New Image features exercise

I have used dense SIFT features

In [1]:

```
import random
import numpy as np
from asgn1.data_utils import load_CIFAR10
import matplotlib.pyplot as plt
%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 extenrnal modules
# see http://stackoverflow.com/questions/1907993/autoreload-of-modules-in-ipython
%load_ext autoreload
%autoreload 2
```

Load data

Similar to previous exercises, we will load CIFAR-10 data from disk.

In [2]:

```
from asgn1.features import color_histogram_hsv, hog_feature

def get_CIFAR10_data(num_training=49000, num_validation=1000, num_test=1000):
  # Load the raw CIFAR-10 data
  cifar10_dir = '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]

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

X_train, y_train, X_val, y_val, X_test, y_test = get_CIFAR10_data()
```

Extract Features

making use of dSIFT features code for the paper :Y. Jia and T. Darrell. ``Heavy-tailed Distances for Gradient Based Image Descriptors''. NIPS 2011."

In [3]:

```
from dsift import *
from scipy import misc
extractor = DsiftExtractor(8,16,1)
fe=[]
sha=[X_train.shape[0],X_val.shape[0],X_test.shape[0]]
for j in range(3):
    f=[]
    for i in range(0,sha[j]):
        if j==0:
            img = X_train[i].reshape(32,32,3)
        else:
            if j==1:
                img = X_val[i].reshape(32,32,3)
            else:
                img = X_test[i].reshape(32,32,3)

        img = np.mean(np.double(img),axis=2)
        feaArr = extractor.process_image(img)
        feaArr = feaArr.flatten()
        f.append(feaArr)
        if i%1000 == 0 :
            print str(i)+" images extracted"
    fe.append(f)
print np.array(f).shape
X_train_feats = fe[0]
X_val_feats = fe[1]
X_test_feats = fe[2]

# Preprocessing: Subtract the mean feature
mean_feat = np.mean(X_train_feats, axis=0, keepdims=True)
X_train_feats -= mean_feat
X_val_feats -= mean_feat
X_test_feats -= mean_feat

# Preprocessing: Divide by standard deviation. This ensures that each feature
# has roughly the same scale.
std_feat = np.std(X_train_feats, axis=0, keepdims=True)
X_train_feats /= std_feat
X_val_feats /= std_feat
X_test_feats /= std_feat

# Preprocessing: Add a bias dimension
X_train_feats = np.hstack([X_train_feats, np.ones((X_train_feats.shape[0], 1))])
X_val_feats = np.hstack([X_val_feats, np.ones((X_val_feats.shape[0], 1))])
X_test_feats = np.hstack([X_test_feats, np.ones((X_test_feats.shape[0], 1))])
```

```
0 images extracted
1000 images extracted
2000 images extracted
3000 images extracted
4000 images extracted
5000 images extracted
6000 images extracted
7000 images extracted
8000 images extracted
9000 images extracted
10000 images extracted
11000 images extracted
12000 images extracted
```

```
13000 images extracted
14000 images extracted
15000 images extracted
16000 images extracted
17000 images extracted
18000 images extracted
19000 images extracted
20000 images extracted
21000 images extracted
22000 images extracted
23000 images extracted
24000 images extracted
25000 images extracted
26000 images extracted
27000 images extracted
28000 images extracted
29000 images extracted
30000 images extracted
31000 images extracted
32000 images extracted
33000 images extracted
34000 images extracted
35000 images extracted
36000 images extracted
37000 images extracted
38000 images extracted
39000 images extracted
40000 images extracted
41000 images extracted
42000 images extracted
43000 images extracted
44000 images extracted
45000 images extracted
46000 images extracted
47000 images extracted
48000 images extracted
0 images extracted
0 images extracted
(1000, 1152)
```

Train SVM on features

Using the multiclass SVM code developed earlier in the assignment, train SVMs on top of the features extracted above; this should achieve better results than training SVMs directly on top of raw pixels.

In [4]:

```
# Use the validation set to tune the learning rate and regularization strength

from asgn1.classifiers.linear_classifier import LinearSVM

X_train_feats=np.array(X_train_feats)
X_val_feats=np.array(X_val_feats)

learning_rates = [1e-8, 1e-7,1e-6]
regularization_strengths = [1e4,1e5,5e5,1e6]

results = {}
best_val = -1
best_svm = None
for i in learning_rates:
    for j in regularization_strengths:
        svm=LinearSVM()
        loss_hist = svm.train(X_train_feats, y_train, learning_rate=i, reg=j,
                      num_iters=2000, verbose=True)
        y_train_pred = svm.predict(X_train_feats)
        y_val_pred = svm.predict(X_val_feats)
        results[i,j]=[np.mean(y_train == y_train_pred),np.mean(y_val == y_val_pred)
        print np.mean(y_val == y_val_pred)
        if (np.mean(y_val == y_val_pred))>best_val:
            best_val=np.mean(y_val == y_val_pred)
            best_svm=svm


# Print out results.
for lr, reg in sorted(results):
    train_accuracy, val_accuracy = results[(lr, reg)]
    print 'lr %e reg %e train accuracy: %f val accuracy: %f' % (
                lr, reg, train_accuracy, val_accuracy)

print 'best validation accuracy achieved during cross-validation: %f' % best_val
```

```
iteration 0 / 2000: loss 66.260143
iteration 100 / 2000: loss 65.079175
iteration 200 / 2000: loss 64.028854
iteration 300 / 2000: loss 62.879741
iteration 400 / 2000: loss 61.823113
iteration 500 / 2000: loss 60.808731
iteration 600 / 2000: loss 59.755392
iteration 700 / 2000: loss 58.751626
iteration 800 / 2000: loss 57.811802
iteration 900 / 2000: loss 56.798424
iteration 1000 / 2000: loss 55.864402
iteration 1100 / 2000: loss 54.928443
iteration 1200 / 2000: loss 54.021427
iteration 1300 / 2000: loss 53.142665
iteration 1400 / 2000: loss 52.243390
iteration 1500 / 2000: loss 51.437374
iteration 1600 / 2000: loss 50.558399
iteration 1700 / 2000: loss 49.767724
iteration 1800 / 2000: loss 48.907608
iteration 1900 / 2000: loss 48.126269
```

In [5]:

```
# Evaluate your trained SVM on the test set
y_test_pred = best_svm.predict(np.array(X_test_feats))
test_accuracy = np.mean(y_test == y_test_pred)
print test_accuracy
```

```
0.417
```

Neural Network on dSIFT image features

In [6]:

```
print X_train_feats.shape
```

```
(49000, 1153)
```

In [8]:

```
from asgn1.classifiers.neural_net import TwoLayerNet

input_dim = X_train_feats.shape[1]
hidden_dim = 70
num_classes = 10
learning_rates=[2e-1,3e-1,5e-1]
regularization_strengths = [0.0005,0.002,0.003,0.005,0.01]
best_val=-1
results={}
best_net = None
input_size = 32 * 32 * 3
for i in learning_rates:
    for j in regularization_strengths:
        net = TwoLayerNet(input_dim, hidden_dim, num_classes)
        stats = net.train(X_train_feats, y_train, X_val_feats, y_val,
                                num_iters=1600,batch_size=200,
                                learning_rate=i, learning_rate_decay=0.95,
                                    reg=j, verbose=True)
        y_train_pred = net.predict(X_train_feats)
        y_val_pred = net.predict(X_val_feats)
        results[i,j]=[np.mean(y_train == y_train_pred),np.mean(y_val == y_val_pred)
        print np.mean(y_val == y_val_pred)
        if (np.mean(y_val == y_val_pred))>best_val:
            best_val=np.mean(y_val == y_val_pred)
            best_net=net
print best_val
for lr, reg in sorted(results):
    train_accuracy, val_accuracy = results[(lr, reg)]
    print 'lr %e reg %e train accuracy: %f val accuracy: %f' % (
                lr, reg, train_accuracy, val_accuracy)

print 'best validation accuracy achieved during cross-validation: %f' % best_val
```

```
iteration 0 / 1600: loss 2.302585
iteration 100 / 1600: loss 1.544467
iteration 200 / 1600: loss 1.318993
iteration 300 / 1600: loss 1.216357
iteration 400 / 1600: loss 1.226037
iteration 500 / 1600: loss 1.076611
iteration 600 / 1600: loss 1.040595
iteration 700 / 1600: loss 1.050253
iteration 800 / 1600: loss 0.965504
iteration 900 / 1600: loss 1.161479
iteration 1000 / 1600: loss 1.101231
iteration 1100 / 1600: loss 1.005848
iteration 1200 / 1600: loss 0.863756
iteration 1300 / 1600: loss 0.856009
iteration 1400 / 1600: loss 0.907827
iteration 1500 / 1600: loss 0.906499
0.609
iteration 0 / 1600: loss 2.302586
iteration 100 / 1600: loss 1.688927
iteration 200 / 1600: loss 1.456226
```

In [12]:

```
# Run your neural net classifier on the test set. You should be able to
# get more than 55% accuracy.
confmatrix = np.zeros((num_classes,num_classes))
for i in range(0,y_test_pred.shape[0]):
    confmatrix[y_test[i],y_test_pred[i]]+=1
test_acc = (best_net.predict(np.array(X_test_feats)) == y_test).mean()
print test_acc
```

```
0.602
```

In [ ]:

```
#Acheived 60.2% accuracy when using dSift features
```

In [13]:

```
# saving the best svm model to pickel file
import pickle
pickle.dump({"best_net":best_net,"cmatrix":confmatrix}, open( "best_net_SIFT.p", "w
```