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.

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 [11]:

```
# 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 classifier 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.       #
################################################################################
# *****START OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****

for rate in learning_rates:
    for strength in regularization_strengths:
        svm = LinearSVM()

        svm.train(X_train_feats, y_train, rate, strength, num_iters=1000)

        y_train_pred = svm.predict(X_train_feats)
        training_accuracy = np.mean(y_train == y_train_pred)

        y_val_pred = svm.predict(X_val_feats)
        validation_accuracy = np.mean(y_val == y_val_pred)

        # Store every result
        results[(rate, strength)] = (training_accuracy, validation_accuracy)

        # Store the best validation accuracy as the best_val
        if validation_accuracy > best_val:
            best_val = validation_accuracy
            best_svm = svm

# *****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: 0.102612 val accuracy: 0.105000
lr 1.000000e-09 reg 5.000000e+04 train accuracy: 0.102612 val accuracy: 0.105000
lr 1.000000e-09 reg 5.000000e+05 train accuracy: 0.094776 val accuracy: 0.097000
lr 1.000000e-09 reg 5.000000e+06 train accuracy: 0.312796 val accuracy: 0.298000
lr 1.000000e-08 reg 5.000000e+04 train accuracy: 0.119388 val accuracy: 0.106000
lr 1.000000e-08 reg 5.000000e+05 train accuracy: 0.415592 val accuracy: 0.420000
lr 1.000000e-08 reg 5.000000e+06 train accuracy: 0.411531 val accuracy: 0.395000
lr 1.000000e-07 reg 5.000000e+04 train accuracy: 0.415265 val accuracy: 0.425000
lr 1.000000e-07 reg 5.000000e+05 train accuracy: 0.404633 val accuracy: 0.406000
lr 1.000000e-07 reg 5.000000e+06 train accuracy: 0.302163 val accuracy: 0.333000
best validation accuracy achieved during cross-validation: 0.425000
```

In [12]:

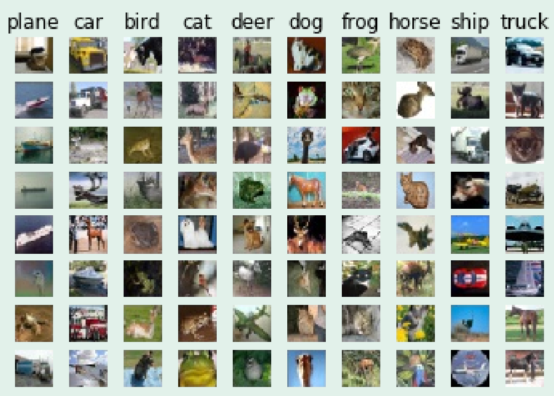
```
# Evaluate your trained SVM on the test set
y_test_pred = best_svm.predict(X_test_feats)
test_accuracy = np.mean(y_test == y_test_pred)
print(test_accuracy)
```

```
0.419
```

In [13]:

```
# 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?

YourAnswer: The misclassification results look similar in shape or background color. For example the first column is classified as plane, their shapes are like a straight stick and their background colors are mostly blue. This make sense because we capture the textures and colors of the pictures.

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 [21]:

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

learning_rates = [1e-2, 1e-1, 1]
regularization_strengths = [1e-3, 1e-2, 1e-1, 1]

results = {}
best_val = -1

for rate in learning_rates:
    for strength in regularization_strengths:
        net = TwoLayerNet(input_dim, hidden_dim, num_classes)

        net.train(X_train_feats, y_train, rate, strength, num_iters=1000)

        stats = net.train(X_train_feats, y_train, X_val_feats, y_val,
                          num_iters=1000, batch_size=200,
                          learning_rate=rate, learning_rate_decay=0.95,
                          reg= strength, verbose=True)

        y_train_pred = net.predict(X_train_feats)
        training_accuracy = np.mean(y_train == y_train_pred)

        y_val_pred = net.predict(X_val_feats)
        validation_accuracy = np.mean(y_val == y_val_pred)

        # Store every result
        results[(rate, strength)] = (training_accuracy, validation_accuracy)

        # Print out results.
        print('lr %e reg %e train accuracy: %f val accuracy: %f' % (rate, strength, train_accuracy,
val_accuracy))

        # Store the best validation accuracy as the best_val
        if validation_accuracy > best_val:
            best_val = validation_accuracy
            best_net = net



print ('best validation accuracy achieved is: %f' % best_val)

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

```
iteration 0 / 1000: loss 2.302570
iteration 100 / 1000: loss 2.302482
iteration 200 / 1000: loss 2.302697
iteration 300 / 1000: loss 2.302547
iteration 400 / 1000: loss 2.302587
iteration 500 / 1000: loss 2.302597
iteration 600 / 1000: loss 2.302416
iteration 700 / 1000: loss 2.302114
iteration 800 / 1000: loss 2.302466
iteration 900 / 1000: loss 2.302180
lr 1.000000e-02 reg 1.000000e-03 train accuracy: 0.100265 val accuracy: 0.087000
iteration 0 / 1000: loss 2.302624
iteration 100 / 1000: loss 2.302628
iteration 200 / 1000: loss 2.302626
iteration 300 / 1000: loss 2.302599
iteration 400 / 1000: loss 2.302298
iteration 500 / 1000: loss 2.302437
iteration 600 / 1000: loss 2.302588
iteration 700 / 1000: loss 2.301851
iteration 800 / 1000: loss 2.302178
iteration 900 / 1000: loss 2.302359
lr 1.000000e-02 reg 1.000000e-02 train accuracy: 0.100265 val accuracy: 0.087000
iteration 0 / 1000: loss 2.302670
iteration 100 / 1000: loss 2.302473
iteration 200 / 1000: loss 2.302705
iteration 300 / 1000: loss 2.302314
iteration 400 / 1000: loss 2.302871
iteration 500 / 1000: loss 2.302476
iteration 600 / 1000: loss 2.302393
iteration 700 / 1000: loss 2.302601
iteration 800 / 1000: loss 2.302751
iteration 900 / 1000: loss 2.303035
lr 1.000000e-02 reg 1.000000e-01 train accuracy: 0.100265 val accuracy: 0.087000
iteration 0 / 1000: loss 2.303399
iteration 100 / 1000: loss 2.302626
iteration 200 / 1000: loss 2.302598
iteration 300 / 1000: loss 2.302792
iteration 400 / 1000: loss 2.302290
iteration 500 / 1000: loss 2.302472
iteration 600 / 1000: loss 2.302354
iteration 700 / 1000: loss 2.302781
iteration 800 / 1000: loss 2.302649
iteration 900 / 1000: loss 2.302364
lr 1.000000e-02 reg 1.000000e+00 train accuracy: 0.100265 val accuracy: 0.087000
iteration 0 / 1000: loss 2.302591
iteration 100 / 1000: loss 2.301658
iteration 200 / 1000: loss 2.077883
iteration 300 / 1000: loss 1.808689
iteration 400 / 1000: loss 1.642119
iteration 500 / 1000: loss 1.607456
iteration 600 / 1000: loss 1.488302
iteration 700 / 1000: loss 1.468350
iteration 800 / 1000: loss 1.374784
iteration 900 / 1000: loss 1.462700
lr 1.000000e-01 reg 1.000000e-03 train accuracy: 0.100265 val accuracy: 0.087000
iteration 0 / 1000: loss 2.302580
iteration 100 / 1000: loss 2.303193
iteration 200 / 1000: loss 2.162910
iteration 300 / 1000: loss 1.944073
iteration 400 / 1000: loss 1.788377
iteration 500 / 1000: loss 1.633414
iteration 600 / 1000: loss 1.644684
iteration 700 / 1000: loss 1.763350
iteration 800 / 1000: loss 1.609069
iteration 900 / 1000: loss 1.630226
lr 1.000000e-01 reg 1.000000e-02 train accuracy: 0.100265 val accuracy: 0.087000
iteration 0 / 1000: loss 2.302674
iteration 100 / 1000: loss 2.303649
iteration 200 / 1000: loss 2.301822
iteration 300 / 1000: loss 2.293622
iteration 400 / 1000: loss 2.249799
iteration 500 / 1000: loss 2.262788
iteration 600 / 1000: loss 2.159708
iteration 700 / 1000: loss 2.196876
iteration 800 / 1000: loss 2.201736
iteration 900 / 1000: loss 2.195844
lr 1.000000e-01 reg 1.000000e-01 train accuracy: 0.100265 val accuracy: 0.087000
iteration 0 / 1000: loss 2.303426
iteration 100 / 1000: loss 2.302915
iteration 200 / 1000: loss 2.303354
iteration 300 / 1000: loss 2.302222
iteration 400 / 1000: loss 2.302448
iteration 500 / 1000: loss 2.302909
iteration 600 / 1000: loss 2.302180
iteration 700 / 1000: loss 2.303045
iteration 800 / 1000: loss 2.302386
iteration 900 / 1000: loss 2.303590
lr 1.000000e-01 reg 1.000000e+00 train accuracy: 0.100265 val accuracy: 0.087000
iteration 0 / 1000: loss 2.302629
iteration 100 / 1000: loss 1.514732
iteration 200 / 1000: loss 1.166685
iteration 300 / 1000: loss 1.349503
iteration 400 / 1000: loss 1.486098
iteration 500 / 1000: loss 1.252753
iteration 600 / 1000: loss 1.268897
iteration 700 / 1000: loss 1.286091
iteration 800 / 1000: loss 1.286079
iteration 900 / 1000: loss 1.233952
lr 1.000000e+00 reg 1.000000e-03 train accuracy: 0.100265 val accuracy: 0.087000
iteration 0 / 1000: loss 2.302666
iteration 100 / 1000: loss 1.643877
iteration 200 / 1000: loss 1.632431
iteration 300 / 1000: loss 1.671285
iteration 400 / 1000: loss 1.634135
iteration 500 / 1000: loss 1.628061
iteration 600 / 1000: loss 1.724921
iteration 700 / 1000: loss 1.647692
iteration 800 / 1000: loss 1.632797
iteration 900 / 1000: loss 1.553428
lr 1.000000e+00 reg 1.000000e-02 train accuracy: 0.100265 val accuracy: 0.087000
iteration 0 / 1000: loss 2.302642
iteration 100 / 1000: loss 2.161184
iteration 200 / 1000: loss 2.220938
iteration 300 / 1000: loss 2.244840
iteration 400 / 1000: loss 2.216159
iteration 500 / 1000: loss 2.191677
iteration 600 / 1000: loss 2.157572
iteration 700 / 1000: loss 2.144860
iteration 800 / 1000: loss 2.207613
iteration 900 / 1000: loss 2.137819
lr 1.000000e+00 reg 1.000000e-01 train accuracy: 0.100265 val accuracy: 0.087000
iteration 0 / 1000: loss 2.303367
iteration 100 / 1000: loss inf
iteration 200 / 1000: loss inf
iteration 300 / 1000: loss inf
iteration 400 / 1000: loss inf
iteration 500 / 1000: loss inf
iteration 600 / 1000: loss inf
iteration 700 / 1000: loss inf
iteration 800 / 1000: loss inf
iteration 900 / 1000: loss nan
lr 1.000000e+00 reg 1.000000e+00 train accuracy: 0.100265 val accuracy: 0.087000
best validation accuracy achieved is: 0.577000
```

In [22]:

```python
# 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)
```

```
0.567
```