# 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-fa18.github.io/assignments2018/assignment1)](https://compsci682-fa18.github.io/assignments2018/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]:
```
import random
import numpy as np
from cs682.data_utils import load_CIFAR10
import matplotlib.pyplot as plt

from __future__ import print_function

%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 [3]:
```
from cs682.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=False)
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))])
```

## 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]:

```
# 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]
regularization_strengths = [5e4, 5e5, 5e6]

results = {}
best_val = -1
best_svm = None

for lr in learning_rates:
    for reg_str in regularization_strengths:
        print("Running" + str((lr,reg_str)))
        svm = LinearSVM()
        loss_hist = svm.train( X_train_feats,  y_train, learning_rate=lr, reg=reg_str, num_iters=1500, verbose=
False)
        y_valid_pred = svm.predict( X_val_feats)
        y_train_pred = svm.predict(X_train_feats)
        valid_accuracy =  (np.mean(y_val == y_valid_pred) )
        train_accuracy =  (np.mean(y_train == y_train_pred) )
        if valid_accuracy > best_val:
            best_val = valid_accuracy
            best_svm = svm

        results[(lr,reg_str)] = (train_accuracy, valid_accuracy)

# 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)
```

```
Running(1e-09, 50000.0)
Running(1e-09, 500000.0)
Running(1e-09, 5000000.0)
Running(1e-08, 50000.0)
Running(1e-08, 500000.0)
Running(1e-08, 5000000.0)
Running(1e-07, 50000.0)
Running(1e-07, 500000.0)
Running(1e-07, 5000000.0)
lr 1.000000e-09 reg 5.000000e+04 train accuracy: 0.099735 val accuracy: 0.094000
lr 1.000000e-09 reg 5.000000e+05 train accuracy: 0.123061 val accuracy: 0.124000
lr 1.000000e-09 reg 5.000000e+06 train accuracy: 0.126449 val accuracy: 0.111000
lr 1.000000e-08 reg 5.000000e+04 train accuracy: 0.097694 val accuracy: 0.093000
lr 1.000000e-08 reg 5.000000e+05 train accuracy: 0.362490 val accuracy: 0.373000
lr 1.000000e-08 reg 5.000000e+06 train accuracy: 0.402388 val accuracy: 0.392000
lr 1.000000e-07 reg 5.000000e+04 train accuracy: 0.412939 val accuracy: 0.409000
lr 1.000000e-07 reg 5.000000e+05 train accuracy: 0.417122 val accuracy: 0.428000
lr 1.000000e-07 reg 5.000000e+06 train accuracy: 0.350571 val accuracy: 0.340000
best validation accuracy achieved during cross-validation: 0.428000
```

In [6]:
```
# 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)
```

0.421

In [7]:
```
# 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?

They do make sense. For example, if you look at the mislabels for cat, a lot of the images are dogs. Since dogs look very close to cats (four leg animal) it make sense that the network would misclassify these.

# 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 [8]:

```
# 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 [9]:

```
from cs682.classifiers.neural_net import TwoLayerNet

input_size = X_train_feats.shape[1]
num_classes = 10


best_net = None
best_val = -1

#I tried all of these when experimenting to find the best parameters
learning_rates = [1e-1,5e-1,1e0,5e0]
regularization_strengths = [1e-4 , 1e-3, 0.0]
hidden_sizes = [100,250,500]
epochs = [2000]
dropouts = [0.5,0.8,1.0]

#I tried all of these when experimenting to find the best parameters
learning_rates = [5e-1]
regularization_strengths = [0.0]
hidden_sizes = [500]
epochs = [10000]
dropouts = [0.8]
use_dropout = True


for lr in learning_rates:
    for reg_str in regularization_strengths:
        for hidden_size in hidden_sizes:
            for epoch in epochs:
                for dr in dropouts:
                    print("Running" + str((lr,reg_str, hidden_size,epoch, dr)))
                    net = TwoLayerNet(input_size, hidden_size, num_classes)
                    # Train the network
                    stats = net.train(X_train_feats, y_train, X_val_feats, y_val, num_iters=epoch, batch_size=200,learning_rate=lr, learning_rate_decay=0.95, reg=reg_str, verbose=False, dropout_percent = dr, use_dropout = use_dropout)
                    y_valid_pred = net.predict( X_val_feats)
                    y_train_pred = net.predict(X_train_feats)
                    valid_accuracy =  (np.mean(y_val == y_valid_pred) )
                    train_accuracy =  (np.mean(y_train == y_train_pred) )
                    if valid_accuracy > best_val:
                        print("best accuracy so far: " +  str(valid_accuracy) )
                        best_val = valid_accuracy
                        best_net= net

                    results[(lr,reg_str)] = (train_accuracy, valid_accuracy)
```

```
Running(0.5, 0.0, 500, 10000, 0.8)
best accuracy so far: 0.593
```

In [10]:
```
# 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)
```

```
0.605
```