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]() 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 cs231n.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
print ("done")
```

```
done
```

Load data

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

In [2]:

```
from cs231n.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 = 'cs231n/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()
print ("setup done\n")
```

```
setup done
```

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 [3]:

```
from cs231n.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_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))])
print ("done here")
```

```
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 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
done here
```

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 [5]:

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

from cs231n.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.       #
################################################################################

best_lr = None
best_reg = None

range_lr = np.linspace(learning_rates[0],learning_rates[1],5)
range_reg = np.linspace(regularization_strengths[0],regularization_strengths[1],5)

print ("strating loop ! This may take a while\n\n")
# loop through all the combinations
for cur_lr in range_lr: #go over the learning rates
    for cur_reg in range_reg:#go over the regularization strength
        # initiate linear classifier with hyperparameters
        svm = LinearSVM()
        svm.train(X_train_feats, y_train, learning_rate=cur_lr, reg=cur_reg,num_iters=1600, verbose
=True)

        # Training
        y_pred = svm.predict(X_train_feats)
        train_accuracy = np.mean(np.equal(y_train, y_pred, dtype=float))


        # Validation
        y_pred = svm.predict(X_val_feats)
        val_accuracy = np.mean(np.equal(y_val, y_pred, dtype=float))

        results[(cur_lr, cur_reg)] = (train_accuracy, val_accuracy)

        if val_accuracy > best_val:
            best_val = val_accuracy
            best_svm = svm
            best_lr = cur_lr
            best_reg = cur_reg




# 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)
```

```
strating loop ! This may take a while


X shape   (49000, 155)
iteration 0 / 1600: loss 47.650660
iteration 100 / 1600: loss 47.277811
iteration 200 / 1600: loss 46.892079
iteration 300 / 1600: loss 46.513451
iteration 400 / 1600: loss 46.136074
iteration 500 / 1600: loss 45.771115
iteration 600 / 1600: loss 45.391001
iteration 700 / 1600: loss 45.050272
iteration 800 / 1600: loss 44.675026
iteration 900 / 1600: loss 44.313301
iteration 1000 / 1600: loss 43.957297
iteration 1100 / 1600: loss 43.626499
iteration 1200 / 1600: loss 43.275254
iteration 1300 / 1600: loss 42.937266
iteration 1400 / 1600: loss 42.598399
iteration 1500 / 1600: loss 42.268559
X shape   (49000, 155)
iteration 0 / 1600: loss 132.244065
iteration 100 / 1600: loss 128.304558
iteration 200 / 1600: loss 124.504448
iteration 300 / 1600: loss 120.810138
iteration 400 / 1600: loss 117.210564
iteration 500 / 1600: loss 113.762904
iteration 600 / 1600: loss 110.403294
iteration 700 / 1600: loss 107.173281
iteration 800 / 1600: loss 104.030066
iteration 900 / 1600: loss 100.994789
iteration 1000 / 1600: loss 98.044505
iteration 1100 / 1600: loss 95.187813
iteration 1200 / 1600: loss 92.447095
iteration 1300 / 1600: loss 89.771675
iteration 1400 / 1600: loss 87.187570
iteration 1500 / 1600: loss 84.686131
X shape   (49000, 155)
iteration 0 / 1600: loss 212.754564
iteration 100 / 1600: loss 201.838976
iteration 200 / 1600: loss 191.524155
iteration 300 / 1600: loss 181.750277
iteration 400 / 1600: loss 172.510087
iteration 500 / 1600: loss 163.753328
iteration 600 / 1600: loss 155.467490
iteration 700 / 1600: loss 147.637129
iteration 800 / 1600: loss 140.207991
iteration 900 / 1600: loss 133.193956
iteration 1000 / 1600: loss 126.553552
iteration 1100 / 1600: loss 120.242690
iteration 1200 / 1600: loss 114.289387
iteration 1300 / 1600: loss 108.656472
iteration 1400 / 1600: loss 103.334271
iteration 1500 / 1600: loss 98.270954
X shape   (49000, 155)
iteration 0 / 1600: loss 297.352521
iteration 100 / 1600: loss 275.843804
iteration 200 / 1600: loss 255.935280
iteration 300 / 1600: loss 237.522019
iteration 400 / 1600: loss 220.488559
iteration 500 / 1600: loss 204.711977
iteration 600 / 1600: loss 190.111026
iteration 700 / 1600: loss 176.606645
iteration 800 / 1600: loss 164.111579
iteration 900 / 1600: loss 152.528056
iteration 1000 / 1600: loss 141.823409
iteration 1100 / 1600: loss 131.921573
iteration 1200 / 1600: loss 122.755478
iteration 1300 / 1600: loss 114.261816
iteration 1400 / 1600: loss 106.410973
iteration 1500 / 1600: loss 99.144999
X shape   (49000, 155)
iteration 0 / 1600: loss 403.105268
iteration 100 / 1600: loss 365.582786
iteration 200 / 1600: loss 331.654764
iteration 300 / 1600: loss 300.948918
iteration 400 / 1600: loss 273.163072
iteration 500 / 1600: loss 247.992836
iteration 600 / 1600: loss 225.264087
iteration 700 / 1600: loss 204.671213
iteration 800 / 1600: loss 186.049321
iteration 900 / 1600: loss 169.191251
iteration 1000 / 1600: loss 153.942127
iteration 1100 / 1600: loss 140.149808
iteration 1200 / 1600: loss 127.665078
iteration 1300 / 1600: loss 116.368314
iteration 1400 / 1600: loss 106.151446
iteration 1500 / 1600: loss 96.900922
X shape   (49000, 155)
iteration 0 / 1600: loss 49.210447
iteration 100 / 1600: loss 47.927295
iteration 200 / 1600: loss 46.677469
iteration 300 / 1600: loss 45.465489
iteration 400 / 1600: loss 44.297914
iteration 500 / 1600: loss 43.176929
iteration 600 / 1600: loss 42.087005
iteration 700 / 1600: loss 41.023196
iteration 800 / 1600: loss 39.997521
iteration 900 / 1600: loss 39.010061
iteration 1000 / 1600: loss 38.054178
iteration 1100 / 1600: loss 37.132981
iteration 1200 / 1600: loss 36.227173
iteration 1300 / 1600: loss 35.351321
iteration 1400 / 1600: loss 34.508729
iteration 1500 / 1600: loss 33.691397
X shape   (49000, 155)
iteration 0 / 1600: loss 134.912966
iteration 100 / 1600: loss 122.294057
iteration 200 / 1600: loss 110.949095
iteration 300 / 1600: loss 100.714956
iteration 400 / 1600: loss 91.523611
iteration 500 / 1600: loss 83.255655
iteration 600 / 1600: loss 75.807739
iteration 700 / 1600: loss 69.106644
iteration 800 / 1600: loss 63.076547
iteration 900 / 1600: loss 57.660297
iteration 1000 / 1600: loss 52.771796
iteration 1100 / 1600: loss 48.387648
iteration 1200 / 1600: loss 44.435088
iteration 1300 / 1600: loss 40.883567
iteration 1400 / 1600: loss 37.685448
iteration 1500 / 1600: loss 34.810020
X shape   (49000, 155)
iteration 0 / 1600: loss 222.638727
iteration 100 / 1600: loss 187.640225
iteration 200 / 1600: loss 158.380613
iteration 300 / 1600: loss 133.922389
iteration 400 / 1600: loss 113.463086
iteration 500 / 1600: loss 96.354946
iteration 600 / 1600: loss 82.051437
iteration 700 / 1600: loss 70.097094
iteration 800 / 1600: loss 60.096990
iteration 900 / 1600: loss 51.728974
iteration 1000 / 1600: loss 44.727497
iteration 1100 / 1600: loss 38.876556
iteration 1200 / 1600: loss 33.982117
iteration 1300 / 1600: loss 29.893141
iteration 1400 / 1600: loss 26.469825
iteration 1500 / 1600: loss 23.610470
X shape   (49000, 155)
iteration 0 / 1600: loss 317.225352
iteration 100 / 1600: loss 248.544719
iteration 200 / 1600: loss 195.178366
iteration 300 / 1600: loss 153.701559
iteration 400 / 1600: loss 121.473577
iteration 500 / 1600: loss 96.405925
iteration 600 / 1600: loss 76.933502
iteration 700 / 1600: loss 61.801130
iteration 800 / 1600: loss 50.038664
iteration 900 / 1600: loss 40.894426
iteration 1000 / 1600: loss 33.792082
iteration 1100 / 1600: loss 28.266951
iteration 1200 / 1600: loss 23.974886
iteration 1300 / 1600: loss 20.640730
iteration 1400 / 1600: loss 18.047079
iteration 1500 / 1600: loss 16.029199
X shape   (49000, 155)
iteration 0 / 1600: loss 403.516599
iteration 100 / 1600: loss 293.954638
iteration 200 / 1600: loss 214.850621
iteration 300 / 1600: loss 157.678388
iteration 400 / 1600: loss 116.406933
iteration 500 / 1600: loss 86.576801
iteration 600 / 1600: loss 65.042556
iteration 700 / 1600: loss 49.480855
iteration 800 / 1600: loss 38.238582
iteration 900 / 1600: loss 30.118978
iteration 1000 / 1600: loss 24.255067
iteration 1100 / 1600: loss 20.020058
iteration 1200 / 1600: loss 16.959936
iteration 1300 / 1600: loss 14.749765
iteration 1400 / 1600: loss 13.153291
iteration 1500 / 1600: loss 11.999703
X shape   (49000, 155)
iteration 0 / 1600: loss 49.178972
iteration 100 / 1600: loss 47.031960
iteration 200 / 1600: loss 44.998759
iteration 300 / 1600: loss 43.072350
iteration 400 / 1600: loss 41.253487
iteration 500 / 1600: loss 39.518897
iteration 600 / 1600: loss 37.889538
iteration 700 / 1600: loss 36.336856
iteration 800 / 1600: loss 34.884378
iteration 900 / 1600: loss 33.485373
iteration 1000 / 1600: loss 32.185083
iteration 1100 / 1600: loss 30.944639
iteration 1200 / 1600: loss 29.760973
iteration 1300 / 1600: loss 28.650960
iteration 1400 / 1600: loss 27.601485
iteration 1500 / 1600: loss 26.609421
X shape   (49000, 155)
iteration 0 / 1600: loss 130.408620
iteration 100 / 1600: loss 110.529009
iteration 200 / 1600: loss 93.898367
iteration 300 / 1600: loss 79.997236
iteration 400 / 1600: loss 68.371022
iteration 500 / 1600: loss 58.648089
iteration 600 / 1600: loss 50.516638
iteration 700 / 1600: loss 43.720916
iteration 800 / 1600: loss 38.029094
iteration 900 / 1600: loss 33.282663
iteration 1000 / 1600: loss 29.303409
iteration 1100 / 1600: loss 25.982536
iteration 1200 / 1600: loss 23.203212
iteration 1300 / 1600: loss 20.873287
iteration 1400 / 1600: loss 18.931308
iteration 1500 / 1600: loss 17.303865
```

```
X shape   (49000, 155)
iteration 0 / 1600: loss 217.824416
iteration 100 / 1600: loss 163.289751
iteration 200 / 1600: loss 122.990836
iteration 300 / 1600: loss 93.211117
iteration 400 / 1600: loss 71.219452
iteration 500 / 1600: loss 54.961415
iteration 600 / 1600: loss 42.962528
iteration 700 / 1600: loss 34.089839
iteration 800 / 1600: loss 27.533962
iteration 900 / 1600: loss 22.696983
iteration 1000 / 1600: loss 19.120100
iteration 1100 / 1600: loss 16.474381
iteration 1200 / 1600: loss 14.522289
iteration 1300 / 1600: loss 13.081795
iteration 1400 / 1600: loss 12.013729
iteration 1500 / 1600: loss 11.227075
X shape   (49000, 155)
iteration 0 / 1600: loss 302.900270
iteration 100 / 1600: loss 200.830513
iteration 200 / 1600: loss 134.192374
iteration 300 / 1600: loss 90.716863
iteration 400 / 1600: loss 62.328986
iteration 500 / 1600: loss 43.804336
iteration 600 / 1600: loss 31.716825
iteration 700 / 1600: loss 23.822956
iteration 800 / 1600: loss 18.675258
iteration 900 / 1600: loss 15.316113
iteration 1000 / 1600: loss 13.119664
iteration 1100 / 1600: loss 11.690648
iteration 1200 / 1600: loss 10.755231
iteration 1300 / 1600: loss 10.145348
iteration 1400 / 1600: loss 9.747933
iteration 1500 / 1600: loss 9.487759
X shape   (49000, 155)
iteration 0 / 1600: loss 403.599774
iteration 100 / 1600: loss 236.502020
iteration 200 / 1600: loss 140.156982
iteration 300 / 1600: loss 84.611497
iteration 400 / 1600: loss 52.593245
iteration 500 / 1600: loss 34.132668
iteration 600 / 1600: loss 23.486387
iteration 700 / 1600: loss 17.353692
iteration 800 / 1600: loss 13.815103
iteration 900 / 1600: loss 11.775551
iteration 1000 / 1600: loss 10.600755
iteration 1100 / 1600: loss 9.923125
iteration 1200 / 1600: loss 9.531907
iteration 1300 / 1600: loss 9.306853
iteration 1400 / 1600: loss 9.176434
iteration 1500 / 1600: loss 9.101579
X shape   (49000, 155)
iteration 0 / 1600: loss 47.657456
iteration 100 / 1600: loss 44.763659
iteration 200 / 1600: loss 42.087064
iteration 300 / 1600: loss 39.634056
iteration 400 / 1600: loss 37.315477
iteration 500 / 1600: loss 35.234896
iteration 600 / 1600: loss 33.277023
iteration 700 / 1600: loss 31.455779
iteration 800 / 1600: loss 29.794778
iteration 900 / 1600: loss 28.234308
iteration 1000 / 1600: loss 26.805859
iteration 1100 / 1600: loss 25.472995
iteration 1200 / 1600: loss 24.248200
iteration 1300 / 1600: loss 23.111971
iteration 1400 / 1600: loss 22.051056
iteration 1500 / 1600: loss 21.084494
X shape   (49000, 155)
iteration 0 / 1600: loss 138.311154
iteration 100 / 1600: loss 109.506488
iteration 200 / 1600: loss 87.103627
iteration 300 / 1600: loss 69.704277
iteration 400 / 1600: loss 56.182195
iteration 500 / 1600: loss 45.669599
iteration 600 / 1600: loss 37.490264
iteration 700 / 1600: loss 31.152385
iteration 800 / 1600: loss 26.216556
iteration 900 / 1600: loss 22.385011
iteration 1000 / 1600: loss 19.402310
iteration 1100 / 1600: loss 17.083487
iteration 1200 / 1600: loss 15.283283
iteration 1300 / 1600: loss 13.881455
iteration 1400 / 1600: loss 12.793279
iteration 1500 / 1600: loss 11.946857
X shape   (49000, 155)
iteration 0 / 1600: loss 224.581048
iteration 100 / 1600: loss 149.714439
iteration 200 / 1600: loss 100.824905
iteration 300 / 1600: loss 68.940249
iteration 400 / 1600: loss 48.118175
iteration 500 / 1600: loss 34.528545
iteration 600 / 1600: loss 25.666241
iteration 700 / 1600: loss 19.873196
iteration 800 / 1600: loss 16.095987
iteration 900 / 1600: loss 13.634015
iteration 1000 / 1600: loss 12.020742
iteration 1100 / 1600: loss 10.971346
iteration 1200 / 1600: loss 10.287630
iteration 1300 / 1600: loss 9.839899
iteration 1400 / 1600: loss 9.547980
iteration 1500 / 1600: loss 9.357791
X shape   (49000, 155)
iteration 0 / 1600: loss 308.954281
iteration 100 / 1600: loss 173.360300
iteration 200 / 1600: loss 99.056315
iteration 300 / 1600: loss 58.358394
iteration 400 / 1600: loss 36.046558
iteration 500 / 1600: loss 23.821556
iteration 600 / 1600: loss 17.120419
iteration 700 / 1600: loss 13.450470
iteration 800 / 1600: loss 11.436645
iteration 900 / 1600: loss 10.336032
iteration 1000 / 1600: loss 9.731930
iteration 1100 / 1600: loss 9.401285
iteration 1200 / 1600: loss 9.219926
iteration 1300 / 1600: loss 9.120481
iteration 1400 / 1600: loss 9.066129
iteration 1500 / 1600: loss 9.036153
X shape   (49000, 155)
iteration 0 / 1600: loss 414.869810
iteration 100 / 1600: loss 195.691410
iteration 200 / 1600: loss 94.886511
iteration 300 / 1600: loss 48.513653
iteration 400 / 1600: loss 27.175055
iteration 500 / 1600: loss 17.359730
iteration 600 / 1600: loss 12.846188
iteration 700 / 1600: loss 10.769515
iteration 800 / 1600: loss 9.813188
iteration 900 / 1600: loss 9.374410
iteration 1000 / 1600: loss 9.171955
iteration 1100 / 1600: loss 9.079046
iteration 1200 / 1600: loss 9.036382
iteration 1300 / 1600: loss 9.016741
iteration 1400 / 1600: loss 9.007640
iteration 1500 / 1600: loss 9.003504
X shape   (49000, 155)
iteration 0 / 1600: loss 48.440545
iteration 100 / 1600: loss 44.688410
iteration 200 / 1600: loss 41.300883
iteration 300 / 1600: loss 38.228060
iteration 400 / 1600: loss 35.431651
iteration 500 / 1600: loss 32.920978
iteration 600 / 1600: loss 30.649028
iteration 700 / 1600: loss 28.577650
iteration 800 / 1600: loss 26.717261
iteration 900 / 1600: loss 25.031622
iteration 1000 / 1600: loss 23.506140
iteration 1100 / 1600: loss 22.127586
iteration 1200 / 1600: loss 20.877425
iteration 1300 / 1600: loss 19.746093
iteration 1400 / 1600: loss 18.724800
iteration 1500 / 1600: loss 17.801814
X shape   (49000, 155)
```

```
iteration 0 / 1600: loss 137.062174
iteration 100 / 1600: loss 101.503885
iteration 200 / 1600: loss 75.816582
iteration 300 / 1600: loss 57.269019
iteration 400 / 1600: loss 43.863155
iteration 500 / 1600: loss 34.183268
iteration 600 / 1600: loss 27.188853
iteration 700 / 1600: loss 22.141753
iteration 800 / 1600: loss 18.490423
iteration 900 / 1600: loss 15.856321
iteration 1000 / 1600: loss 13.955633
iteration 1100 / 1600: loss 12.578099
iteration 1200 / 1600: loss 11.583582
iteration 1300 / 1600: loss 10.865409
iteration 1400 / 1600: loss 10.347177
iteration 1500 / 1600: loss 9.973478
X shape   (49000, 155)
iteration 0 / 1600: loss 220.916078
iteration 100 / 1600: loss 131.165681
iteration 200 / 1600: loss 79.436134
iteration 300 / 1600: loss 49.608431
iteration 400 / 1600: loss 32.409883
iteration 500 / 1600: loss 22.496316
iteration 600 / 1600: loss 16.778930
iteration 700 / 1600: loss 13.485490
iteration 800 / 1600: loss 11.585910
iteration 900 / 1600: loss 10.491423
iteration 1000 / 1600: loss 9.858887
iteration 1100 / 1600: loss 9.495600
iteration 1200 / 1600: loss 9.285728
iteration 1300 / 1600: loss 9.164575
iteration 1400 / 1600: loss 9.094790
iteration 1500 / 1600: loss 9.054379
X shape   (49000, 155)
iteration 0 / 1600: loss 304.032240
iteration 100 / 1600: loss 144.716071
iteration 200 / 1600: loss 71.429755
iteration 300 / 1600: loss 37.716516
iteration 400 / 1600: loss 22.212571
iteration 500 / 1600: loss 15.077792
iteration 600 / 1600: loss 11.795255
iteration 700 / 1600: loss 10.286036
iteration 800 / 1600: loss 9.590799
iteration 900 / 1600: loss 9.272089
iteration 1000 / 1600: loss 9.125194
iteration 1100 / 1600: loss 9.057366
iteration 1200 / 1600: loss 9.026311
iteration 1300 / 1600: loss 9.012125
iteration 1400 / 1600: loss 9.005629
iteration 1500 / 1600: loss 9.002489
X shape   (49000, 155)
iteration 0 / 1600: loss 401.956902
iteration 100 / 1600: loss 153.194450
iteration 200 / 1600: loss 61.912979
iteration 300 / 1600: loss 28.415891
iteration 400 / 1600: loss 16.125517
iteration 500 / 1600: loss 11.613843
iteration 600 / 1600: loss 9.959500
iteration 700 / 1600: loss 9.351939
iteration 800 / 1600: loss 9.129035
iteration 900 / 1600: loss 9.047305
iteration 1000 / 1600: loss 9.017468
iteration 1100 / 1600: loss 9.006313
iteration 1200 / 1600: loss 9.002267
iteration 1300 / 1600: loss 9.000786
iteration 1400 / 1600: loss 9.000265
iteration 1500 / 1600: loss 9.000055
lr 1.000000e-09 reg 5.000000e+04 train accuracy: 0.097857 val accuracy: 0.112000
lr 1.000000e-09 reg 1.625000e+05 train accuracy: 0.118408 val accuracy: 0.109000
lr 1.000000e-09 reg 2.750000e+05 train accuracy: 0.108755 val accuracy: 0.116000
lr 1.000000e-09 reg 3.875000e+05 train accuracy: 0.100918 val accuracy: 0.107000
lr 1.000000e-09 reg 5.000000e+05 train accuracy: 0.086143 val accuracy: 0.101000
lr 3.250000e-09 reg 5.000000e+04 train accuracy: 0.096286 val accuracy: 0.098000
lr 3.250000e-09 reg 1.625000e+05 train accuracy: 0.101286 val accuracy: 0.106000
lr 3.250000e-09 reg 2.750000e+05 train accuracy: 0.072939 val accuracy: 0.092000
lr 3.250000e-09 reg 3.875000e+05 train accuracy: 0.099837 val accuracy: 0.097000
lr 3.250000e-09 reg 5.000000e+05 train accuracy: 0.099898 val accuracy: 0.098000
lr 5.500000e-09 reg 5.000000e+04 train accuracy: 0.106857 val accuracy: 0.116000
lr 5.500000e-09 reg 1.625000e+05 train accuracy: 0.090408 val accuracy: 0.100000
lr 5.500000e-09 reg 2.750000e+05 train accuracy: 0.105367 val accuracy: 0.099000
lr 5.500000e-09 reg 3.875000e+05 train accuracy: 0.126041 val accuracy: 0.138000
lr 5.500000e-09 reg 5.000000e+05 train accuracy: 0.080143 val accuracy: 0.065000
lr 7.750000e-09 reg 5.000000e+04 train accuracy: 0.087449 val accuracy: 0.082000
lr 7.750000e-09 reg 1.625000e+05 train accuracy: 0.112490 val accuracy: 0.102000
lr 7.750000e-09 reg 2.750000e+05 train accuracy: 0.128939 val accuracy: 0.134000
lr 7.750000e-09 reg 3.875000e+05 train accuracy: 0.108449 val accuracy: 0.131000
lr 7.750000e-09 reg 5.000000e+05 train accuracy: 0.225551 val accuracy: 0.220000
lr 1.000000e-08 reg 5.000000e+04 train accuracy: 0.105633 val accuracy: 0.130000
lr 1.000000e-08 reg 1.625000e+05 train accuracy: 0.135592 val accuracy: 0.142000
lr 1.000000e-08 reg 2.750000e+05 train accuracy: 0.151327 val accuracy: 0.153000
lr 1.000000e-08 reg 3.875000e+05 train accuracy: 0.215020 val accuracy: 0.220000
lr 1.000000e-08 reg 5.000000e+05 train accuracy: 0.381796 val accuracy: 0.407000
best validation accuracy achieved during cross-validation: 0.407000
```

In [7]:

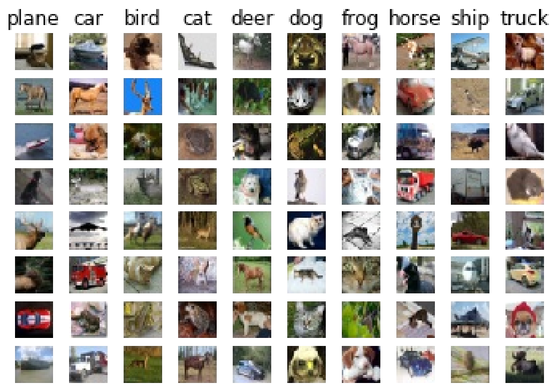
```
# Evaluate your trained SVM on the test set
y_test_pred = best_svm.predict(X_test_feats)
test_accuracy = np.mean(y_test == y_test_pred)
print("test accuracy :",test_accuracy)
```

```
test accuracy : 0.401
```

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 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 :To be honest I am failing to find any common features amongst the misclassified images within each class. Perhaps this is a reflection of the quality of the design of the two layer network, maybe a deeper analysis on each learned feature.

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 [9]:

```
# 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 [10]:

```
from cs231n.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.                                              #
################################################################################

# defaults

num_classes = 10

# hyperparameters
learning_rates = [1,1e-3]
regularization_strengths = [0, 1]
hidden_sizes = [100,800]
batch_sizes = [100,600]

# additional variables
best_lr = None
best_reg = None
best_hidd = None
best_batch_size = None

# tune list of hyperparameters used
range_lr = np.linspace(learning_rates[0],learning_rates[1],6)
range_reg = np.linspace(regularization_strengths[0],regularization_strengths[1],3)
range_hs = [200,300,400,500,600,700]
range_bs = [200,300, 400]

best_val_acc = -1
print ("starting optimization, this may take a while :(")
for hs in range_hs:
    for lr in range_lr:
        for bs in range_bs:
            for rg in range_reg:
                print ('hs:', hs, 'lr:', lr, 'bs', bs, 'reg', rg)

                net = TwoLayerNet(input_dim, hs, output_size=num_classes)

                # train
                stats = net.train(X_train_feats, y_train, X_val_feats,
y_val,num_iters=500,learning_rate=lr,reg=rg, batch_size=bs)
                # check validation accuracy
                val_acc = (net.predict(X_val_feats) == y_val).mean()

                #print ('hs:', hs, 'lr:', lr, 'bs', bs, 'reg', rg)
                print ('Validation accuracy: ', val_acc)

                # store optimal hyperparameters
                if val_acc > best_val_acc:
                    best_val_acc = val_acc
                    best_net = net
                    best_lr = lr
                    best_reg = rg
                    best_hidd = hs
                    best_batch_size = bs

print ("\n** best validation achieved during cross validation: " ,best_val_acc)
```

```
starting optimization, this may take a while :(
hs: 200 lr: 1.0 bs 200 reg 0.0
Validation accuracy:  0.51
hs: 200 lr: 1.0 bs 200 reg 0.5
Validation accuracy:  0.119
hs: 200 lr: 1.0 bs 200 reg 1.0
Validation accuracy:  0.079
hs: 200 lr: 1.0 bs 300 reg 0.0
Validation accuracy:  0.543
hs: 200 lr: 1.0 bs 300 reg 0.5
Validation accuracy:  0.112
hs: 200 lr: 1.0 bs 300 reg 1.0
Validation accuracy:  0.078
hs: 200 lr: 1.0 bs 400 reg 0.0
Validation accuracy:  0.569
hs: 200 lr: 1.0 bs 400 reg 0.5
Validation accuracy:  0.079
hs: 200 lr: 1.0 bs 400 reg 1.0
Validation accuracy:  0.119
hs: 200 lr: 0.8002 bs 200 reg 0.0
Validation accuracy:  0.533
hs: 200 lr: 0.8002 bs 200 reg 0.5
Validation accuracy:  0.087
hs: 200 lr: 0.8002 bs 200 reg 1.0
Validation accuracy:  0.113
hs: 200 lr: 0.8002 bs 300 reg 0.0
Validation accuracy:  0.537
hs: 200 lr: 0.8002 bs 300 reg 0.5
Validation accuracy:  0.105
hs: 200 lr: 0.8002 bs 300 reg 1.0
Validation accuracy:  0.113
hs: 200 lr: 0.8002 bs 400 reg 0.0
Validation accuracy:  0.55
hs: 200 lr: 0.8002 bs 400 reg 0.5
Validation accuracy:  0.078
hs: 200 lr: 0.8002 bs 400 reg 1.0
Validation accuracy:  0.079
hs: 200 lr: 0.6004 bs 200 reg 0.0
Validation accuracy:  0.554
hs: 200 lr: 0.6004 bs 200 reg 0.5
Validation accuracy:  0.119
hs: 200 lr: 0.6004 bs 200 reg 1.0
Validation accuracy:  0.119
hs: 200 lr: 0.6004 bs 300 reg 0.0
Validation accuracy:  0.552
hs: 200 lr: 0.6004 bs 300 reg 0.5
Validation accuracy:  0.107
hs: 200 lr: 0.6004 bs 300 reg 1.0
Validation accuracy:  0.107
hs: 200 lr: 0.6004 bs 400 reg 0.0
Validation accuracy:  0.559
hs: 200 lr: 0.6004 bs 400 reg 0.5
Validation accuracy:  0.079
hs: 200 lr: 0.6004 bs 400 reg 1.0
Validation accuracy:  0.112
hs: 200 lr: 0.40059999999999996 bs 200 reg 0.0
Validation accuracy:  0.536
hs: 200 lr: 0.40059999999999996 bs 200 reg 0.5
```

```
hs: 200 lr: 0.40059999999999996 bs 200 reg 0.5
Validation accuracy:  0.107
hs: 200 lr: 0.40059999999999996 bs 200 reg 1.0
Validation accuracy:  0.112
hs: 200 lr: 0.40059999999999996 bs 300 reg 0.0
Validation accuracy:  0.541
hs: 200 lr: 0.40059999999999996 bs 300 reg 0.5
Validation accuracy:  0.098
hs: 200 lr: 0.40059999999999996 bs 300 reg 1.0
Validation accuracy:  0.078
hs: 200 lr: 0.40059999999999996 bs 400 reg 0.0
Validation accuracy:  0.529
hs: 200 lr: 0.40059999999999996 bs 400 reg 0.5
Validation accuracy:  0.113
hs: 200 lr: 0.40059999999999996 bs 400 reg 1.0
Validation accuracy:  0.087
hs: 200 lr: 0.20079999999999998 bs 200 reg 0.0
Validation accuracy:  0.499
hs: 200 lr: 0.20079999999999998 bs 200 reg 0.5
Validation accuracy:  0.102
hs: 200 lr: 0.20079999999999998 bs 200 reg 1.0
Validation accuracy:  0.079
hs: 200 lr: 0.20079999999999998 bs 300 reg 0.0
Validation accuracy:  0.508
hs: 200 lr: 0.20079999999999998 bs 300 reg 0.5
Validation accuracy:  0.113
hs: 200 lr: 0.20079999999999998 bs 300 reg 1.0
Validation accuracy:  0.112
hs: 200 lr: 0.20079999999999998 bs 400 reg 0.0
Validation accuracy:  0.514
hs: 200 lr: 0.20079999999999998 bs 400 reg 0.5
Validation accuracy:  0.078
hs: 200 lr: 0.20079999999999998 bs 400 reg 1.0
Validation accuracy:  0.102
hs: 200 lr: 0.001 bs 200 reg 0.0
Validation accuracy:  0.102
hs: 200 lr: 0.001 bs 200 reg 0.5
Validation accuracy:  0.107
hs: 200 lr: 0.001 bs 200 reg 1.0
Validation accuracy:  0.078
hs: 200 lr: 0.001 bs 300 reg 0.0
Validation accuracy:  0.087
hs: 200 lr: 0.001 bs 300 reg 0.5
Validation accuracy:  0.112
hs: 200 lr: 0.001 bs 300 reg 1.0
Validation accuracy:  0.11
hs: 200 lr: 0.001 bs 400 reg 0.0
Validation accuracy:  0.079
hs: 200 lr: 0.001 bs 400 reg 0.5
Validation accuracy:  0.113
hs: 200 lr: 0.001 bs 400 reg 1.0
Validation accuracy:  0.079
hs: 300 lr: 1.0 bs 200 reg 0.0
Validation accuracy:  0.558
hs: 300 lr: 1.0 bs 200 reg 0.5
Validation accuracy:  0.098
hs: 300 lr: 1.0 bs 200 reg 1.0
Validation accuracy:  0.087
hs: 300 lr: 1.0 bs 300 reg 0.0
Validation accuracy:  0.512
hs: 300 lr: 1.0 bs 300 reg 0.5
Validation accuracy:  0.079
hs: 300 lr: 1.0 bs 300 reg 1.0
Validation accuracy:  0.102
hs: 300 lr: 1.0 bs 400 reg 0.0
Validation accuracy:  0.57
hs: 300 lr: 1.0 bs 400 reg 0.5
Validation accuracy:  0.087
hs: 300 lr: 1.0 bs 400 reg 1.0
Validation accuracy:  0.078
hs: 300 lr: 0.8002 bs 200 reg 0.0
Validation accuracy:  0.554
hs: 300 lr: 0.8002 bs 200 reg 0.5
Validation accuracy:  0.098
hs: 300 lr: 0.8002 bs 200 reg 1.0
Validation accuracy:  0.112
hs: 300 lr: 0.8002 bs 300 reg 0.0
Validation accuracy:  0.559
```

```
hs: 300 lr: 0.8002 bs 300 reg 0.5
Validation accuracy:  0.087
hs: 300 lr: 0.8002 bs 300 reg 1.0
Validation accuracy:  0.119
hs: 300 lr: 0.8002 bs 400 reg 0.0
Validation accuracy:  0.553
hs: 300 lr: 0.8002 bs 400 reg 0.5
Validation accuracy:  0.078
hs: 300 lr: 0.8002 bs 400 reg 1.0
Validation accuracy:  0.102
hs: 300 lr: 0.6004 bs 200 reg 0.0
Validation accuracy:  0.546
hs: 300 lr: 0.6004 bs 200 reg 0.5
Validation accuracy:  0.113
hs: 300 lr: 0.6004 bs 200 reg 1.0
Validation accuracy:  0.079
hs: 300 lr: 0.6004 bs 300 reg 0.0
Validation accuracy:  0.524
hs: 300 lr: 0.6004 bs 300 reg 0.5
Validation accuracy:  0.107
hs: 300 lr: 0.6004 bs 300 reg 1.0
Validation accuracy:  0.119
hs: 300 lr: 0.6004 bs 400 reg 0.0
Validation accuracy:  0.571
hs: 300 lr: 0.6004 bs 400 reg 0.5
Validation accuracy:  0.078
hs: 300 lr: 0.6004 bs 400 reg 1.0
Validation accuracy:  0.078
hs: 300 lr: 0.40059999999999996 bs 200 reg 0.0
Validation accuracy:  0.538
hs: 300 lr: 0.40059999999999996 bs 200 reg 0.5
Validation accuracy:  0.107
hs: 300 lr: 0.40059999999999996 bs 200 reg 1.0
Validation accuracy:  0.087
hs: 300 lr: 0.40059999999999996 bs 300 reg 0.0
Validation accuracy:  0.531
hs: 300 lr: 0.40059999999999996 bs 300 reg 0.5
Validation accuracy:  0.098
hs: 300 lr: 0.40059999999999996 bs 300 reg 1.0
Validation accuracy:  0.113
hs: 300 lr: 0.40059999999999996 bs 400 reg 0.0
Validation accuracy:  0.545
hs: 300 lr: 0.40059999999999996 bs 400 reg 0.5
Validation accuracy:  0.078
hs: 300 lr: 0.40059999999999996 bs 400 reg 1.0
Validation accuracy:  0.087
hs: 300 lr: 0.20079999999999998 bs 200 reg 0.0
Validation accuracy:  0.53
hs: 300 lr: 0.20079999999999998 bs 200 reg 0.5
Validation accuracy:  0.087
hs: 300 lr: 0.20079999999999998 bs 200 reg 1.0
Validation accuracy:  0.079
hs: 300 lr: 0.20079999999999998 bs 300 reg 0.0
Validation accuracy:  0.524
hs: 300 lr: 0.20079999999999998 bs 300 reg 0.5
Validation accuracy:  0.098
hs: 300 lr: 0.20079999999999998 bs 300 reg 1.0
Validation accuracy:  0.105
hs: 300 lr: 0.20079999999999998 bs 400 reg 0.0
Validation accuracy:  0.515
hs: 300 lr: 0.20079999999999998 bs 400 reg 0.5
Validation accuracy:  0.087
hs: 300 lr: 0.20079999999999998 bs 400 reg 1.0
Validation accuracy:  0.079
hs: 300 lr: 0.001 bs 200 reg 0.0
Validation accuracy:  0.102
hs: 300 lr: 0.001 bs 200 reg 0.5
Validation accuracy:  0.102
hs: 300 lr: 0.001 bs 200 reg 1.0
Validation accuracy:  0.078
hs: 300 lr: 0.001 bs 300 reg 0.0
Validation accuracy:  0.079
hs: 300 lr: 0.001 bs 300 reg 0.5
Validation accuracy:  0.098
hs: 300 lr: 0.001 bs 300 reg 1.0
Validation accuracy:  0.087
hs: 300 lr: 0.001 bs 400 reg 0.0
```

```
hs: 300 lr: 0.001 bs 400 reg 0.0
Validation accuracy:  0.112
hs: 300 lr: 0.001 bs 400 reg 0.5
Validation accuracy:  0.078
hs: 300 lr: 0.001 bs 400 reg 1.0
Validation accuracy:  0.102
hs: 400 lr: 1.0 bs 200 reg 0.0
Validation accuracy:  0.547
hs: 400 lr: 1.0 bs 200 reg 0.5
Validation accuracy:  0.087
hs: 400 lr: 1.0 bs 200 reg 1.0
Validation accuracy:  0.112
hs: 400 lr: 1.0 bs 300 reg 0.0
Validation accuracy:  0.53
hs: 400 lr: 1.0 bs 300 reg 0.5
Validation accuracy:  0.105
hs: 400 lr: 1.0 bs 300 reg 1.0
Validation accuracy:  0.119
hs: 400 lr: 1.0 bs 400 reg 0.0
Validation accuracy:  0.566
hs: 400 lr: 1.0 bs 400 reg 0.5
Validation accuracy:  0.119
hs: 400 lr: 1.0 bs 400 reg 1.0
Validation accuracy:  0.105
hs: 400 lr: 0.8002 bs 200 reg 0.0
Validation accuracy:  0.561
hs: 400 lr: 0.8002 bs 200 reg 0.5
Validation accuracy:  0.078
hs: 400 lr: 0.8002 bs 200 reg 1.0
Validation accuracy:  0.078
hs: 400 lr: 0.8002 bs 300 reg 0.0
Validation accuracy:  0.565
hs: 400 lr: 0.8002 bs 300 reg 0.5
Validation accuracy:  0.079
hs: 400 lr: 0.8002 bs 300 reg 1.0
Validation accuracy:  0.078
hs: 400 lr: 0.8002 bs 400 reg 0.0
Validation accuracy:  0.552
hs: 400 lr: 0.8002 bs 400 reg 0.5
Validation accuracy:  0.119
hs: 400 lr: 0.8002 bs 400 reg 1.0
Validation accuracy:  0.112
hs: 400 lr: 0.6004 bs 200 reg 0.0
Validation accuracy:  0.556
hs: 400 lr: 0.6004 bs 200 reg 0.5
Validation accuracy:  0.102
hs: 400 lr: 0.6004 bs 200 reg 1.0
Validation accuracy:  0.107
hs: 400 lr: 0.6004 bs 300 reg 0.0
Validation accuracy:  0.56
hs: 400 lr: 0.6004 bs 300 reg 0.5
Validation accuracy:  0.105
hs: 400 lr: 0.6004 bs 300 reg 1.0
Validation accuracy:  0.079
hs: 400 lr: 0.6004 bs 400 reg 0.0
Validation accuracy:  0.553
hs: 400 lr: 0.6004 bs 400 reg 0.5
Validation accuracy:  0.107
hs: 400 lr: 0.6004 bs 400 reg 1.0
Validation accuracy:  0.107
hs: 400 lr: 0.40059999999999996 bs 200 reg 0.0
Validation accuracy:  0.528
hs: 400 lr: 0.40059999999999996 bs 200 reg 0.5
Validation accuracy:  0.112
hs: 400 lr: 0.40059999999999996 bs 200 reg 1.0
Validation accuracy:  0.078
hs: 400 lr: 0.40059999999999996 bs 300 reg 0.0
Validation accuracy:  0.541
hs: 400 lr: 0.40059999999999996 bs 300 reg 0.5
Validation accuracy:  0.087
hs: 400 lr: 0.40059999999999996 bs 300 reg 1.0
Validation accuracy:  0.107
hs: 400 lr: 0.40059999999999996 bs 400 reg 0.0
Validation accuracy:  0.539
hs: 400 lr: 0.40059999999999996 bs 400 reg 0.5
Validation accuracy:  0.119
hs: 400 lr: 0.40059999999999996 bs 400 reg 1.0
Validation accuracy:  0.078
Validation accuracy:  0.078
hs: 400 lr: 0.20079999999999998 bs 200 reg 0.0
Validation accuracy:  0.501
hs: 400 lr: 0.20079999999999998 bs 200 reg 0.5
Validation accuracy:  0.079
hs: 400 lr: 0.20079999999999998 bs 200 reg 1.0
Validation accuracy:  0.079
hs: 400 lr: 0.20079999999999998 bs 300 reg 0.0
Validation accuracy:  0.524
hs: 400 lr: 0.20079999999999998 bs 300 reg 0.5
Validation accuracy:  0.112
hs: 400 lr: 0.20079999999999998 bs 300 reg 1.0
Validation accuracy:  0.087
hs: 400 lr: 0.20079999999999998 bs 400 reg 0.0
Validation accuracy:  0.501
hs: 400 lr: 0.20079999999999998 bs 400 reg 0.5
Validation accuracy:  0.078
hs: 400 lr: 0.20079999999999998 bs 400 reg 1.0
Validation accuracy:  0.105
hs: 400 lr: 0.001 bs 200 reg 0.0
Validation accuracy:  0.079
hs: 400 lr: 0.001 bs 200 reg 0.5
Validation accuracy:  0.078
hs: 400 lr: 0.001 bs 200 reg 1.0
Validation accuracy:  0.098
hs: 400 lr: 0.001 bs 300 reg 0.0
Validation accuracy:  0.102
hs: 400 lr: 0.001 bs 300 reg 0.5
Validation accuracy:  0.079
hs: 400 lr: 0.001 bs 300 reg 1.0
Validation accuracy:  0.078
hs: 400 lr: 0.001 bs 400 reg 0.0
Validation accuracy:  0.098
hs: 400 lr: 0.001 bs 400 reg 0.5
Validation accuracy:  0.078
hs: 400 lr: 0.001 bs 400 reg 1.0
Validation accuracy:  0.079
hs: 500 lr: 1.0 bs 200 reg 0.0
Validation accuracy:  0.551
hs: 500 lr: 1.0 bs 200 reg 0.5
Validation accuracy:  0.078
hs: 500 lr: 1.0 bs 200 reg 1.0
Validation accuracy:  0.102
hs: 500 lr: 1.0 bs 300 reg 0.0
Validation accuracy:  0.552
hs: 500 lr: 1.0 bs 300 reg 0.5
Validation accuracy:  0.112
hs: 500 lr: 1.0 bs 300 reg 1.0
Validation accuracy:  0.105
hs: 500 lr: 1.0 bs 400 reg 0.0
Validation accuracy:  0.57
hs: 500 lr: 1.0 bs 400 reg 0.5
Validation accuracy:  0.107
hs: 500 lr: 1.0 bs 400 reg 1.0
Validation accuracy:  0.098
hs: 500 lr: 0.8002 bs 200 reg 0.0
Validation accuracy:  0.562
hs: 500 lr: 0.8002 bs 200 reg 0.5
Validation accuracy:  0.112
hs: 500 lr: 0.8002 bs 200 reg 1.0
Validation accuracy:  0.098
hs: 500 lr: 0.8002 bs 300 reg 0.0
Validation accuracy:  0.541
hs: 500 lr: 0.8002 bs 300 reg 0.5
Validation accuracy:  0.078
hs: 500 lr: 0.8002 bs 300 reg 1.0
Validation accuracy:  0.102
hs: 500 lr: 0.8002 bs 400 reg 0.0
Validation accuracy:  0.56
hs: 500 lr: 0.8002 bs 400 reg 0.5
Validation accuracy:  0.119
hs: 500 lr: 0.8002 bs 400 reg 1.0
Validation accuracy:  0.098
hs: 500 lr: 0.6004 bs 200 reg 0.0
Validation accuracy:  0.541
hs: 500 lr: 0.6004 bs 200 reg 0.5
Validation accuracy:  0.107
hs: 500 lr: 0.6004 bs 200 reg 1.0
```

```
hs: 500 lr: 0.6004 bs 200 reg 1.0
Validation accuracy:  0.078
hs: 500 lr: 0.6004 bs 300 reg 0.0
Validation accuracy:  0.544
hs: 500 lr: 0.6004 bs 300 reg 0.5
Validation accuracy:  0.098
hs: 500 lr: 0.6004 bs 300 reg 1.0
Validation accuracy:  0.087
hs: 500 lr: 0.6004 bs 400 reg 0.0
Validation accuracy:  0.559
hs: 500 lr: 0.6004 bs 400 reg 0.5
Validation accuracy:  0.105
hs: 500 lr: 0.6004 bs 400 reg 1.0
Validation accuracy:  0.079
hs: 500 lr: 0.40059999999999996 bs 200 reg 0.0
Validation accuracy:  0.542
hs: 500 lr: 0.40059999999999996 bs 200 reg 0.5
Validation accuracy:  0.107
hs: 500 lr: 0.40059999999999996 bs 200 reg 1.0
Validation accuracy:  0.107
hs: 500 lr: 0.40059999999999996 bs 300 reg 0.0
Validation accuracy:  0.532
hs: 500 lr: 0.40059999999999996 bs 300 reg 0.5
Validation accuracy:  0.105
hs: 500 lr: 0.40059999999999996 bs 300 reg 1.0
Validation accuracy:  0.105
hs: 500 lr: 0.40059999999999996 bs 400 reg 0.0
Validation accuracy:  0.539
hs: 500 lr: 0.40059999999999996 bs 400 reg 0.5
Validation accuracy:  0.098
hs: 500 lr: 0.40059999999999996 bs 400 reg 1.0
Validation accuracy:  0.079
hs: 500 lr: 0.20079999999999998 bs 200 reg 0.0
Validation accuracy:  0.509
hs: 500 lr: 0.20079999999999998 bs 200 reg 0.5
Validation accuracy:  0.107
hs: 500 lr: 0.20079999999999998 bs 200 reg 1.0
Validation accuracy:  0.112
hs: 500 lr: 0.20079999999999998 bs 300 reg 0.0
Validation accuracy:  0.509
hs: 500 lr: 0.20079999999999998 bs 300 reg 0.5
Validation accuracy:  0.107
hs: 500 lr: 0.20079999999999998 bs 300 reg 1.0
Validation accuracy:  0.105
hs: 500 lr: 0.20079999999999998 bs 400 reg 0.0
Validation accuracy:  0.516
hs: 500 lr: 0.20079999999999998 bs 400 reg 0.5
Validation accuracy:  0.102
hs: 500 lr: 0.20079999999999998 bs 400 reg 1.0
Validation accuracy:  0.078
hs: 500 lr: 0.001 bs 200 reg 0.0
Validation accuracy:  0.113
hs: 500 lr: 0.001 bs 200 reg 0.5
Validation accuracy:  0.078
hs: 500 lr: 0.001 bs 200 reg 1.0
Validation accuracy:  0.087
hs: 500 lr: 0.001 bs 300 reg 0.0
Validation accuracy:  0.078
hs: 500 lr: 0.001 bs 300 reg 0.5
Validation accuracy:  0.102
hs: 500 lr: 0.001 bs 300 reg 1.0
Validation accuracy:  0.078
hs: 500 lr: 0.001 bs 400 reg 0.0
Validation accuracy:  0.087
hs: 500 lr: 0.001 bs 400 reg 0.5
Validation accuracy:  0.079
hs: 500 lr: 0.001 bs 400 reg 1.0
Validation accuracy:  0.078
hs: 600 lr: 1.0 bs 200 reg 0.0
Validation accuracy:  0.532
hs: 600 lr: 1.0 bs 200 reg 0.5
Validation accuracy:  0.107
hs: 600 lr: 1.0 bs 200 reg 1.0
Validation accuracy:  0.079
hs: 600 lr: 1.0 bs 300 reg 0.0
Validation accuracy:  0.569
hs: 600 lr: 1.0 bs 300 reg 0.5
Validation accuracy:  0.079
Validation accuracy:  0.079
hs: 600 lr: 1.0 bs 300 reg 1.0
Validation accuracy:  0.105
hs: 600 lr: 1.0 bs 400 reg 0.0
Validation accuracy:  0.59
hs: 600 lr: 1.0 bs 400 reg 0.5
Validation accuracy:  0.078
hs: 600 lr: 1.0 bs 400 reg 1.0
Validation accuracy:  0.113
hs: 600 lr: 0.8002 bs 200 reg 0.0
Validation accuracy:  0.553
hs: 600 lr: 0.8002 bs 200 reg 0.5
Validation accuracy:  0.107
hs: 600 lr: 0.8002 bs 200 reg 1.0
Validation accuracy:  0.087
hs: 600 lr: 0.8002 bs 300 reg 0.0
Validation accuracy:  0.554
hs: 600 lr: 0.8002 bs 300 reg 0.5
Validation accuracy:  0.105
hs: 600 lr: 0.8002 bs 300 reg 1.0
Validation accuracy:  0.078
hs: 600 lr: 0.8002 bs 400 reg 0.0
Validation accuracy:  0.532
hs: 600 lr: 0.8002 bs 400 reg 0.5
Validation accuracy:  0.087
hs: 600 lr: 0.8002 bs 400 reg 1.0
Validation accuracy:  0.113
hs: 600 lr: 0.6004 bs 200 reg 0.0
Validation accuracy:  0.539
hs: 600 lr: 0.6004 bs 200 reg 0.5
Validation accuracy:  0.105
hs: 600 lr: 0.6004 bs 200 reg 1.0
Validation accuracy:  0.078
hs: 600 lr: 0.6004 bs 300 reg 0.0
Validation accuracy:  0.573
hs: 600 lr: 0.6004 bs 300 reg 0.5
Validation accuracy:  0.078
hs: 600 lr: 0.6004 bs 300 reg 1.0
Validation accuracy:  0.112
hs: 600 lr: 0.6004 bs 400 reg 0.0
Validation accuracy:  0.567
hs: 600 lr: 0.6004 bs 400 reg 0.5
Validation accuracy:  0.078
hs: 600 lr: 0.6004 bs 400 reg 1.0
Validation accuracy:  0.112
hs: 600 lr: 0.40059999999999996 bs 200 reg 0.0
Validation accuracy:  0.548
hs: 600 lr: 0.40059999999999996 bs 200 reg 0.5
Validation accuracy:  0.107
hs: 600 lr: 0.40059999999999996 bs 200 reg 1.0
Validation accuracy:  0.087
hs: 600 lr: 0.40059999999999996 bs 300 reg 0.0
Validation accuracy:  0.539
hs: 600 lr: 0.40059999999999996 bs 300 reg 0.5
Validation accuracy:  0.087
hs: 600 lr: 0.40059999999999996 bs 300 reg 1.0
Validation accuracy:  0.112
hs: 600 lr: 0.40059999999999996 bs 400 reg 0.0
Validation accuracy:  0.541
hs: 600 lr: 0.40059999999999996 bs 400 reg 0.5
Validation accuracy:  0.105
hs: 600 lr: 0.40059999999999996 bs 400 reg 1.0
Validation accuracy:  0.113
hs: 600 lr: 0.20079999999999998 bs 200 reg 0.0
Validation accuracy:  0.518
hs: 600 lr: 0.20079999999999998 bs 200 reg 0.5
Validation accuracy:  0.087
hs: 600 lr: 0.20079999999999998 bs 200 reg 1.0
Validation accuracy:  0.078
hs: 600 lr: 0.20079999999999998 bs 300 reg 0.0
Validation accuracy:  0.519
hs: 600 lr: 0.20079999999999998 bs 300 reg 0.5
Validation accuracy:  0.119
hs: 600 lr: 0.20079999999999998 bs 300 reg 1.0
Validation accuracy:  0.112
hs: 600 lr: 0.20079999999999998 bs 400 reg 0.0
Validation accuracy:  0.497
```

```
hs: 600 lr: 0.20079999999999998 bs 400 reg 0.5
Validation accuracy:  0.087
hs: 600 lr: 0.20079999999999998 bs 400 reg 1.0
Validation accuracy:  0.087
hs: 600 lr: 0.001 bs 200 reg 0.0
Validation accuracy:  0.102
hs: 600 lr: 0.001 bs 200 reg 0.5
Validation accuracy:  0.105
hs: 600 lr: 0.001 bs 200 reg 1.0
Validation accuracy:  0.087
hs: 600 lr: 0.001 bs 300 reg 0.0
Validation accuracy:  0.078
hs: 600 lr: 0.001 bs 300 reg 0.5
Validation accuracy:  0.087
hs: 600 lr: 0.001 bs 300 reg 1.0
Validation accuracy:  0.107
hs: 600 lr: 0.001 bs 400 reg 0.0
Validation accuracy:  0.112
hs: 600 lr: 0.001 bs 400 reg 0.5
Validation accuracy:  0.112
hs: 600 lr: 0.001 bs 400 reg 1.0
Validation accuracy:  0.112
hs: 700 lr: 1.0 bs 200 reg 0.0
Validation accuracy:  0.553
hs: 700 lr: 1.0 bs 200 reg 0.5
Validation accuracy:  0.107
hs: 700 lr: 1.0 bs 200 reg 1.0
Validation accuracy:  0.112
hs: 700 lr: 1.0 bs 300 reg 0.0
Validation accuracy:  0.537
hs: 700 lr: 1.0 bs 300 reg 0.5
Validation accuracy:  0.098
hs: 700 lr: 1.0 bs 300 reg 1.0
Validation accuracy:  0.098
hs: 700 lr: 1.0 bs 400 reg 0.0
Validation accuracy:  0.573
hs: 700 lr: 1.0 bs 400 reg 0.5
Validation accuracy:  0.087
hs: 700 lr: 1.0 bs 400 reg 1.0
Validation accuracy:  0.112
hs: 700 lr: 0.8002 bs 200 reg 0.0
Validation accuracy:  0.543
hs: 700 lr: 0.8002 bs 200 reg 0.5
Validation accuracy:  0.079
hs: 700 lr: 0.8002 bs 200 reg 1.0
Validation accuracy:  0.105
hs: 700 lr: 0.8002 bs 300 reg 0.0
Validation accuracy:  0.543
hs: 700 lr: 0.8002 bs 300 reg 0.5
Validation accuracy:  0.107
hs: 700 lr: 0.8002 bs 300 reg 1.0
Validation accuracy:  0.098
hs: 700 lr: 0.8002 bs 400 reg 0.0
Validation accuracy:  0.552
hs: 700 lr: 0.8002 bs 400 reg 0.5
Validation accuracy:  0.107
hs: 700 lr: 0.8002 bs 400 reg 1.0
Validation accuracy:  0.112
hs: 700 lr: 0.6004 bs 200 reg 0.0
Validation accuracy:  0.537
hs: 700 lr: 0.6004 bs 200 reg 0.5
Validation accuracy:  0.098
hs: 700 lr: 0.6004 bs 200 reg 1.0
Validation accuracy:  0.098
hs: 700 lr: 0.6004 bs 300 reg 0.0
Validation accuracy:  0.562
hs: 700 lr: 0.6004 bs 300 reg 0.5
Validation accuracy:  0.112
hs: 700 lr: 0.6004 bs 300 reg 1.0
Validation accuracy:  0.098
hs: 700 lr: 0.6004 bs 400 reg 0.0
Validation accuracy:  0.566
hs: 700 lr: 0.6004 bs 400 reg 0.5
Validation accuracy:  0.119
hs: 700 lr: 0.6004 bs 400 reg 1.0
Validation accuracy:  0.102
hs: 700 lr: 0.40059999999999996 bs 200 reg 0.0
```

```
Validation accuracy:  0.532
hs: 700 lr: 0.40059999999999996 bs 200 reg 0.5
Validation accuracy:  0.105
hs: 700 lr: 0.40059999999999996 bs 200 reg 1.0
Validation accuracy:  0.079
hs: 700 lr: 0.40059999999999996 bs 300 reg 0.0
Validation accuracy:  0.536
hs: 700 lr: 0.40059999999999996 bs 300 reg 0.5
Validation accuracy:  0.079
hs: 700 lr: 0.40059999999999996 bs 300 reg 1.0
Validation accuracy:  0.098
hs: 700 lr: 0.40059999999999996 bs 400 reg 0.0
Validation accuracy:  0.548
hs: 700 lr: 0.40059999999999996 bs 400 reg 0.5
Validation accuracy:  0.102
hs: 700 lr: 0.40059999999999996 bs 400 reg 1.0
Validation accuracy:  0.112
hs: 700 lr: 0.20079999999999998 bs 200 reg 0.0
Validation accuracy:  0.515
hs: 700 lr: 0.20079999999999998 bs 200 reg 0.5
Validation accuracy:  0.112
hs: 700 lr: 0.20079999999999998 bs 200 reg 1.0
Validation accuracy:  0.102
hs: 700 lr: 0.20079999999999998 bs 300 reg 0.0
Validation accuracy:  0.522
hs: 700 lr: 0.20079999999999998 bs 300 reg 0.5
Validation accuracy:  0.098
hs: 700 lr: 0.20079999999999998 bs 300 reg 1.0
Validation accuracy:  0.113
hs: 700 lr: 0.20079999999999998 bs 400 reg 0.0
Validation accuracy:  0.523
hs: 700 lr: 0.20079999999999998 bs 400 reg 0.5
Validation accuracy:  0.113
hs: 700 lr: 0.20079999999999998 bs 400 reg 1.0
Validation accuracy:  0.079
hs: 700 lr: 0.001 bs 200 reg 0.0
Validation accuracy:  0.078
hs: 700 lr: 0.001 bs 200 reg 0.5
Validation accuracy:  0.079
hs: 700 lr: 0.001 bs 200 reg 1.0
Validation accuracy:  0.112
hs: 700 lr: 0.001 bs 300 reg 0.0
Validation accuracy:  0.087
hs: 700 lr: 0.001 bs 300 reg 0.5
Validation accuracy:  0.078
hs: 700 lr: 0.001 bs 300 reg 1.0
Validation accuracy:  0.079
hs: 700 lr: 0.001 bs 400 reg 0.0
Validation accuracy:  0.078
hs: 700 lr: 0.001 bs 400 reg 0.5
Validation accuracy:  0.087
hs: 700 lr: 0.001 bs 400 reg 1.0
Validation accuracy:  0.079

** best validation achieved during cross validation:  0.59
```

In [11]:

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

# winning hyperparameters
print ("best hypeparameters ->  hidden ", best_hidd, ' lr ', best_lr, ' bs', best_batch_size, ' reg
', best_reg)
# add validation accuracy just for clarity for below inline question
val_acc = (best_net.predict(X_val_feats) == y_val).mean()
print ("Best net Validation accuracy :", val_acc)

test_acc = (best_net.predict(X_test_feats) == y_test).mean()
print("Test accuracy   ",test_acc)
```

```
best hypeparameters ->  hidden  600  lr  1.0  bs 400  reg   0.0
Best net Validation accuracy : 0.59
Test accuracy    0.578
```

In [ ]:

```

```