Image features exercise

Complete and hand in this completed worksheet (including its outputs and any supporting code outside of the worksheet) with your assignment submission. For more details see the [assignments page (https://compsci697l.github.io/assignments.html)](https://compsci697l.github.io/assignments.html) on the course website.

We have seen that we can achieve reasonable performance on an image classification task by training a linear classifier on the pixels of the input image. In this exercise we will show that we can improve our classification performance by training linear classifiers not on raw pixels but on features that are computed from the raw pixels.

All of your work for this exercise will be done in this notebook.

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

For each image we will compute a Histogram of Oriented Gradients (HOG) as well as a color histogram using the hue channel in HSV color space. We form our final feature vector for each image by concatenating the HOG and color histogram feature vectors.

Roughly speaking, HOG should capture the texture of the image while ignoring color information, and the color histogram represents the color of the input image while ignoring texture. As a result, we expect that using both together ought to work better than using either alone. Verifying this assumption would be a good thing to try for the bonus section.

The `hog_feature` and `color_histogram_hsv` functions both operate on a single image and return a feature vector for that image. The extract_features function takes a set of images and a list of feature functions and evaluates each feature function on each image, storing the results in a matrix where each row is the concatenation of all feature vectors for a single image.

In [36]:

```
from asgn1.features import *

num_color_bins = 20 # Number of bins in the color histogram
feature_fns = [hog_feature, lambda img: color_histogram_hsv(img, nbin=num_color_bin
X_train_feats = extract_features(X_train, feature_fns, verbose=True)
X_val_feats = extract_features(X_val, feature_fns)
X_test_feats = extract_features(X_test, feature_fns)

# 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))])
```

```
Done extracting features for 1000 / 49000 images
Done extracting features for 2000 / 49000 images
Done extracting features for 3000 / 49000 images
Done extracting features for 4000 / 49000 images
Done extracting features for 5000 / 49000 images
Done extracting features for 6000 / 49000 images
Done extracting features for 7000 / 49000 images
Done extracting features for 8000 / 49000 images
Done extracting features for 9000 / 49000 images
Done extracting features for 10000 / 49000 images
Done extracting features for 11000 / 49000 images
Done extracting features for 12000 / 49000 images
Done extracting features for 13000 / 49000 images
Done extracting features for 14000 / 49000 images
Done extracting features for 15000 / 49000 images
Done extracting features for 16000 / 49000 images
Done extracting features for 17000 / 49000 images
Done extracting features for 18000 / 49000 images
Done extracting features for 19000 / 49000 images
Done extracting features for 20000 / 49000 images
Done extracting features for 21000 / 49000 images
Done extracting features for 22000 / 49000 images
Done extracting features for 23000 / 49000 images
Done extracting features for 24000 / 49000 images
Done extracting features for 25000 / 49000 images
Done extracting features for 26000 / 49000 images
Done extracting features for 27000 / 49000 images
Done extracting features for 28000 / 49000 images
Done extracting features for 29000 / 49000 images
Done extracting features for 30000 / 49000 images
Done extracting features for 31000 / 49000 images
Done extracting features for 32000 / 49000 images
Done extracting features for 33000 / 49000 images
```

```
Done extracting features for 34000 / 49000 images
Done extracting features for 35000 / 49000 images
Done extracting features for 36000 / 49000 images
Done extracting features for 37000 / 49000 images
Done extracting features for 38000 / 49000 images
Done extracting features for 39000 / 49000 images
Done extracting features for 40000 / 49000 images
Done extracting features for 41000 / 49000 images
Done extracting features for 42000 / 49000 images
Done extracting features for 43000 / 49000 images
Done extracting features for 44000 / 49000 images
Done extracting features for 45000 / 49000 images
Done extracting features for 46000 / 49000 images
Done extracting features for 47000 / 49000 images
Done extracting features for 48000 / 49000 images
```

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 [69]:

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

from asgn1.classifiers.linear_classifier import LinearSVM

learning_rates = [1e-9, 1e-8, 1e-7]
regularization_strengths = [1e6, 1e7]

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=5000, 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 results

################################################################################
# TODO:                                                                        #
# Use the validation set to set the learning rate and regularization strength. #
# This should be identical to the validation that you did for the SVM; save    #
# the best trained classifer in best_svm. You might also want to play          #
# with different numbers of bins in the color histogram. If you are careful    #
# you should be able to get accuracy of near 0.44 on the validation set.       #
################################################################################

################################################################################
#                              END OF YOUR CODE                                #
################################################################################

# 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 3900 / 5000: loss 9.345487
iteration 4000 / 5000: loss 9.283063
iteration 4100 / 5000: loss 9.231611
iteration 4200 / 5000: loss 9.189456
iteration 4300 / 5000: loss 9.155080
iteration 4400 / 5000: loss 9.127086
iteration 4500 / 5000: loss 9.104020
iteration 4600 / 5000: loss 9.085292
iteration 4700 / 5000: loss 9.069717
iteration 4800 / 5000: loss 9.056985
iteration 4900 / 5000: loss 9.046679
0.118
iteration 0 / 5000: loss 8072.252029
iteration 100 / 5000: loss 1089.309002
iteration 200 / 5000: loss 153.740998
```

```
iteration 300 / 5000: loss 28.392002
iteration 400 / 5000: loss 11.598460
iteration 500 / 5000: loss 9.348007
iteration 600 / 5000: loss 9.046611
iteration 700 / 5000: loss 9.006253
```

In [46]:

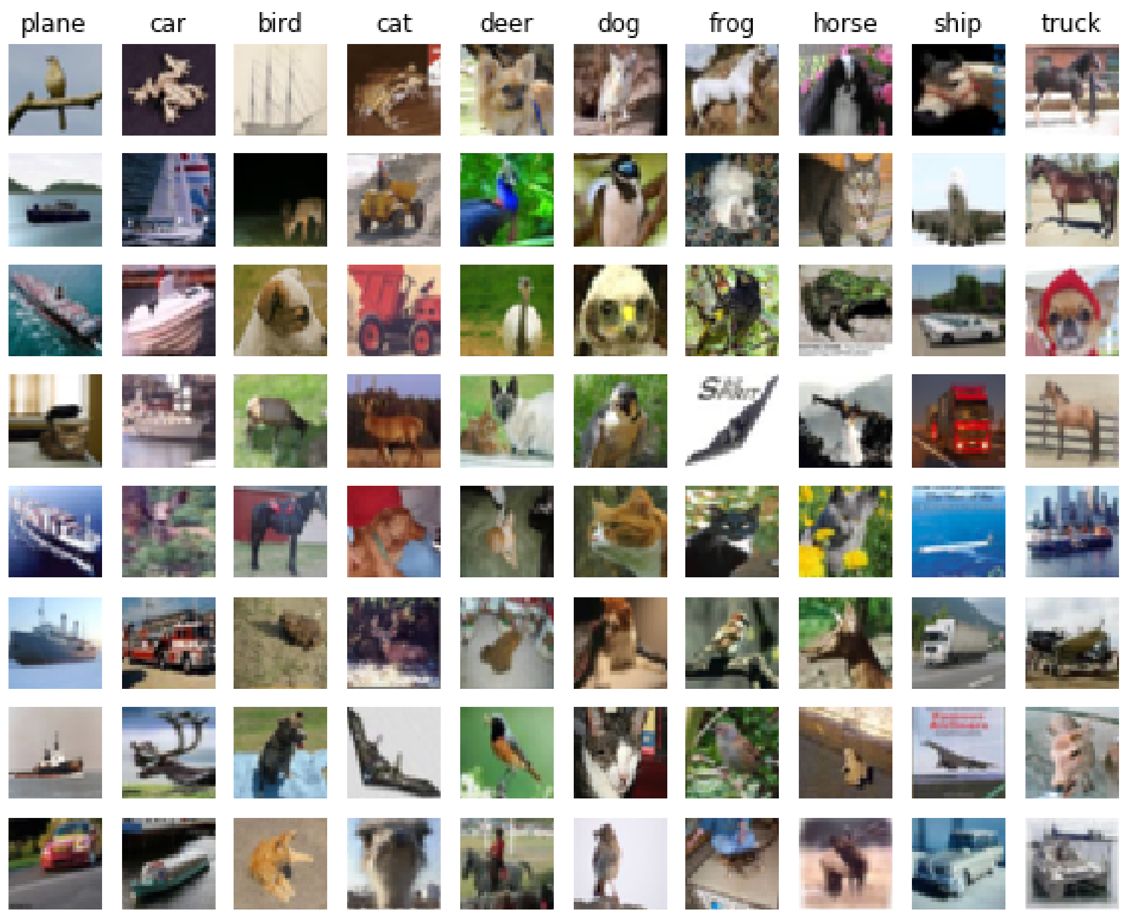
```
# Evaluate your trained SVM on the test set
y_test_pred = best_svm.predict(X_test_feats)
test_accuracy = np.mean(y_test == y_test_pred)
print test_accuracy
```

```
0.423
```

In [47]:

```
# An important way to gain intuition about how an algorithm works is to
# visualize the mistakes that it makes. In this visualization, we show examples
# of images that are misclassified by our current system. The first column
# shows images that our system labeled as "plane" but whose true label is
# something other than "plane".

examples_per_class = 8
classes = ['plane', 'car', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', '
for cls, cls_name in enumerate(classes):
    idxs = np.where((y_test != cls) & (y_test_pred == cls))[0]
    idxs = np.random.choice(idxs, examples_per_class, replace=False)
    for i, idx in enumerate(idxs):
        plt.subplot(examples_per_class, len(classes), i * len(classes) + cls + 1)
        plt.imshow(X_test[idx].astype('uint8'))
        plt.axis('off')
        if i == 0:
            plt.title(cls_name)
plt.show()
```



Inline question 1:

Describe the misclassification results that you see. Do they make sense?

Since we are considering the color histogram feature, the background of the object comes to play. Hence in many images birds and animals are confused with the green background which normally a deer in forest has. Edge features also get higighted in HOG hence all vehicles like truck, car, van are prone to misclassification.

Neural Network on image features

Earlier in this assigment we saw that training a two-layer neural network on raw pixels achieved better classification performance than linear classifiers on raw pixels. In this notebook we have seen that linear classifiers on image features outperform linear classifiers on raw pixels.

For completeness, we should also try training a neural network on image features. This approach should outperform all previous approaches: you should easily be able to achieve over 55% classification accuracy on the test set; our best model achieves about 60% classification accuracy.

In [48]:

```
print X_train_feats.shape
```

```
(49000, 165)
```

In [99]:

```
from asgn1.classifiers.neural_net import TwoLayerNet

input_dim = X_train_feats.shape[1]
print input_dim
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

################################################################################
# TODO: Train a two-layer neural network on image features. You may want to    #
# cross-validate various parameters as in previous sections. Store your best   #
# model in the best_net variable.                                              #
################################################################################

################################################################################
#                               END OF YOUR CODE                               #
################################################################################
```

```
165
iteration 0 / 1600: loss 2.302585
iteration 100 / 1600: loss 2.252427
iteration 200 / 1600: loss 1.644659
iteration 300 / 1600: loss 1.534713
iteration 400 / 1600: loss 1.252559
iteration 500 / 1600: loss 1.328565
iteration 600 / 1600: loss 1.408849
iteration 700 / 1600: loss 1.262950
iteration 800 / 1600: loss 1.329263
iteration 900 / 1600: loss 1.249572
iteration 1000 / 1600: loss 1.221419
iteration 1100 / 1600: loss 1.218546
iteration 1200 / 1600: loss 1.272558
iteration 1300 / 1600: loss 1.158977
```