图像特征练习

补充并完成本练习。

我们已经看到，通过在输入图像的像素上训练线性分类器，从而在图像分类任务上达到一个合理的性能。在本练习中，我们将展示我们可以通过对线性分类器（不是在原始像素上，而是在根据原始像素计算出的特征上）进行训练来改善分类性能。

你将在此notebook中完成本练习的所有工作。

In [1]:
```python
import random
import numpy as np
from daseCV.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
```

数据加载

与之前的练习类似，我们将从磁盘加载CIFAR-10数据。

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

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

    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

X_train, y_train, X_val, y_val, X_test, y_test = get_CIFAR10_data()
```

特征提取

对于每一张图片我们将会计算它的方向梯度直方图（英語：Histogram of oriented gradient，简称HOG）以及在HSV颜色空间使用色相通道的颜色直方图。

简单来讲，HOG能提取图片的纹理信息而忽略颜色信息，颜色直方图则提取出颜色信息而忽略纹理信息。 因此，我们希望将两者结合使用而不是单独使用任一个。去实现这个设想是一个十分有趣的事情。

`hog_feature` 和 `color_histogram_hsv` 两个函数都可以对单个图像进行运算并返回改图像的一个特征向量。 extract_features函数输入一个图像集合和一个特征函数列表然后对每张图片运行每个特征函数， 然后将结果存储在一个矩阵中，矩阵的每一列是单个图像的所有特征向量的串联。

In [3]:
```
from daseCV.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))])
```

```
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
Done extracting features for 49000 / 49000 images
```

使用特征训练SVM

使用之前作业完成的多分类SVM代码来训练上面提取的特征。这应该比原始数据直接在SVM上训练会去的更好的效果。

In [4]:

```python
# 使用验证集调整学习率和正则化强度

from daseCV.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

################################################################################
# 你需要做的:
# 使用验证集设置学习率和正则化强度。
# 这应该与你对SVM所做的验证相同;
# 将训练最好的的分类器保存在best_svm中。
# 你可能还想在颜色直方图中使用不同数量的bins。
# 如果你细心一点应该能够在验证集上获得0.44以上的准确性。
################################################################################
# *****START OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****

for lr in learning_rates:
      for reg in regularization_strengths:
        net = LinearSVM()
```

```
            loss_history = net.train(X_train_feats, y_train)
            y_train_pred = net.predict(X_train_feats)
            y_val_pred = net.predict(X_val_feats)
            train_acc = np.sum(y_train_pred == y_train) / len(y_train)
            val_acc = np.sum(y_val_pred == y_val) / len(y_val)
            results[(lr, reg)] = train_acc, val_acc
            if val_acc > best_val:
                best_val = val_acc
                best_svm = net

# *****END OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****

# 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 1.000000e-09 reg 5.000000e+04 train accuracy: 0.439041 val accuracy: 0.441000
lr 1.000000e-09 reg 5.000000e+05 train accuracy: 0.436327 val accuracy: 0.425000
lr 1.000000e-09 reg 5.000000e+06 train accuracy: 0.440245 val accuracy: 0.438000
lr 1.000000e-08 reg 5.000000e+04 train accuracy: 0.436367 val accuracy: 0.438000
lr 1.000000e-08 reg 5.000000e+05 train accuracy: 0.433327 val accuracy: 0.440000
lr 1.000000e-08 reg 5.000000e+06 train accuracy: 0.441020 val accuracy: 0.437000
lr 1.000000e-07 reg 5.000000e+04 train accuracy: 0.438531 val accuracy: 0.437000
lr 1.000000e-07 reg 5.000000e+05 train accuracy: 0.434102 val accuracy: 0.438000
lr 1.000000e-07 reg 5.000000e+06 train accuracy: 0.438020 val accuracy: 0.434000
best validation accuracy achieved during cross-validation: 0.441000
```

In [5]:

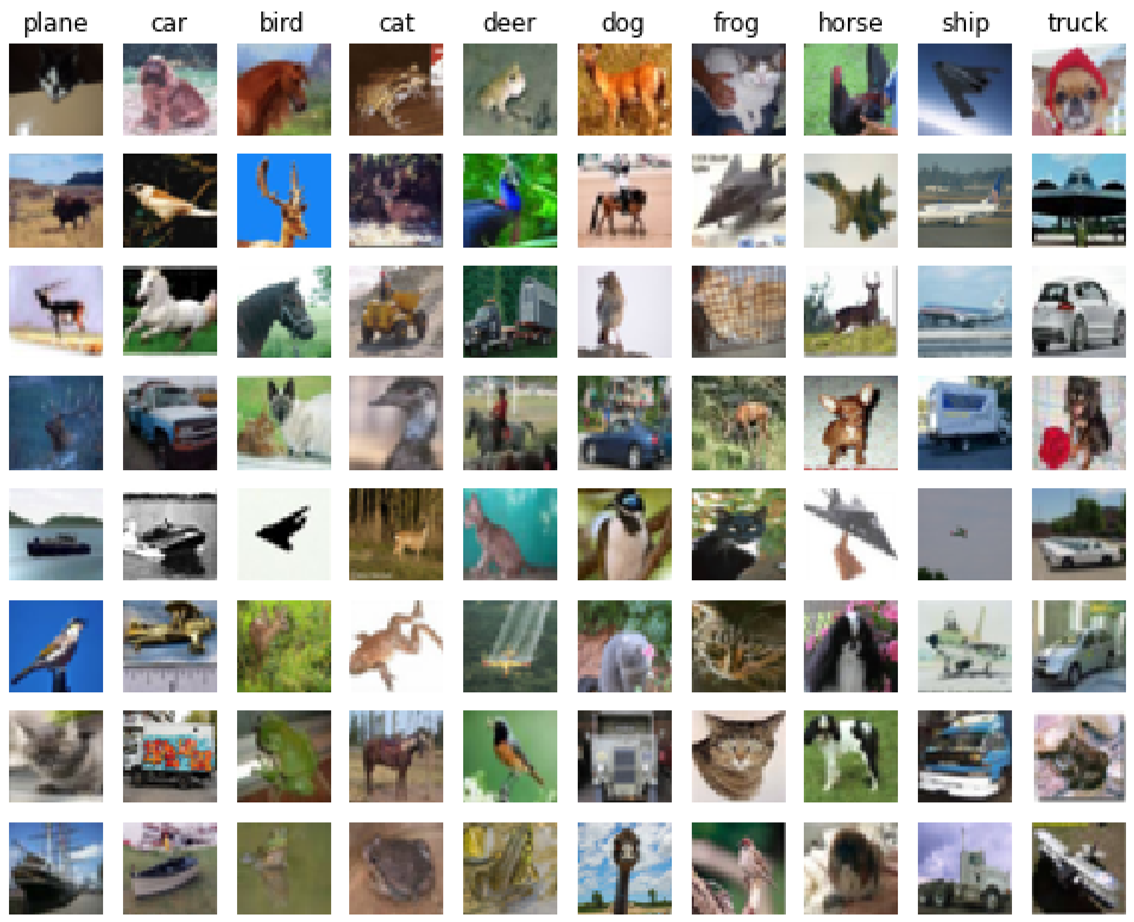
```
# Evaluate your trained SVM on the test set
y_test_pred = best_svm.predict(X_test_feats)
test_accuracy = np.mean(y_test == y_test_pred)
print(test_accuracy)
```

```
0.443
```

In [6]:

```
# 直观了解算法工作原理的一种重要方法是可视化它所犯的错误。
# 在此可视化中，我们显示了当前系统未正确分类的图像示例。
# 第一列显示的图像是我们的系统标记为“ plane”，但其真实标记不是“ plane”。

examples_per_class = 8
classes = ['plane', 'car', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'tru
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()
```



问题 1:

描述你看到的错误分类结果。你认为他们有道理吗?

答:

有一部分图片是合理的，虽然他们是不同的类别，但是之间存在着相似的特征，比如飞机和鸟，都有“翅膀”这一特征，因此这几个彼此间的错误分类是比较合理的。但也存在一些无道理的即不存在上述问题却还是被分在了一起。

图像特征神经网络

在之前的练习中，我们看到在原始像素上训练两层神经网络比线性分类器具有更好的分类精度。在这里，我们已经看到使用图像特征的线性分类器优于使用原始像素的线性分类器。 为了完整起见，我们还应该尝试在图像特征上训练神经网络。这种方法应优于以前所有的方法：你应该能够轻松地在测试集上达到55%以上的分类精度；我们最好的模型可达到约60%的精度。

In [7]:
```
# 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 [8]:
```
from daseCV.classifiers.neural_net import TwoLayerNet

input_dim = X_train_feats.shape[1]
hidden_dim = 500
num_classes = 10
best_acc = 0.0

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

################################################################################
# TODO: 使用图像特征训练两层神经网络。
# 您可能希望像上一节中那样对各种参数进行交叉验证。
# 将最佳的模型存储在best_net变量中。
################################################################################
# *****START OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****

learning_rates = np.random.choice(range(1,100),2,replace=False) * 1e-2
regularization_strengths = np.random.choice(range(1,100),2,replace=False) * 1e-6

results = {}
best_val = -1
for lr in learning_rates:
      for reg in regularization_strengths:
        net.__init__(input_dim, hidden_dim, num_classes)
        net.train(X_train_feats, y_train, X_val_feats, y_val, verbose=True,
             learning_rate=lr, reg=reg, num_iters=2000, batch_size=400)
        y_train_pred = net.predict(X_train_feats)
        y_val_pred = net.predict(X_val_feats)
        train_acc = (y_train_pred == y_train).mean()
        val_acc = (y_val_pred == y_val).mean()
        results[(lr, reg)] = train_acc, val_acc
        if val_acc > best_val:
            best_val = val_acc
            best_net = net

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)

# *****END OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****
```

```
iteration 0 / 2000: loss 2.302585
iteration 100 / 2000: loss 1.532730
iteration 200 / 2000: loss 1.339194
iteration 300 / 2000: loss 1.342787
iteration 400 / 2000: loss 1.268524
iteration 500 / 2000: loss 1.170777
iteration 600 / 2000: loss 1.133140
iteration 700 / 2000: loss 1.152869
iteration 800 / 2000: loss 1.118325
iteration 900 / 2000: loss 1.121230
iteration 1000 / 2000: loss 1.084236
iteration 1100 / 2000: loss 0.989279
iteration 1200 / 2000: loss 0.950257
iteration 1300 / 2000: loss 0.892208
iteration 1400 / 2000: loss 0.935663
iteration 1500 / 2000: loss 0.982601
iteration 1600 / 2000: loss 0.893773
iteration 1700 / 2000: loss 0.826656
iteration 1800 / 2000: loss 0.745611
iteration 1900 / 2000: loss 0.845612
iteration 0 / 2000: loss 2.302585
iteration 100 / 2000: loss 1.514170
iteration 200 / 2000: loss 1.473088
iteration 300 / 2000: loss 1.248063
iteration 400 / 2000: loss 1.213919
iteration 500 / 2000: loss 1.199444
iteration 600 / 2000: loss 1.158884
iteration 700 / 2000: loss 1.070348
iteration 800 / 2000: loss 1.123213
iteration 900 / 2000: loss 1.043740
iteration 1000 / 2000: loss 1.018279
iteration 1100 / 2000: loss 0.987670
iteration 1200 / 2000: loss 1.049354
iteration 1300 / 2000: loss 0.921886
iteration 1400 / 2000: loss 0.949525
iteration 1500 / 2000: loss 0.893252
iteration 1600 / 2000: loss 0.834292
iteration 1700 / 2000: loss 0.757300
iteration 1800 / 2000: loss 0.755015
iteration 1900 / 2000: loss 0.867543
iteration 0 / 2000: loss 2.302585
iteration 100 / 2000: loss 1.492405
iteration 200 / 2000: loss 1.290956
iteration 300 / 2000: loss 1.160124
iteration 400 / 2000: loss 1.052865
iteration 500 / 2000: loss 1.131236
iteration 600 / 2000: loss 0.992797
iteration 700 / 2000: loss 0.973281
iteration 800 / 2000: loss 0.920474
iteration 900 / 2000: loss 0.898211
iteration 1000 / 2000: loss 0.837771
iteration 1100 / 2000: loss 0.833077
iteration 1200 / 2000: loss 0.848556
iteration 1300 / 2000: loss 0.743020
iteration 1400 / 2000: loss 0.833879
iteration 1500 / 2000: loss 0.704953
iteration 1600 / 2000: loss 0.582575
iteration 1700 / 2000: loss 0.648608
iteration 1800 / 2000: loss 0.523335
iteration 1900 / 2000: loss 0.551944
iteration 0 / 2000: loss 2.302585
iteration 100 / 2000: loss 1.432685
iteration 200 / 2000: loss 1.313327
iteration 300 / 2000: loss 1.152146
iteration 400 / 2000: loss 1.091026
iteration 500 / 2000: loss 0.994852
iteration 600 / 2000: loss 1.035289
iteration 700 / 2000: loss 1.094513
iteration 800 / 2000: loss 0.989558
iteration 900 / 2000: loss 0.919298
iteration 1000 / 2000: loss 0.869840
iteration 1100 / 2000: loss 0.815571
iteration 1200 / 2000: loss 0.779088
iteration 1300 / 2000: loss 0.779466
iteration 1400 / 2000: loss 0.767195
iteration 1500 / 2000: loss 0.653306
iteration 1600 / 2000: loss 0.652198
iteration 1700 / 2000: loss 0.686021
iteration 1800 / 2000: loss 0.575389
iteration 1900 / 2000: loss 0.629303
lr 4.200000e-01 reg 2.100000e-05 train accuracy: 0.740612 val accuracy: 0.590000
lr 4.200000e-01 reg 5.800000e-05 train accuracy: 0.737490 val accuracy: 0.602000
lr 9.000000e-01 reg 2.100000e-05 train accuracy: 0.816184 val accuracy: 0.581000
lr 9.000000e-01 reg 5.800000e-05 train accuracy: 0.821755 val accuracy: 0.594000
best validation accuracy achieved during cross-validation: 0.602000
```

In [9]:

```
# 在测试集上运行得到的最好的神经网络分类器，应该能够获得55%以上的准确性。
```

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

```
0.553
```

In [ ]: