cost 2.279487, train: 0.240000, val 0.109000, lr 9.500000e-05
Finished epoch 2 / 10: cost 2.456843, train: 0.280000, val 0.116000, lr 9.025000e-05
```

```
starting iteration  10
Finished epoch 3 / 10: cost 1.863759, train: 0.340000, val 0.163000, lr 8.573750e-05
Finished epoch 4 / 10: cost 1.198169, train: 0.580000, val 0.127000, lr 8.145062e-05
starting iteration  20
Finished epoch 5 / 10: cost 0.999505, train: 0.500000, val 0.186000, lr 7.737809e-05
Finished epoch 6 / 10: cost 0.829464, train: 0.600000, val 0.185000, lr 7.350919e-05
starting iteration  30
Finished epoch 7 / 10: cost 0.713109, train: 0.740000, val 0.158000, lr 6.983373e-05
Finished epoch 8 / 10: cost 0.758027, train: 0.820000, val 0.136000, lr 6.634204e-05
starting iteration  40
Finished epoch 9 / 10: cost 0.395614, train: 0.880000, val 0.189000, lr 6.302494e-05
Finished epoch 10 / 10: cost 0.185496, train: 0.940000, val 0.180000, lr 5.987369e-05
finished optimization. best validation accuracy: 0.189000
```

Plotting the loss, training accuracy, and validation accuracy should show clear overfitting:

```
In [11]: plt.subplot(2, 1, 1)
         plt.plot(loss_history)
         plt.xlabel('iteration')
         plt.ylabel('loss')

         plt.subplot(2, 1, 2)
         plt.plot(train_acc_history)
         plt.plot(val_acc_history)
         plt.legend(['train', 'val'], loc='upper left')
         plt.xlabel('epoch')
         plt.ylabel('accuracy')
         plt.show()
```



5 Train the net

Once the above works, training the net is the next thing to try. You can set the `acc_frequency` parameter to change the frequency at which the training and validation set accuracies are tested. If your parameters are set properly, you should see the training and validation accuracy start to improve within a hundred iterations, and you should be able to train a reasonable model with just one epoch.

Using the parameters below you should be able to get around 50% accuracy on the validation set.

```
In [12]: model = init_two_layer_convnet(filter_size=7)
         trainer = ClassifierTrainer()
         best_model, loss_history, train_acc_history, val_acc_history = trainer.train(
                   X_train, y_train, X_val, y_val, model, two_layer_convnet,
                   reg=0.001, momentum=0.9, learning_rate=0.0001, batch_size=50, num_epochs=1,
                   acc_frequency=50, verbose=True)

starting iteration  0
Finished epoch 0 / 1: cost 2.298741, train: 0.106000, val 0.108000, lr 1.000000e-04
starting iteration  10
starting iteration  20
starting iteration  30
starting iteration  40
starting iteration  50
```

```
Finished epoch 0 / 1: cost 1.934964, train: 0.319000, val 0.336000, lr 1.000000e-04
starting iteration  60
starting iteration  70
starting iteration  80
starting iteration  90
starting iteration  100
Finished epoch 0 / 1: cost 1.947233, train: 0.289000, val 0.308000, lr 1.000000e-04
starting iteration  110
starting iteration  120
starting iteration  130
starting iteration  140
starting iteration  150
Finished epoch 0 / 1: cost 1.616109, train: 0.403000, val 0.379000, lr 1.000000e-04
starting iteration  160
starting iteration  170
starting iteration  180
starting iteration  190
starting iteration  200
Finished epoch 0 / 1: cost 1.689934, train: 0.420000, val 0.415000, lr 1.000000e-04
starting iteration  210
starting iteration  220
starting iteration  230
starting iteration  240
starting iteration  250
Finished epoch 0 / 1: cost 1.461824, train: 0.414000, val 0.408000, lr 1.000000e-04
starting iteration  260
starting iteration  270
starting iteration  280
starting iteration  290
starting iteration  300
Finished epoch 0 / 1: cost 1.394433, train: 0.465000, val 0.450000, lr 1.000000e-04
starting iteration  310
starting iteration  320
starting iteration  330
starting iteration  340
starting iteration  350
Finished epoch 0 / 1: cost 1.366479, train: 0.426000, val 0.410000, lr 1.000000e-04
starting iteration  360
starting iteration  370
starting iteration  380
starting iteration  390
starting iteration  400
Finished epoch 0 / 1: cost 1.698907, train: 0.408000, val 0.432000, lr 1.000000e-04
starting iteration  410
starting iteration  420
starting iteration  430
starting iteration  440
starting iteration  450
Finished epoch 0 / 1: cost 1.808498, train: 0.448000, val 0.413000, lr 1.000000e-04
starting iteration  460
starting iteration  470
starting iteration  480
starting iteration  490
starting iteration  500
```

```
Finished epoch 0 / 1: cost 2.835956, train: 0.472000, val 0.440000, lr 1.000000e-04
starting iteration  510
starting iteration  520
starting iteration  530
starting iteration  540
starting iteration  550
Finished epoch 0 / 1: cost 1.945160, train: 0.443000, val 0.458000, lr 1.000000e-04
starting iteration  560
starting iteration  570
starting iteration  580
starting iteration  590
starting iteration  600
Finished epoch 0 / 1: cost 1.716514, train: 0.499000, val 0.475000, lr 1.000000e-04
starting iteration  610
starting iteration  620
starting iteration  630
starting iteration  640
starting iteration  650
Finished epoch 0 / 1: cost 1.739311, train: 0.515000, val 0.471000, lr 1.000000e-04
starting iteration  660
starting iteration  670
starting iteration  680
starting iteration  690
starting iteration  700
Finished epoch 0 / 1: cost 1.226189, train: 0.482000, val 0.479000, lr 1.000000e-04
starting iteration  710
starting iteration  720
starting iteration  730
starting iteration  740
starting iteration  750
Finished epoch 0 / 1: cost 1.617910, train: 0.535000, val 0.494000, lr 1.000000e-04
starting iteration  760
starting iteration  770
starting iteration  780
starting iteration  790
starting iteration  800
Finished epoch 0 / 1: cost 1.438191, train: 0.475000, val 0.457000, lr 1.000000e-04
starting iteration  810
starting iteration  820
starting iteration  830
starting iteration  840
starting iteration  850
Finished epoch 0 / 1: cost 2.431353, train: 0.438000, val 0.442000, lr 1.000000e-04
starting iteration  860
starting iteration  870
starting iteration  880
starting iteration  890
starting iteration  900
Finished epoch 0 / 1: cost 2.309801, train: 0.528000, val 0.519000, lr 1.000000e-04
starting iteration  910
starting iteration  920
starting iteration  930
starting iteration  940
starting iteration  950
```

```
Finished epoch 0 / 1: cost 1.674035, train: 0.473000, val 0.443000, lr 1.000000e-04
starting iteration  960
starting iteration  970
Finished epoch 1 / 1: cost 2.045021, train: 0.491000, val 0.463000, lr 9.500000e-05
finished optimization. best validation accuracy: 0.519000
```

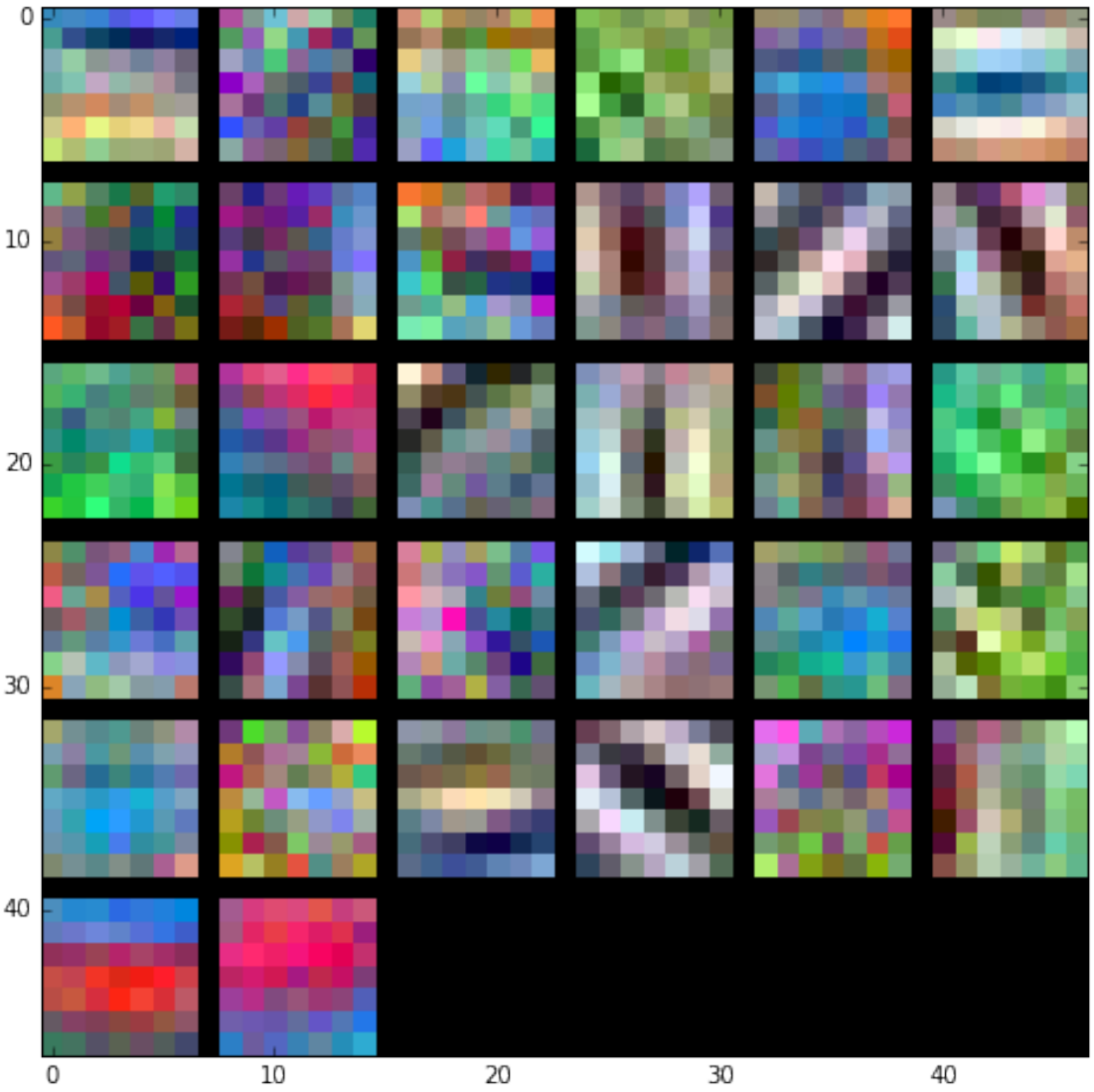
6 Visualize weights

We can visualize the convolutional weights from the first layer. If everything worked properly, these will usually be edges and blobs of various colors and orientations.

In [8]:
```
from cs231n.vis_utils import visualize_grid

grid = visualize_grid(best_model['W1'].transpose(0, 2, 3, 1))
plt.imshow(grid.astype('uint8'))
```

Out[8]: <matplotlib.image.AxesImage at 0x1069d21d0>



7 Experiment!

Experiment and try to get the best performance that you can on CIFAR-10 using a ConvNet. Here are some ideas to get you started:

7.0.1 Things you should try:

- Filter size: Above we used 7x7; this makes pretty pictures but smaller filters may be more efficient
- Number of filters: Above we used 32 filters. Do more or fewer do better?
- Network depth: The network above has two layers of trainable parameters. Can you do better with a deeper network? You can implement alternative architectures in the file `cs231n/classifiers/convnet.py`. Some good architectures to try include:
  - [conv-relu-pool]xN - conv - relu - [affine]xM - [softmax or SVM]
  - [conv-relu-pool]XN - [affine]XM - [softmax or SVM]
  - [conv-relu-conv-relu-pool]xN - [affine]xM - [softmax or SVM]

7.0.2 Tips for training

For each network architecture that you try, you should tune the learning rate and regularization strength. When doing this there are a couple important things to keep in mind:

- If the parameters are working well, you should see improvement within a few hundred iterations
- Remember the course-to-fine approach for hyperparameter tuning: start by testing a large range of hyperparameters for just a few training iterations to find the combinations of parameters that are working at all.
- Once you have found some sets of parameters that seem to work, search more finely around these parameters. You may need to train for more epochs.

7.0.3 Going above and beyond

If you are feeling adventurous there are many other features you can implement to try and improve your performance. You are **not required** to implement any of these; however they would be good things to try for extra credit.

- Alternative update steps: For the assignment we implemented SGD+momentum and RMSprop; you could try alternatives like AdaGrad or AdaDelta.
- Other forms of regularization such as L1 or Dropout
- Alternative activation functions such as leaky ReLU or maxout
- Model ensembles
- Data augmentation

7.0.4 What we expect

At the very least, you should be able to train a ConvNet that gets at least 65% accuracy on the validation set. This is just a lower bound - if you are careful it should be possible to get accuracies much higher than that! Extra credit points will be awarded for particularly high-scoring models or unique approaches.

You should use the space below to experiment and train your network. The final cell in this notebook should contain the training, validation, and test set accuracies for your final trained network. In this notebook you should also write an explanation of what you did, any additional features that you implemented, and any visualizations or graphs that you make in the process of training and evaluating your network.

Have fun and happy training!

In [35]:
```
# TODO: Train a ConvNet to do really well on CIFAR-10!
from cs231n.classifiers.convnet import *
from cs231n.layers import *

#model = init_two_layer_convnet(filter_size=3, num_filters=64)
#trainer = ClassifierTrainer()
#best_model, loss_history, train_acc_history, val_acc_history = trainer.train(
#          X_train, y_train, X_val, y_val, model, two_layer_convnet,
#          reg=0.001, momentum=0.9, learning_rate=0.0001, batch_size=50, num_epochs=1,
#          acc_frequency=50, verbose=True)

model = init_my_convnet()
trainer = ClassifierTrainer()
best_model, loss_history, train_acc_history, val_acc_history = trainer.train(
         X_train, y_train, X_val, y_val, model, my_convnet, momentum=0.9, learning_rate=0.0005
         batch_size=50, num_epochs=5, acc_frequency=50, verbose=True)
```

```
starting iteration  0
Finished epoch 0 / 5: cost 2.300679, train: 0.081000, val 0.102000, lr 5.000000e-04
starting iteration  10
starting iteration  20
starting iteration  30
starting iteration  40
starting iteration  50
Finished epoch 0 / 5: cost 2.200456, train: 0.233000, val 0.251000, lr 5.000000e-04
starting iteration  60
starting iteration  70
starting iteration  80
starting iteration  90
starting iteration  100
Finished epoch 0 / 5: cost 2.011854, train: 0.297000, val 0.319000, lr 5.000000e-04
starting iteration  110
starting iteration  120
starting iteration  130
starting iteration  140
starting iteration  150
Finished epoch 0 / 5: cost 1.868806, train: 0.367000, val 0.375000, lr 5.000000e-04
starting iteration  160
starting iteration  170
starting iteration  180
starting iteration  190
starting iteration  200
Finished epoch 0 / 5: cost 1.982107, train: 0.400000, val 0.404000, lr 5.000000e-04
starting iteration  210
starting iteration  220
starting iteration  230
starting iteration  240
starting iteration  250
Finished epoch 0 / 5: cost 1.626580, train: 0.431000, val 0.444000, lr 5.000000e-04
starting iteration  260
starting iteration  270
starting iteration  280
starting iteration  290
starting iteration  300
```

```
Finished epoch 0 / 5: cost 1.516094, train: 0.421000, val 0.447000, lr 5.000000e-04
starting iteration  310
starting iteration  320
starting iteration  330
starting iteration  340
starting iteration  350
Finished epoch 0 / 5: cost 1.481755, train: 0.457000, val 0.470000, lr 5.000000e-04
starting iteration  360
starting iteration  370
starting iteration  380
starting iteration  390
starting iteration  400
Finished epoch 0 / 5: cost 1.550152, train: 0.476000, val 0.477000, lr 5.000000e-04
starting iteration  410
starting iteration  420
starting iteration  430
starting iteration  440
starting iteration  450
Finished epoch 0 / 5: cost 1.520160, train: 0.500000, val 0.485000, lr 5.000000e-04
starting iteration  460
starting iteration  470
starting iteration  480
starting iteration  490
starting iteration  500
Finished epoch 0 / 5: cost 1.626071, train: 0.526000, val 0.518000, lr 5.000000e-04
starting iteration  510
starting iteration  520
starting iteration  530
starting iteration  540
starting iteration  550
Finished epoch 0 / 5: cost 1.209234, train: 0.522000, val 0.543000, lr 5.000000e-04
starting iteration  560
starting iteration  570
starting iteration  580
starting iteration  590
starting iteration  600
Finished epoch 0 / 5: cost 1.277498, train: 0.519000, val 0.524000, lr 5.000000e-04
starting iteration  610
starting iteration  620
starting iteration  630
starting iteration  640
starting iteration  650
Finished epoch 0 / 5: cost 1.236845, train: 0.537000, val 0.542000, lr 5.000000e-04
starting iteration  660
starting iteration  670
starting iteration  680
starting iteration  690
starting iteration  700
Finished epoch 0 / 5: cost 1.311758, train: 0.561000, val 0.578000, lr 5.000000e-04
starting iteration  710
starting iteration  720
starting iteration  730
starting iteration  740
starting iteration  750
```

```
Finished epoch 0 / 5: cost 1.264987, train: 0.606000, val 0.575000, lr 5.000000e-04
starting iteration  760
starting iteration  770
starting iteration  780
starting iteration  790
starting iteration  800
Finished epoch 0 / 5: cost 1.207748, train: 0.574000, val 0.563000, lr 5.000000e-04
starting iteration  810
starting iteration  820
starting iteration  830
starting iteration  840
starting iteration  850
Finished epoch 0 / 5: cost 1.367678, train: 0.609000, val 0.585000, lr 5.000000e-04
starting iteration  860
starting iteration  870
starting iteration  880
starting iteration  890
starting iteration  900
Finished epoch 0 / 5: cost 1.154976, train: 0.632000, val 0.601000, lr 5.000000e-04
starting iteration  910
starting iteration  920
starting iteration  930
starting iteration  940
starting iteration  950
Finished epoch 0 / 5: cost 1.243287, train: 0.610000, val 0.589000, lr 5.000000e-04
starting iteration  960
starting iteration  970
Finished epoch 1 / 5: cost 1.069391, train: 0.606000, val 0.583000, lr 4.750000e-04
starting iteration  980
starting iteration  990
starting iteration  1000
Finished epoch 1 / 5: cost 1.215283, train: 0.601000, val 0.606000, lr 4.750000e-04
starting iteration  1010
starting iteration  1020
starting iteration  1030
starting iteration  1040
starting iteration  1050
Finished epoch 1 / 5: cost 1.183057, train: 0.592000, val 0.575000, lr 4.750000e-04
starting iteration  1060
starting iteration  1070
starting iteration  1080
starting iteration  1090
starting iteration  1100
Finished epoch 1 / 5: cost 1.375366, train: 0.589000, val 0.584000, lr 4.750000e-04
starting iteration  1110
starting iteration  1120
starting iteration  1130
starting iteration  1140
starting iteration  1150
Finished epoch 1 / 5: cost 1.222407, train: 0.614000, val 0.624000, lr 4.750000e-04
starting iteration  1160
starting iteration  1170
starting iteration  1180
starting iteration  1190
```

```
starting iteration  1200
Finished epoch 1 / 5: cost 0.874685, train: 0.638000, val 0.621000, lr 4.750000e-04
starting iteration  1210
starting iteration  1220
starting iteration  1230
starting iteration  1240
starting iteration  1250
Finished epoch 1 / 5: cost 0.868269, train: 0.628000, val 0.607000, lr 4.750000e-04
starting iteration  1260
starting iteration  1270
starting iteration  1280
starting iteration  1290
starting iteration  1300
Finished epoch 1 / 5: cost 0.899256, train: 0.648000, val 0.618000, lr 4.750000e-04
starting iteration  1310
starting iteration  1320
starting iteration  1330
starting iteration  1340
starting iteration  1350
Finished epoch 1 / 5: cost 0.999799, train: 0.632000, val 0.615000, lr 4.750000e-04
starting iteration  1360
starting iteration  1370
starting iteration  1380
starting iteration  1390
starting iteration  1400
Finished epoch 1 / 5: cost 0.885498, train: 0.672000, val 0.636000, lr 4.750000e-04
starting iteration  1410
starting iteration  1420
starting iteration  1430
starting iteration  1440
starting iteration  1450
Finished epoch 1 / 5: cost 1.156575, train: 0.622000, val 0.630000, lr 4.750000e-04
starting iteration  1460
starting iteration  1470
starting iteration  1480
starting iteration  1490
starting iteration  1500
Finished epoch 1 / 5: cost 0.927210, train: 0.651000, val 0.649000, lr 4.750000e-04
starting iteration  1510
starting iteration  1520
starting iteration  1530
starting iteration  1540
starting iteration  1550
Finished epoch 1 / 5: cost 0.686877, train: 0.689000, val 0.651000, lr 4.750000e-04
starting iteration  1560
starting iteration  1570
starting iteration  1580
starting iteration  1590
starting iteration  1600
Finished epoch 1 / 5: cost 0.830908, train: 0.668000, val 0.665000, lr 4.750000e-04
starting iteration  1610
starting iteration  1620
starting iteration  1630
starting iteration  1640
```

```
starting iteration  1650
Finished epoch 1 / 5: cost 0.804865, train: 0.691000, val 0.647000, lr 4.750000e-04
starting iteration  1660
starting iteration  1670
starting iteration  1680
starting iteration  1690
starting iteration  1700
Finished epoch 1 / 5: cost 0.969134, train: 0.622000, val 0.654000, lr 4.750000e-04
starting iteration  1710
starting iteration  1720
starting iteration  1730
starting iteration  1740
starting iteration  1750
Finished epoch 1 / 5: cost 0.833219, train: 0.675000, val 0.655000, lr 4.750000e-04
starting iteration  1760
starting iteration  1770
starting iteration  1780
starting iteration  1790
starting iteration  1800
Finished epoch 1 / 5: cost 0.699026, train: 0.636000, val 0.649000, lr 4.750000e-04
starting iteration  1810
starting iteration  1820
starting iteration  1830
starting iteration  1840
starting iteration  1850
Finished epoch 1 / 5: cost 0.931040, train: 0.675000, val 0.632000, lr 4.750000e-04
starting iteration  1860
starting iteration  1870
starting iteration  1880
starting iteration  1890
starting iteration  1900
Finished epoch 1 / 5: cost 0.890078, train: 0.709000, val 0.660000, lr 4.750000e-04
starting iteration  1910
starting iteration  1920
starting iteration  1930
starting iteration  1940
starting iteration  1950
Finished epoch 1 / 5: cost 0.938553, train: 0.693000, val 0.661000, lr 4.750000e-04
Finished epoch 2 / 5: cost 0.901983, train: 0.693000, val 0.658000, lr 4.512500e-04
starting iteration  1960
starting iteration  1970
starting iteration  1980
starting iteration  1990
starting iteration  2000
Finished epoch 2 / 5: cost 0.901837, train: 0.703000, val 0.680000, lr 4.512500e-04
starting iteration  2010
starting iteration  2020
starting iteration  2030
starting iteration  2040
starting iteration  2050
Finished epoch 2 / 5: cost 1.011459, train: 0.725000, val 0.667000, lr 4.512500e-04
starting iteration  2060
starting iteration  2070
starting iteration  2080
```

```
starting iteration  2090
starting iteration  2100
Finished epoch 2 / 5: cost 1.004735, train: 0.731000, val 0.683000, lr 4.512500e-04
starting iteration  2110
starting iteration  2120
starting iteration  2130
starting iteration  2140
starting iteration  2150
Finished epoch 2 / 5: cost 1.187211, train: 0.690000, val 0.657000, lr 4.512500e-04
starting iteration  2160
starting iteration  2170
starting iteration  2180
starting iteration  2190
starting iteration  2200
Finished epoch 2 / 5: cost 1.319832, train: 0.715000, val 0.668000, lr 4.512500e-04
starting iteration  2210
starting iteration  2220
starting iteration  2230
starting iteration  2240
starting iteration  2250
Finished epoch 2 / 5: cost 0.560827, train: 0.713000, val 0.668000, lr 4.512500e-04
starting iteration  2260
starting iteration  2270
starting iteration  2280
starting iteration  2290
starting iteration  2300
Finished epoch 2 / 5: cost 0.899339, train: 0.718000, val 0.690000, lr 4.512500e-04
starting iteration  2310
starting iteration  2320
starting iteration  2330
starting iteration  2340
starting iteration  2350
Finished epoch 2 / 5: cost 1.006655, train: 0.713000, val 0.688000, lr 4.512500e-04
starting iteration  2360
starting iteration  2370
starting iteration  2380
starting iteration  2390
starting iteration  2400
Finished epoch 2 / 5: cost 1.024293, train: 0.729000, val 0.683000, lr 4.512500e-04
starting iteration  2410
starting iteration  2420
starting iteration  2430
starting iteration  2440
starting iteration  2450
Finished epoch 2 / 5: cost 0.751033, train: 0.712000, val 0.689000, lr 4.512500e-04
starting iteration  2460
starting iteration  2470
starting iteration  2480
starting iteration  2490
starting iteration  2500
Finished epoch 2 / 5: cost 0.714302, train: 0.708000, val 0.665000, lr 4.512500e-04
starting iteration  2510
starting iteration  2520
starting iteration  2530
```

```
starting iteration  2540
starting iteration  2550
Finished epoch 2 / 5: cost 0.957750, train: 0.718000, val 0.693000, lr 4.512500e-04
starting iteration  2560
starting iteration  2570
starting iteration  2580
starting iteration  2590
starting iteration  2600
Finished epoch 2 / 5: cost 0.602884, train: 0.762000, val 0.700000, lr 4.512500e-04
starting iteration  2610
starting iteration  2620
starting iteration  2630
starting iteration  2640
starting iteration  2650
Finished epoch 2 / 5: cost 0.655529, train: 0.736000, val 0.697000, lr 4.512500e-04
starting iteration  2660
starting iteration  2670
starting iteration  2680
starting iteration  2690
starting iteration  2700
Finished epoch 2 / 5: cost 0.707096, train: 0.758000, val 0.710000, lr 4.512500e-04
starting iteration  2710
starting iteration  2720
starting iteration  2730
starting iteration  2740
starting iteration  2750
Finished epoch 2 / 5: cost 0.550205, train: 0.734000, val 0.685000, lr 4.512500e-04
starting iteration  2760
starting iteration  2770
starting iteration  2780
starting iteration  2790
starting iteration  2800
Finished epoch 2 / 5: cost 0.629230, train: 0.755000, val 0.708000, lr 4.512500e-04
starting iteration  2810
starting iteration  2820
starting iteration  2830
starting iteration  2840
starting iteration  2850
Finished epoch 2 / 5: cost 0.965902, train: 0.724000, val 0.677000, lr 4.512500e-04
starting iteration  2860
starting iteration  2870
starting iteration  2880
starting iteration  2890
starting iteration  2900
Finished epoch 2 / 5: cost 0.852758, train: 0.747000, val 0.708000, lr 4.512500e-04
starting iteration  2910
starting iteration  2920
starting iteration  2930
Finished epoch 3 / 5: cost 0.554288, train: 0.735000, val 0.702000, lr 4.286875e-04
starting iteration  2940
starting iteration  2950
Finished epoch 3 / 5: cost 0.606847, train: 0.760000, val 0.699000, lr 4.286875e-04
starting iteration  2960
starting iteration  2970
```

```
starting iteration  2980
starting iteration  2990
starting iteration  3000
Finished epoch 3 / 5: cost 0.717710, train: 0.759000, val 0.708000, lr 4.286875e-04
starting iteration  3010
starting iteration  3020
starting iteration  3030
starting iteration  3040
starting iteration  3050
Finished epoch 3 / 5: cost 0.787513, train: 0.763000, val 0.695000, lr 4.286875e-04
starting iteration  3060
starting iteration  3070
starting iteration  3080
starting iteration  3090
starting iteration  3100
Finished epoch 3 / 5: cost 0.704798, train: 0.710000, val 0.667000, lr 4.286875e-04
starting iteration  3110
starting iteration  3120
starting iteration  3130
starting iteration  3140
starting iteration  3150
Finished epoch 3 / 5: cost 0.599914, train: 0.773000, val 0.708000, lr 4.286875e-04
starting iteration  3160
starting iteration  3170
starting iteration  3180
starting iteration  3190
starting iteration  3200
Finished epoch 3 / 5: cost 0.781879, train: 0.764000, val 0.698000, lr 4.286875e-04
starting iteration  3210
starting iteration  3220
starting iteration  3230
starting iteration  3240
starting iteration  3250
Finished epoch 3 / 5: cost 0.639903, train: 0.774000, val 0.713000, lr 4.286875e-04
starting iteration  3260
starting iteration  3270
starting iteration  3280
starting iteration  3290
starting iteration  3300
Finished epoch 3 / 5: cost 0.692868, train: 0.747000, val 0.705000, lr 4.286875e-04
starting iteration  3310
starting iteration  3320
starting iteration  3330
starting iteration  3340
starting iteration  3350
Finished epoch 3 / 5: cost 0.809297, train: 0.762000, val 0.724000, lr 4.286875e-04
starting iteration  3360
starting iteration  3370
starting iteration  3380
starting iteration  3390
starting iteration  3400
Finished epoch 3 / 5: cost 0.844133, train: 0.771000, val 0.720000, lr 4.286875e-04
starting iteration  3410
starting iteration  3420
```

```
starting iteration  3430
starting iteration  3440
starting iteration  3450
Finished epoch 3 / 5: cost 0.456535, train: 0.753000, val 0.716000, lr 4.286875e-04
starting iteration  3460
starting iteration  3470
starting iteration  3480
starting iteration  3490
starting iteration  3500
Finished epoch 3 / 5: cost 0.523346, train: 0.776000, val 0.691000, lr 4.286875e-04
starting iteration  3510
starting iteration  3520
starting iteration  3530
starting iteration  3540
starting iteration  3550
Finished epoch 3 / 5: cost 0.522825, train: 0.764000, val 0.706000, lr 4.286875e-04
starting iteration  3560
starting iteration  3570
starting iteration  3580
starting iteration  3590
starting iteration  3600
Finished epoch 3 / 5: cost 0.764020, train: 0.777000, val 0.706000, lr 4.286875e-04
starting iteration  3610
starting iteration  3620
starting iteration  3630
starting iteration  3640
starting iteration  3650
Finished epoch 3 / 5: cost 0.751349, train: 0.799000, val 0.708000, lr 4.286875e-04
starting iteration  3660
starting iteration  3670
starting iteration  3680
starting iteration  3690
starting iteration  3700
Finished epoch 3 / 5: cost 0.715394, train: 0.775000, val 0.704000, lr 4.286875e-04
starting iteration  3710
starting iteration  3720
starting iteration  3730
starting iteration  3740
starting iteration  3750
Finished epoch 3 / 5: cost 0.582312, train: 0.766000, val 0.705000, lr 4.286875e-04
starting iteration  3760
starting iteration  3770
starting iteration  3780
starting iteration  3790
starting iteration  3800
Finished epoch 3 / 5: cost 0.736625, train: 0.811000, val 0.724000, lr 4.286875e-04
starting iteration  3810
starting iteration  3820
starting iteration  3830
starting iteration  3840
starting iteration  3850
Finished epoch 3 / 5: cost 0.599804, train: 0.784000, val 0.709000, lr 4.286875e-04
starting iteration  3860
starting iteration  3870
```

```
starting iteration  3880
starting iteration  3890
starting iteration  3900
Finished epoch 3 / 5: cost 0.505844, train: 0.771000, val 0.702000, lr 4.286875e-04
starting iteration  3910
Finished epoch 4 / 5: cost 0.667030, train: 0.797000, val 0.716000, lr 4.072531e-04
starting iteration  3920
starting iteration  3930
starting iteration  3940
starting iteration  3950
Finished epoch 4 / 5: cost 0.430920, train: 0.809000, val 0.711000, lr 4.072531e-04
starting iteration  3960
starting iteration  3970
starting iteration  3980
starting iteration  3990
starting iteration  4000
Finished epoch 4 / 5: cost 0.543429, train: 0.789000, val 0.718000, lr 4.072531e-04
starting iteration  4010
starting iteration  4020
starting iteration  4030
starting iteration  4040
starting iteration  4050
Finished epoch 4 / 5: cost 0.567855, train: 0.801000, val 0.730000, lr 4.072531e-04
starting iteration  4060
starting iteration  4070
starting iteration  4080
starting iteration  4090
starting iteration  4100
Finished epoch 4 / 5: cost 0.563883, train: 0.804000, val 0.722000, lr 4.072531e-04
starting iteration  4110
starting iteration  4120
starting iteration  4130
starting iteration  4140
starting iteration  4150
Finished epoch 4 / 5: cost 0.681282, train: 0.765000, val 0.702000, lr 4.072531e-04
starting iteration  4160
starting iteration  4170
starting iteration  4180
starting iteration  4190
starting iteration  4200
Finished epoch 4 / 5: cost 0.908928, train: 0.792000, val 0.733000, lr 4.072531e-04
starting iteration  4210
starting iteration  4220
starting iteration  4230
starting iteration  4240
starting iteration  4250
Finished epoch 4 / 5: cost 0.891740, train: 0.791000, val 0.732000, lr 4.072531e-04
starting iteration  4260
starting iteration  4270
starting iteration  4280
starting iteration  4290
starting iteration  4300
Finished epoch 4 / 5: cost 0.569775, train: 0.834000, val 0.735000, lr 4.072531e-04
starting iteration  4310
```

```
starting iteration  4320
starting iteration  4330
starting iteration  4340
starting iteration  4350
Finished epoch 4 / 5: cost 0.409690, train: 0.802000, val 0.748000, lr 4.072531e-04
starting iteration  4360
starting iteration  4370
starting iteration  4380
starting iteration  4390
starting iteration  4400
Finished epoch 4 / 5: cost 0.634047, train: 0.818000, val 0.715000, lr 4.072531e-04
starting iteration  4410
starting iteration  4420
starting iteration  4430
starting iteration  4440
starting iteration  4450
Finished epoch 4 / 5: cost 0.577226, train: 0.838000, val 0.736000, lr 4.072531e-04
starting iteration  4460
starting iteration  4470
starting iteration  4480
starting iteration  4490
starting iteration  4500
Finished epoch 4 / 5: cost 0.647941, train: 0.792000, val 0.735000, lr 4.072531e-04
starting iteration  4510
starting iteration  4520
starting iteration  4530
starting iteration  4540
starting iteration  4550
Finished epoch 4 / 5: cost 0.578878, train: 0.822000, val 0.736000, lr 4.072531e-04
starting iteration  4560
starting iteration  4570
starting iteration  4580
starting iteration  4590
starting iteration  4600
Finished epoch 4 / 5: cost 0.435067, train: 0.812000, val 0.731000, lr 4.072531e-04
starting iteration  4610
starting iteration  4620
starting iteration  4630
starting iteration  4640
starting iteration  4650
Finished epoch 4 / 5: cost 0.423294, train: 0.811000, val 0.714000, lr 4.072531e-04
starting iteration  4660
starting iteration  4670
starting iteration  4680
starting iteration  4690
starting iteration  4700
Finished epoch 4 / 5: cost 0.714353, train: 0.823000, val 0.734000, lr 4.072531e-04
starting iteration  4710
starting iteration  4720
starting iteration  4730
starting iteration  4740
starting iteration  4750
Finished epoch 4 / 5: cost 0.534575, train: 0.794000, val 0.734000, lr 4.072531e-04
starting iteration  4760
```

```
starting iteration  4770
starting iteration  4780
starting iteration  4790
starting iteration  4800
Finished epoch 4 / 5: cost 0.375923, train: 0.823000, val 0.745000, lr 4.072531e-04
starting iteration  4810
starting iteration  4820
starting iteration  4830
starting iteration  4840
starting iteration  4850
Finished epoch 4 / 5: cost 0.363778, train: 0.806000, val 0.745000, lr 4.072531e-04
starting iteration  4860
starting iteration  4870
starting iteration  4880
starting iteration  4890
Finished epoch 5 / 5: cost 0.409641, train: 0.823000, val 0.730000, lr 3.868905e-04
finished optimization. best validation accuracy: 0.748000
```

In [41]:
```
#NOTE: There is a typo in the get_CIFAR10_data code above
#right before the return statement.
#Rather than regenerate this book, I am running this line below
x_test = X_test.transpose(0, 3, 1, 2).copy()
```

In [42]:
```
print X_val.shape, x_test.shape
```

```
(1000, 3, 32, 32) (1000, 3, 32, 32)
```

In [43]:
```
#Make sure that I get 0.748 for the validation set with best_model
scores = my_convnet(X_val, best_model)
np.mean(np.argmax(scores, axis=1) == y_val)
```

Out[43]: 0.748

In [44]:
```
#Now check the test set score
scores = my_convnet(x_test, best_model)
np.mean(np.argmax(scores, axis=1) == y_test)
```

Out[44]: 0.72799999999999998

In [ ]: