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://compsci682-fa19.github.io/assignments2019/assignment1) 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]:
```
from __future__ import print_function
import random
import numpy as np
from cs682.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 cs682.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 = 'cs682/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 cause 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 [48]:
```
from cs682.features import *

num_color_bins = 25 # Number of bins in the color histogram
feature_fns = [hog_feature, lambda img: color_histogram_hsv(img, nbin=num_color_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 [84]:

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

from cs682.classifiers.linear_classifier import LinearSVM

# learning_rates = [1e-9, 1e-8, 1e-7, 1e-6, 1e-5]
# regularization_strengths = [1e3, 1e4, 5e4, 5e5, 5e6]
learning_rates = [5e-7, 1e-6, 2.5e-6, 5e-6, 7.5e-6, 1e-5, 5e-5, 1e-4, 5e-4]
regularization_strengths = [1e2, 5e2, 1e3, 5e3, 1e4, 5e4]

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.       #
################################################################################
np.random.seed(123)
num_trials = 200
for t in range(num_trials):
#   print ('learning rate: %f, regularization strength: %f' %(lr, rs))
    lr = 10 ** np.random.uniform (-7.5, -4)
    rs = 10 ** np.random.uniform (1,5)
    svm = LinearSVM()
    loss_hist = svm.train(X_train_feats, y_train, learning_rate=lr, reg=rs,
                  num_iters=1000, batch_size=400, verbose=False)
    y_train_pred = svm.predict(X_train_feats)
    train_acc = np.mean(y_train == y_train_pred)
#   print('training accuracy: %f' % (train_acc))
    y_val_pred = svm.predict(X_val_feats)
    val_acc = np.mean(y_val == y_val_pred)
#   print('validation accuracy: %f' % (val_acc))
    results [(lr, rs)] = (train_acc, val_acc)
    if (val_acc > best_val):
        best_val = val_acc
        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_val)
```

```
lr 3.215477e-08 reg 5.523815e+02 train accuracy: 0.097286 val accuracy: 0.099000
lr 3.399504e-08 reg 1.592791e+04 train accuracy: 0.128286 val accuracy: 0.130000
lr 3.552800e-08 reg 2.347738e+01 train accuracy: 0.114633 val accuracy: 0.097000
lr 3.708583e-08 reg 7.259182e+01 train accuracy: 0.104286 val accuracy: 0.128000
lr 3.731825e-08 reg 1.216398e+01 train accuracy: 0.106388 val accuracy: 0.101000
lr 3.783558e-08 reg 1.684695e+03 train accuracy: 0.107429 val accuracy: 0.095000
lr 4.687757e-08 reg 4.343593e+01 train accuracy: 0.104122 val accuracy: 0.100000
lr 4.959981e-08 reg 1.646587e+02 train accuracy: 0.122776 val accuracy: 0.140000
lr 5.361862e-08 reg 2.310790e+03 train accuracy: 0.115020 val accuracy: 0.126000
lr 5.424113e-08 reg 1.926131e+02 train accuracy: 0.107918 val accuracy: 0.099000
lr 5.579342e-08 reg 6.632885e+03 train accuracy: 0.104265 val accuracy: 0.104000
lr 5.931982e-08 reg 1.682655e+03 train accuracy: 0.096510 val accuracy: 0.093000
lr 6.057849e-08 reg 1.225719e+02 train accuracy: 0.110592 val accuracy: 0.107000
lr 6.099074e-08 reg 1.313430e+04 train accuracy: 0.112918 val accuracy: 0.108000
lr 6.138025e-08 reg 6.599516e+01 train accuracy: 0.099776 val accuracy: 0.103000
lr 6.241379e-08 reg 3.684925e+03 train accuracy: 0.097796 val accuracy: 0.085000
lr 6.679378e-08 reg 9.758148e+03 train accuracy: 0.100490 val accuracy: 0.092000
lr 7.336283e-08 reg 1.650692e+01 train accuracy: 0.104469 val accuracy: 0.108000
lr 7.753666e-08 reg 3.373391e+03 train accuracy: 0.127143 val accuracy: 0.125000
lr 8.371576e-08 reg 7.639510e+02 train accuracy: 0.125000 val accuracy: 0.126000
lr 9.673969e-08 reg 4.534152e+01 train accuracy: 0.101184 val accuracy: 0.112000
lr 1.019249e-07 reg 7.395322e+03 train accuracy: 0.111327 val accuracy: 0.104000
lr 1.047759e-07 reg 1.489352e+01 train accuracy: 0.096633 val accuracy: 0.105000
lr 1.062432e-07 reg 2.380446e+03 train accuracy: 0.111204 val accuracy: 0.118000
lr 1.085271e-07 reg 8.482845e+01 train accuracy: 0.113367 val accuracy: 0.121000
lr 1.120190e-07 reg 2.836335e+02 train accuracy: 0.126347 val accuracy: 0.129000
lr 1.121666e-07 reg 2.854463e+04 train accuracy: 0.414490 val accuracy: 0.407000
lr 1.160666e-07 reg 1.330320e+03 train accuracy: 0.118429 val accuracy: 0.115000
lr 1.206724e-07 reg 1.382925e+03 train accuracy: 0.103082 val accuracy: 0.109000
lr 1.210385e-07 reg 1.684383e+04 train accuracy: 0.281041 val accuracy: 0.271000
lr 1.215481e-07 reg 4.409044e+02 train accuracy: 0.127367 val accuracy: 0.125000
lr 1.417455e-07 reg 1.992695e+02 train accuracy: 0.114224 val accuracy: 0.102000
lr 1.561872e-07 reg 5.839544e+02 train accuracy: 0.114571 val accuracy: 0.121000
lr 1.563581e-07 reg 1.216483e+04 train accuracy: 0.282837 val accuracy: 0.296000
lr 1.610030e-07 reg 4.254853e+03 train accuracy: 0.130347 val accuracy: 0.121000
lr 1.668774e-07 reg 2.391839e+04 train accuracy: 0.422510 val accuracy: 0.427000
lr 1.867786e-07 reg 2.206809e+04 train accuracy: 0.422102 val accuracy: 0.418000
lr 1.884455e-07 reg 6.781663e+04 train accuracy: 0.420714 val accuracy: 0.427000
lr 1.997262e-07 reg 6.179427e+04 train accuracy: 0.422694 val accuracy: 0.431000
lr 2.060173e-07 reg 2.313650e+01 train accuracy: 0.151531 val accuracy: 0.144000
lr 2.139225e-07 reg 2.623846e+01 train accuracy: 0.138959 val accuracy: 0.151000
lr 2.152326e-07 reg 6.915241e+03 train accuracy: 0.256061 val accuracy: 0.287000
lr 2.185422e-07 reg 3.195334e+03 train accuracy: 0.161388 val accuracy: 0.154000
lr 2.274074e-07 reg 1.604818e+04 train accuracy: 0.419102 val accuracy: 0.416000
lr 2.361381e-07 reg 6.145687e+02 train accuracy: 0.122286 val accuracy: 0.117000
lr 2.400752e-07 reg 2.328278e+03 train accuracy: 0.173898 val accuracy: 0.152000
lr 2.495423e-07 reg 3.957439e+03 train accuracy: 0.198776 val accuracy: 0.183000
lr 2.609098e-07 reg 1.992900e+03 train accuracy: 0.140816 val accuracy: 0.140000
lr 2.646037e-07 reg 3.780309e+03 train accuracy: 0.216980 val accuracy: 0.236000
lr 2.746492e-07 reg 1.043268e+03 train accuracy: 0.156673 val accuracy: 0.173000
lr 2.764092e-07 reg 1.453555e+01 train accuracy: 0.132939 val accuracy: 0.114000
lr 3.050717e-07 reg 2.202507e+02 train accuracy: 0.130653 val accuracy: 0.137000
lr 3.063412e-07 reg 5.771449e+02 train accuracy: 0.169000 val accuracy: 0.163000
lr 3.127928e-07 reg 1.419143e+02 train accuracy: 0.127510 val accuracy: 0.126000
lr 3.165317e-07 reg 7.538234e+01 train accuracy: 0.128796 val accuracy: 0.125000
lr 3.347130e-07 reg 4.111003e+02 train accuracy: 0.162041 val accuracy: 0.181000
lr 3.452989e-07 reg 4.577037e+02 train accuracy: 0.144776 val accuracy: 0.132000
lr 3.572813e-07 reg 4.482377e+03 train accuracy: 0.324714 val accuracy: 0.321000
lr 3.663415e-07 reg 6.621154e+02 train accuracy: 0.146143 val accuracy: 0.143000
lr 3.738338e-07 reg 1.877676e+01 train accuracy: 0.169327 val accuracy: 0.162000
lr 3.828175e-07 reg 4.523546e+04 train accuracy: 0.421306 val accuracy: 0.413000
lr 3.892380e-07 reg 1.149872e+01 train accuracy: 0.147918 val accuracy: 0.141000
lr 3.924042e-07 reg 7.626411e+01 train accuracy: 0.144184 val accuracy: 0.123000
lr 3.939285e-07 reg 1.341938e+01 train accuracy: 0.152184 val accuracy: 0.149000
lr 3.957572e-07 reg 3.208169e+02 train accuracy: 0.133204 val accuracy: 0.140000
lr 4.262258e-07 reg 1.895436e+01 train accuracy: 0.162020 val accuracy: 0.191000
lr 4.584292e-07 reg 1.690614e+02 train accuracy: 0.143592 val accuracy: 0.150000
lr 4.662920e-07 reg 1.537846e+02 train accuracy: 0.144653 val accuracy: 0.145000
lr 4.701563e-07 reg 2.988316e+04 train accuracy: 0.422163 val accuracy: 0.417000
lr 5.187767e-07 reg 2.934000e+03 train accuracy: 0.353388 val accuracy: 0.345000
lr 5.361621e-07 reg 2.697898e+02 train accuracy: 0.174490 val accuracy: 0.186000
lr 5.457122e-07 reg 1.346060e+03 train accuracy: 0.234735 val accuracy: 0.238000
lr 5.606072e-07 reg 2.864756e+02 train accuracy: 0.189592 val accuracy: 0.174000
lr 5.621451e-07 reg 1.372526e+02 train accuracy: 0.153204 val accuracy: 0.171000
lr 5.929073e-07 reg 3.050488e+04 train accuracy: 0.421408 val accuracy: 0.421000
lr 6.069741e-07 reg 1.744836e+03 train accuracy: 0.282184 val accuracy: 0.289000
```

```
lr 6.128538e-07 reg 7.147501e+02 train accuracy: 0.213490 val accuracy: 0.217000
lr 6.210454e-07 reg 4.739310e+01 train accuracy: 0.184347 val accuracy: 0.209000
lr 6.634091e-07 reg 1.322160e+03 train accuracy: 0.265020 val accuracy: 0.259000
lr 6.773313e-07 reg 2.084070e+02 train accuracy: 0.203429 val accuracy: 0.180000
lr 6.891545e-07 reg 1.479571e+03 train accuracy: 0.305122 val accuracy: 0.307000
lr 6.925374e-07 reg 2.097842e+01 train accuracy: 0.159061 val accuracy: 0.163000
lr 7.160274e-07 reg 1.915747e+01 train accuracy: 0.185939 val accuracy: 0.185000
lr 7.168276e-07 reg 1.134728e+01 train accuracy: 0.164265 val accuracy: 0.166000
lr 7.211071e-07 reg 1.591265e+04 train accuracy: 0.419184 val accuracy: 0.405000
lr 7.279735e-07 reg 2.221416e+01 train accuracy: 0.163551 val accuracy: 0.176000
lr 7.544392e-07 reg 3.078819e+01 train accuracy: 0.162367 val accuracy: 0.133000
lr 7.740266e-07 reg 1.083192e+02 train accuracy: 0.197857 val accuracy: 0.182000
lr 7.803893e-07 reg 1.218351e+03 train accuracy: 0.299163 val accuracy: 0.281000
lr 8.228810e-07 reg 5.096185e+03 train accuracy: 0.424082 val accuracy: 0.432000
lr 8.801491e-07 reg 2.235453e+03 train accuracy: 0.414673 val accuracy: 0.420000
lr 9.131907e-07 reg 4.348301e+02 train accuracy: 0.242061 val accuracy: 0.257000
lr 9.359920e-07 reg 9.036733e+04 train accuracy: 0.410490 val accuracy: 0.408000
lr 9.868756e-07 reg 1.370181e+02 train accuracy: 0.217776 val accuracy: 0.203000
lr 1.008992e-06 reg 4.130031e+03 train accuracy: 0.421204 val accuracy: 0.420000
lr 1.063085e-06 reg 4.924258e+02 train accuracy: 0.288408 val accuracy: 0.317000
lr 1.093926e-06 reg 9.191740e+03 train accuracy: 0.423878 val accuracy: 0.425000
lr 1.162361e-06 reg 4.070752e+04 train accuracy: 0.414102 val accuracy: 0.405000
lr 1.206688e-06 reg 2.715384e+01 train accuracy: 0.212286 val accuracy: 0.233000
lr 1.248321e-06 reg 7.498278e+01 train accuracy: 0.254673 val accuracy: 0.277000
lr 1.256727e-06 reg 9.226247e+01 train accuracy: 0.215020 val accuracy: 0.231000
lr 1.327664e-06 reg 1.127984e+04 train accuracy: 0.422571 val accuracy: 0.432000
lr 1.328575e-06 reg 2.544168e+01 train accuracy: 0.263898 val accuracy: 0.267000
lr 1.386882e-06 reg 1.554533e+04 train accuracy: 0.419694 val accuracy: 0.416000
lr 1.472344e-06 reg 3.961437e+02 train accuracy: 0.327306 val accuracy: 0.338000
lr 1.505270e-06 reg 2.625025e+04 train accuracy: 0.420327 val accuracy: 0.415000
lr 1.560074e-06 reg 2.622690e+02 train accuracy: 0.310245 val accuracy: 0.303000
lr 1.592095e-06 reg 1.277238e+01 train accuracy: 0.257245 val accuracy: 0.260000
lr 1.599685e-06 reg 6.811864e+03 train accuracy: 0.419673 val accuracy: 0.424000
lr 1.680692e-06 reg 7.949338e+01 train accuracy: 0.276571 val accuracy: 0.291000
lr 1.807304e-06 reg 3.930772e+04 train accuracy: 0.418204 val accuracy: 0.426000
lr 1.834229e-06 reg 3.080823e+04 train accuracy: 0.415163 val accuracy: 0.422000
lr 1.857352e-06 reg 1.080742e+03 train accuracy: 0.419592 val accuracy: 0.412000
lr 1.898195e-06 reg 1.946133e+03 train accuracy: 0.421796 val accuracy: 0.432000
lr 1.905938e-06 reg 3.883260e+02 train accuracy: 0.363490 val accuracy: 0.366000
lr 1.947440e-06 reg 3.262071e+04 train accuracy: 0.417143 val accuracy: 0.403000
lr 1.980101e-06 reg 5.758813e+04 train accuracy: 0.407857 val accuracy: 0.405000
lr 2.222751e-06 reg 6.287541e+04 train accuracy: 0.403959 val accuracy: 0.391000
lr 2.248450e-06 reg 6.719434e+02 train accuracy: 0.417571 val accuracy: 0.420000
lr 2.293663e-06 reg 8.004866e+03 train accuracy: 0.423122 val accuracy: 0.412000
lr 2.327069e-06 reg 2.635884e+02 train accuracy: 0.378204 val accuracy: 0.371000
lr 2.350620e-06 reg 8.490437e+02 train accuracy: 0.422286 val accuracy: 0.412000
lr 2.617494e-06 reg 2.497538e+04 train accuracy: 0.417592 val accuracy: 0.426000
lr 2.759967e-06 reg 1.061740e+02 train accuracy: 0.357327 val accuracy: 0.364000
lr 2.780461e-06 reg 2.744577e+04 train accuracy: 0.410796 val accuracy: 0.399000
lr 2.970717e-06 reg 9.735884e+03 train accuracy: 0.424531 val accuracy: 0.429000
lr 3.245099e-06 reg 1.136722e+02 train accuracy: 0.384735 val accuracy: 0.384000
lr 3.384527e-06 reg 1.008812e+02 train accuracy: 0.372122 val accuracy: 0.345000
lr 3.390337e-06 reg 2.508739e+03 train accuracy: 0.421551 val accuracy: 0.417000
lr 3.466104e-06 reg 1.371255e+01 train accuracy: 0.353551 val accuracy: 0.378000
lr 3.481548e-06 reg 6.327564e+04 train accuracy: 0.404204 val accuracy: 0.394000
lr 3.728607e-06 reg 3.004157e+01 train accuracy: 0.370755 val accuracy: 0.348000
lr 3.949796e-06 reg 2.504298e+01 train accuracy: 0.367286 val accuracy: 0.384000
lr 3.959333e-06 reg 4.250235e+04 train accuracy: 0.405673 val accuracy: 0.391000
lr 4.105120e-06 reg 1.011434e+04 train accuracy: 0.421327 val accuracy: 0.436000
lr 4.291552e-06 reg 4.179273e+03 train accuracy: 0.421143 val accuracy: 0.410000
lr 4.361936e-06 reg 5.261546e+04 train accuracy: 0.387143 val accuracy: 0.375000
lr 4.391907e-06 reg 7.379842e+03 train accuracy: 0.421592 val accuracy: 0.421000
lr 4.441526e-06 reg 1.337092e+02 train accuracy: 0.402531 val accuracy: 0.396000
lr 4.646572e-06 reg 1.958151e+02 train accuracy: 0.416020 val accuracy: 0.430000
lr 4.893116e-06 reg 1.912094e+03 train accuracy: 0.421429 val accuracy: 0.425000
lr 5.349085e-06 reg 5.113398e+01 train accuracy: 0.403041 val accuracy: 0.411000
lr 5.485054e-06 reg 7.309816e+03 train accuracy: 0.415082 val accuracy: 0.412000
lr 6.218083e-06 reg 2.940384e+03 train accuracy: 0.423633 val accuracy: 0.423000
lr 6.320323e-06 reg 2.420311e+02 train accuracy: 0.422878 val accuracy: 0.424000
lr 6.558777e-06 reg 1.401850e+03 train accuracy: 0.421898 val accuracy: 0.414000
lr 6.784752e-06 reg 1.371298e+02 train accuracy: 0.418510 val accuracy: 0.411000
lr 6.908389e-06 reg 8.631326e+04 train accuracy: 0.355041 val accuracy: 0.356000
lr 8.557884e-06 reg 7.591051e+04 train accuracy: 0.302224 val accuracy: 0.327000
lr 8.662478e-06 reg 1.394946e+02 train accuracy: 0.421143 val accuracy: 0.415000
lr 8.820757e-06 reg 1.146374e+03 train accuracy: 0.422918 val accuracy: 0.426000
lr 8.953919e-06 reg 3.683136e+04 train accuracy: 0.376918 val accuracy: 0.378000
```

```
lr 8.990801e-06 reg 5.629061e+03 train accuracy: 0.412061 val accuracy: 0.415000
lr 1.021579e-05 reg 3.057906e+02 train accuracy: 0.422959 val accuracy: 0.427000
lr 1.174242e-05 reg 2.264260e+01 train accuracy: 0.421571 val accuracy: 0.424000
lr 1.231112e-05 reg 2.248526e+03 train accuracy: 0.424000 val accuracy: 0.434000
lr 1.235419e-05 reg 8.029376e+02 train accuracy: 0.420857 val accuracy: 0.422000
lr 1.291702e-05 reg 2.821390e+04 train accuracy: 0.377694 val accuracy: 0.383000
lr 1.363308e-05 reg 3.004173e+02 train accuracy: 0.422061 val accuracy: 0.423000
lr 1.369823e-05 reg 2.418206e+04 train accuracy: 0.393061 val accuracy: 0.398000
lr 1.458569e-05 reg 1.652118e+04 train accuracy: 0.391531 val accuracy: 0.407000
lr 1.474567e-05 reg 9.787518e+03 train accuracy: 0.406551 val accuracy: 0.384000
lr 1.507723e-05 reg 2.558308e+04 train accuracy: 0.394429 val accuracy: 0.406000
lr 1.581974e-05 reg 3.298719e+02 train accuracy: 0.421469 val accuracy: 0.410000
lr 1.693449e-05 reg 1.074951e+02 train accuracy: 0.423551 val accuracy: 0.424000
lr 1.700689e-05 reg 9.911403e+02 train accuracy: 0.422673 val accuracy: 0.421000
lr 1.724904e-05 reg 7.054054e+02 train accuracy: 0.426898 val accuracy: 0.440000
lr 1.735279e-05 reg 5.692641e+04 train accuracy: 0.185347 val accuracy: 0.174000
lr 1.796524e-05 reg 1.057199e+04 train accuracy: 0.403429 val accuracy: 0.397000
lr 1.866064e-05 reg 9.198029e+03 train accuracy: 0.416061 val accuracy: 0.411000
lr 1.998988e-05 reg 1.377366e+01 train accuracy: 0.420041 val accuracy: 0.426000
lr 2.035542e-05 reg 5.266937e+04 train accuracy: 0.096469 val accuracy: 0.099000
lr 2.280849e-05 reg 1.604221e+03 train accuracy: 0.417551 val accuracy: 0.426000
lr 2.380328e-05 reg 1.195698e+04 train accuracy: 0.396673 val accuracy: 0.403000
lr 2.390705e-05 reg 2.225763e+02 train accuracy: 0.423735 val accuracy: 0.416000
lr 2.579595e-05 reg 6.507990e+03 train accuracy: 0.389918 val accuracy: 0.384000
lr 2.623867e-05 reg 1.367072e+04 train accuracy: 0.385755 val accuracy: 0.382000
lr 2.677629e-05 reg 3.523630e+03 train accuracy: 0.411245 val accuracy: 0.409000
lr 2.711507e-05 reg 1.064365e+04 train accuracy: 0.390102 val accuracy: 0.378000
lr 2.931307e-05 reg 3.833057e+03 train accuracy: 0.415286 val accuracy: 0.417000
lr 3.278565e-05 reg 5.403398e+02 train accuracy: 0.420122 val accuracy: 0.420000
lr 3.489113e-05 reg 8.753801e+03 train accuracy: 0.393000 val accuracy: 0.400000
lr 3.504692e-05 reg 7.462352e+01 train accuracy: 0.422265 val accuracy: 0.422000
lr 3.648821e-05 reg 7.379983e+02 train accuracy: 0.419531 val accuracy: 0.419000
lr 4.065921e-05 reg 2.804168e+01 train accuracy: 0.422490 val accuracy: 0.424000
lr 4.195540e-05 reg 3.163226e+01 train accuracy: 0.421673 val accuracy: 0.424000
lr 4.359904e-05 reg 1.767997e+02 train accuracy: 0.421490 val accuracy: 0.430000
lr 4.500832e-05 reg 2.420866e+03 train accuracy: 0.406061 val accuracy: 0.407000
lr 4.720947e-05 reg 6.716030e+02 train accuracy: 0.419633 val accuracy: 0.418000
lr 4.772076e-05 reg 1.114902e+01 train accuracy: 0.424204 val accuracy: 0.434000
lr 5.231552e-05 reg 1.079649e+02 train accuracy: 0.422245 val accuracy: 0.424000
lr 5.246390e-05 reg 2.575231e+01 train accuracy: 0.421735 val accuracy: 0.426000
lr 5.833050e-05 reg 5.683436e+01 train accuracy: 0.422571 val accuracy: 0.420000
lr 7.307664e-05 reg 2.048875e+02 train accuracy: 0.416551 val accuracy: 0.411000
lr 7.399916e-05 reg 1.289937e+02 train accuracy: 0.421592 val accuracy: 0.425000
lr 7.424579e-05 reg 1.605049e+01 train accuracy: 0.424367 val accuracy: 0.426000
lr 7.638568e-05 reg 6.064908e+02 train accuracy: 0.415245 val accuracy: 0.407000
lr 7.923079e-05 reg 3.758559e+02 train accuracy: 0.426224 val accuracy: 0.434000
lr 8.297855e-05 reg 7.731149e+03 train accuracy: 0.317633 val accuracy: 0.320000
lr 8.490819e-05 reg 2.933328e+04 train accuracy: 0.100265 val accuracy: 0.087000
best validation accuracy achieved during cross-validation: 0.440000
```

In [85]:
```
# 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)
```

```
0.436
```

In [86]:
```
# 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', '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?

Answer: The misclassification reflects the low capacity of the SVM model. The classifier is unable to distinguish between images that have similar hues, color ranges and texture patterns but belong to different classes.

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 [52]:
```
# 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, 170)
(49000, 169)
```

In [63]:
```
from cs682.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.                                              #
################################################################################
num_classes = 10

hidden_sizes = [100, 200, 400, 800, 1600]
learning_rates = [1, 1e-1]
regularizations = [1e-4, 1e-3, 1e-2]
batch_sizes = [100, 200, 400, 800]
best_val_acc = 0
best_hyperparams = None
for hs in hidden_sizes:
    for bs in batch_sizes:
        num_trials = 5
        for t in range(num_trials):
            # Train the network
            lr = 10 ** np.random.uniform (-1,0)
            rs = 10 ** np.random.uniform (-4,-2)
            # Train the network
            net = TwoLayerNet(input_dim, hs, num_classes)
            stats = net.train(X_train_feats, y_train, X_val_feats, y_val,
                        num_iters=2000, batch_size=bs,
                        learning_rate=lr, learning_rate_decay=0.95,
                        reg=reg, verbose=False)

            # Predict on the train set
            train_acc = (net.predict(X_train_feats) == y_train).mean()

            # Predict on the validation set
            val_acc = (net.predict(X_val_feats) == y_val).mean()
            print (val_acc, train_acc, hs, bs, lr, reg)
            if best_val_acc < val_acc:
                best_val_acc = val_acc
                best_net = net
                best_hyper_params = (val_acc, train_acc, hs, bs, lr, reg)
print('Best validation accuracy: ', best_val_acc)
print('Best hyper params: ', best_hyper_params)
################################################################################
#                              END OF YOUR CODE                                #
################################################################################
```

0.55 0.6047142857142858 100 100 1 0.0001
0.529 0.5761632653061225 100 100 1 0.001
0.465 0.4729795918367347 100 100 1 0.01
0.543 0.5570816326530612 100 100 0.1 0.0001
0.534 0.5554897959183673 100 100 0.1 0.001
0.521 0.5203061224489796 100 100 0.1 0.01
0.56 0.6576122448979592 100 200 1 0.0001
0.59 0.6366122448979592 100 200 1 0.001
0.493 0.5062040816326531 100 200 1 0.01
0.54 0.5576122448979591 100 200 0.1 0.0001
0.529 0.5529795918367347 100 200 0.1 0.001
0.521 0.526469387755102 100 200 0.1 0.01
0.56 0.6863673469387755 100 400 1 0.0001
0.578 0.6570816326530612 100 400 1 0.001
0.496 0.5086734693877552 100 400 1 0.01
0.537 0.5588775510204081 100 400 0.1 0.0001
0.54 0.5551836734693878 100 400 0.1 0.001
0.516 0.5275714285714286 100 400 0.1 0.01
0.573 0.7108571428571429 100 800 1 0.0001
0.576 0.6875918367346939 100 800 1 0.001
0.513 0.5312244897959184 100 800 1 0.01
0.55 0.5608775510204081 100 800 0.1 0.0001
0.539 0.556 100 800 0.1 0.001
0.515 0.5290408163265307 100 800 0.1 0.01
0.558 0.6370816326530612 200 100 1 0.0001
0.546 0.5980408163265306 200 100 1 0.001
0.471 0.4766530612244898 200 100 1 0.01
0.534 0.5606326530612245 200 100 0.1 0.0001
0.539 0.5564285714285714 200 100 0.1 0.001
0.525 0.5199183673469387 200 100 0.1 0.01
0.567 0.6975918367346938 200 200 1 0.0001
0.57 0.6599591836734694 200 200 1 0.001
0.497 0.5138571428571429 200 200 1 0.01
0.557 0.5609795918367347 200 200 0.1 0.0001
0.534 0.5525714285714286 200 200 0.1 0.001
0.528 0.5267755102040816 200 200 0.1 0.01
0.583 0.7654897959183673 200 400 1 0.0001
0.59 0.711469387755102 200 400 1 0.001
0.491 0.5113877551020408 200 400 1 0.01
0.547 0.5623673469387755 200 400 0.1 0.0001
0.544 0.557795918367347 200 400 0.1 0.001
0.514 0.5262040816326531 200 400 0.1 0.01
0.561 0.790530612244898 200 800 1 0.0001
0.595 0.7328775510204082 200 800 1 0.001
0.542 0.5281020408163265 200 800 1 0.01
0.543 0.5629387755102041 200 800 0.1 0.0001
0.535 0.5585102040816327 200 800 0.1 0.001
0.515 0.5276530612244898 200 800 0.1 0.01
0.564 0.6476326530612245 400 100 1 0.0001
0.512 0.5950204081632653 400 100 1 0.001
0.469 0.4692857142857143 400 100 1 0.01
0.555 0.566204081632653 400 100 0.1 0.0001
0.544 0.559795918367347 400 100 0.1 0.001
0.508 0.5196530612244898 400 100 0.1 0.01
0.57 0.7428979591836735 400 200 1 0.0001
0.588 0.6970816326530612 400 200 1 0.001
0.487 0.4926938775510204 400 200 1 0.01
0.55 0.5624489795918367 400 200 0.1 0.0001
0.538 0.5561836734693878 400 200 0.1 0.001
0.52 0.5270204081632653 400 200 0.1 0.01
0.58 0.8164081632653061 400 400 1 0.0001
0.614 0.7345306122448979 400 400 1 0.001
0.527 0.5236530612244898 400 400 1 0.01
0.547 0.5638775510204082 400 400 0.1 0.0001
0.543 0.5593469387755102 400 400 0.1 0.001
0.513 0.5279183673469388 400 400 0.1 0.01
0.557 0.8463469387755101 400 800 1 0.0001
0.588 0.7664081632653061 400 800 1 0.001
0.522 0.5323673469387755 400 800 1 0.01
0.548 0.566 400 800 0.1 0.0001
0.544 0.5595102040816327 400 800 0.1 0.001
0.515 0.5293877551020408 400 800 0.1 0.01
0.554 0.6726326530612244 800 100 1 0.0001
0.562 0.6278775510204082 800 100 1 0.001
0.47 0.4774285714285714 800 100 1 0.01
0.542 0.570734693877551 800 100 0.1 0.0001

```
0.537 0.5623877551020409 800 100 0.1 0.001
0.511 0.5193469387755102 800 100 0.1 0.01
0.557 0.7961632653061225 800 200 1 0.0001
0.584 0.695734693877551 800 200 1 0.001
0.493 0.49389795918367346 800 200 1 0.01
0.544 0.5645714285714286 800 200 0.1 0.0001
0.546 0.5611224489795918 800 200 0.1 0.001
0.517 0.5264081632653062 800 200 0.1 0.01
0.57 0.8538163265306122 800 400 1 0.0001
0.602 0.7631836734693878 800 400 1 0.001
0.521 0.525 800 400 1 0.01
0.542 0.5671836734693878 800 400 0.1 0.0001
0.545 0.561469387755102 800 400 0.1 0.001
0.519 0.5283061224489796 800 400 0.1 0.01
0.593 0.933 800 800 1 0.0001
0.604 0.7998571428571428 800 800 1 0.001
0.521 0.5389795918367347 800 800 1 0.01
0.549 0.5683469387755102 800 800 0.1 0.0001
0.548 0.5622448979591836 800 800 0.1 0.001
0.519 0.5290204081632653 800 800 0.1 0.01
0.513 0.672530612244898 1600 100 1 0.0001
0.556 0.6290408163265306 1600 100 1 0.001
0.489 0.4944081632653061 1600 100 1 0.01
0.548 0.5693877551020409 1600 100 0.1 0.0001
0.545 0.560734693877551 1600 100 0.1 0.001
0.516 0.5200204081632653 1600 100 0.1 0.01
0.585 0.8117142857142857 1600 200 1 0.0001
0.588 0.7153061224489796 1600 200 1 0.001
0.471 0.4927142857142857 1600 200 1 0.01
0.558 0.5676530612244898 1600 200 0.1 0.0001
0.545 0.562 1600 200 0.1 0.001
0.528 0.5276938775510204 1600 200 0.1 0.01
0.607 0.8999183673469388 1600 400 1 0.0001
0.619 0.7750612244897959 1600 400 1 0.001
0.547 0.5244081632653061 1600 400 1 0.01
0.556 0.569469387755102 1600 400 0.1 0.0001
0.543 0.5635510204081633 1600 400 0.1 0.001
0.525 0.5288775510204081 1600 400 0.1 0.01
0.591 0.9576530612244898 1600 800 1 0.0001
0.616 0.8164081632653061 1600 800 1 0.001
0.518 0.5301632653061225 1600 800 1 0.01
0.555 0.5710408163265306 1600 800 0.1 0.0001
0.551 0.5639795918367347 1600 800 0.1 0.001
0.529 0.5310408163265307 1600 800 0.1 0.01
Best validation accuracy:  0.619
Best hyper params:  (0.619, 0.7750612244897959, 1600, 400, 1, 0.001)
```

In [64]:
```
# 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)
```

```
0.599
```

In [ ]: