# 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.*

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]:
```
from __future__ import print_function

import random
import numpy as np
from cecs551.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 externnal modules
# see http://stackoverflow.com/questions/1907993/autoreload-of-modules-in-ipyt
hon
%load_ext autoreload
%autoreload 2
```

# Load data

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

In [2]:
```
from cecs551.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 = 'cecs551/datasets/cifar-10-batches-py'

    X_train, y_train, X_test, y_test = load_CIFAR10(cifar10_dir)

    # Subsample the data
    mask = list(range(num_training, num_training + num_validation))
    X_val = X_train[mask]
    y_val = y_train[mask]
    mask = list(range(num_training))
    X_train = X_train[mask]
    y_train = y_train[mask]
    mask = list(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

# Cleaning up variables to prevent loading data multiple times (which may caus
e memory issue)
try:
   del X_train, y_train
   del X_test, y_test
   print('Clear previously loaded data.')
except:
   pass

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 your interests.

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 column is the concatenation of all feature vectors for a single image.

In [4]:
```
from cecs551.features import *

num_color_bins = 10 # Number of bins in the color histogram
feature_fns = [hog_feature, lambda img: color_histogram_hsv(img, nbin=num_colo
r_bins)]
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 [6]:
```
# Use the validation set to tune the learning rate and regularization strength

from cecs551.classifiers.linear_classifier import LinearSVM

learning_rates = [1e-9, 1e-8, 1e-7]
regularization_strengths = [5e4, 5e5, 5e6]

results = {}
best_val = -1
best_svm = None

################################################################################
##
# 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.
#
################################################################################
##
def compute_accuracy(y, y_pred):
    return np.mean(y == y_pred)


for lr in learning_rates:
    for reg in regularization_strengths:
        # train svm
        print ("----------------------------------------------")
        print ("lr: %.8f, reg: %5.1f" %(lr, reg))
        svm = LinearSVM()
        svm.train(X_train_feats, y_train, learning_rate=lr, reg=reg, num_iters
=10000, verbose=False)

        # compute accuracy
        train_accuracy = compute_accuracy(y_train, svm.predict(X_train_feats))
        val_accuracy = compute_accuracy(y_val, svm.predict(X_val_feats))
        print ('train accuracy: %.4f' %train_accuracy)
        print ('validation accuracy: %.4f' %val_accuracy)

        # store accuracy in dictionary
        results[(lr, reg)] = (train_accuracy, val_accuracy)

        # check if validation accuracy is best
        if val_accuracy > best_val:
            best_val = val_accuracy
            best_svm = svm
################################################################################
##
#                              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_v
al)
```

```
--------------------------------------------
lr: 0.00000000, reg: 50000.0
train accuracy: 0.0985
validation accuracy: 0.0990
--------------------------------------------
lr: 0.00000000, reg: 500000.0
train accuracy: 0.1175
validation accuracy: 0.1210
--------------------------------------------
lr: 0.00000000, reg: 5000000.0
train accuracy: 0.4136
validation accuracy: 0.4160
--------------------------------------------
lr: 0.00000001, reg: 50000.0
train accuracy: 0.3150
validation accuracy: 0.3030
--------------------------------------------
lr: 0.00000001, reg: 500000.0
train accuracy: 0.4141
validation accuracy: 0.4180
--------------------------------------------
lr: 0.00000001, reg: 5000000.0
train accuracy: 0.4120
validation accuracy: 0.4120
--------------------------------------------
lr: 0.00000010, reg: 50000.0
train accuracy: 0.4133
validation accuracy: 0.4160
--------------------------------------------
lr: 0.00000010, reg: 500000.0
train accuracy: 0.4069
validation accuracy: 0.4010
--------------------------------------------
lr: 0.00000010, reg: 5000000.0
train accuracy: 0.3792
validation accuracy: 0.3860
lr 1.000000e-09 reg 5.000000e+04 train accuracy: 0.098469 val accuracy: 0.099
000
lr 1.000000e-09 reg 5.000000e+05 train accuracy: 0.117490 val accuracy: 0.121
000
lr 1.000000e-09 reg 5.000000e+06 train accuracy: 0.413551 val accuracy: 0.416
000
lr 1.000000e-08 reg 5.000000e+04 train accuracy: 0.314980 val accuracy: 0.303
000
lr 1.000000e-08 reg 5.000000e+05 train accuracy: 0.414143 val accuracy: 0.418
000
lr 1.000000e-08 reg 5.000000e+06 train accuracy: 0.412020 val accuracy: 0.412
000
lr 1.000000e-07 reg 5.000000e+04 train accuracy: 0.413286 val accuracy: 0.416
000
lr 1.000000e-07 reg 5.000000e+05 train accuracy: 0.406878 val accuracy: 0.401
000
lr 1.000000e-07 reg 5.000000e+06 train accuracy: 0.379224 val accuracy: 0.386
000
best validation accuracy achieved during cross-validation: 0.418000
```

In [7]:
```
# 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 [8]:
```
# An important way to gain intuition about how an algorithm works is to
# visualize the mistakes that it makes. In this visualization, we show example
s
# 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', 'shi
p', 'truck']
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?

## 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 [10]:
```
# Preprocessing: Remove the bias dimension
# Make sure to run this cell only ONCE
print(X_train_feats.shape)
X_train_feats = X_train_feats[:, :-1]
X_val_feats = X_val_feats[:, :-1]
X_test_feats = X_test_feats[:, :-1]

print(X_train_feats.shape)
```

```
(49000, 155)
(49000, 154)
```

In [34]:
```
from cecs551.classifiers.neural_net import TwoLayerNet

input_dim = X_train_feats.shape[1]
hidden_dim = 500
num_classes = 10

net = TwoLayerNet(input_dim, hidden_dim, num_classes)
best_net = None

################################################################################
##
# 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.
#
################################################################################
##
best_val = -1
results = {}
best_stats = None
learning_rates = [1e-3]
regularization_strengths = [0.6, 0.7]
for lr in learning_rates:
    for reg in regularization_strengths:
      # train neural network
      print ("------------------------------------------------")
      print ("Learning rate: %.4f, reg: %.2f" %(lr, reg))
      net = TwoLayerNet(input_dim, hidden_dim, num_classes)
      stats = net.train(X_train, y_train, X_val, y_val, num_iters=2000, batch_
size=200,
            learning_rate=lr, learning_rate_decay=0.95,
            reg=reg, verbose = False)
      print ('train accuracy: %.4f' %stats['train_acc_history'][-1])
      print ('validation accuracy: %.4f' %stats['val_acc_history'][-1])

      # check if validation accuracy is best or not
      if best_val < stats['val_acc_history'][-1]:
        best_val = stats['val_acc_history'][-1]
        best_net = net
        best_stats = stats

print ('best validation accuracy achieved during cross-validation: %f' % best_
val)
################################################################################
##
#                              END OF YOUR CODE
#
################################################################################
##
```

```
------------------------------------------------
Learning rate: 0.0010, reg: 0.60

---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-34-b3aa45057338> in <module>
     26     stats = net.train(X_train, y_train, X_val, y_val, num_iters=200
0, batch_size=200,
     27           learning_rate=lr, learning_rate_decay=0.95,
---> 28           reg=reg)
     29     print ('train accuracy: %.4f' %stats['train_acc_history'][-1])
     30     print ('validation accuracy: %.4f' %stats['val_acc_history'][-1
])

E:\CSULB\Spring 19\Adv AI\HW2\HW2\assignment2\cecs551\classifiers\neural_net.
py in train(self, X, y, X_val, y_val, learning_rate, learning_rate_decay, re
g, num_iters, batch_size, verbose)
    176 
    177         # Compute loss and gradients using the current minibatch
--> 178         loss, grads = self.loss(X_batch, y=y_batch, reg=reg)
    179         loss_history.append(loss)
    180 

E:\CSULB\Spring 19\Adv AI\HW2\HW2\assignment2\cecs551\classifiers\neural_net.
py in loss(self, X, y, reg)
     58     W1, b1 = self.params['W1'], self.params['b1']
     59     W2, b2 = self.params['W2'], self.params['b2']
---> 60     N, D = X.shape
     61 
     62     # Compute the forward pass

ValueError: too many values to unpack (expected 2)
```

In [30]:

```
# Run your best neural net classifier on the test set. You should be able
# to get more than 55% accuracy.

test_acc = (best_net.predict(X_test_feats) == y_test).mean()
print(test_acc)
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-30-961d08ff56b3> in <module>
      2 # to get more than 55% accuracy.
      3 
----> 4 test_acc = (best_net.predict(X_test_feats) == y_test).mean()
      5 print(test_acc)

AttributeError: 'NoneType' object has no attribute 'predict'
```

In [ ]: