Computer Vision (LE48) – Exercise 2

Jan Ondras (jo356)
Trinity College

December 1, 2017

1 Feature extraction

Firstly, I looked at images from all four classes to see what discriminates them most. For the first feature type I decided to use *local features* expecting that they can well capture shapes such as regular structure of bridges or various shapes of trees in rainforests. As a second feature type I used *colour features* since the four classes of images seem to have different colour compositions which helps to discriminate between them. In general, the aim is to choose features that minimize within-class variability and maximize between-class variability.

In the following sections I will assume square images of size $a \times a$ pixels where $a = 256$.

1.1 Local features

I used the Histogram of Oriented Gradients (HOG) feature descriptors that are calculated as follows:

1. **Global image normalization (optional[1])**
   To lessen the effects of variations in illumination and shadowing, the power law compression (Gamma correction) can be applied as a preprocessing step.
2. **Gradient calculation**
   Gradient images g_x, g_y for x and y directions are calculated. In OpenCV this can be achieved by Sobel operator with kernel size 1. g_x captures vertical structures in the image whereas g_y horizontal ones. The gradient image highlights outlines and removes some non-essential information such as constant coloured background. Next, the magnitude and direction of a gradient is calculated (using OpenCV this can be done using the function `cartToPolar`).
3. **Computing gradient histograms**
   First, the image is divided into cells of $n_{pix} \times n_{pix}$ pixels. For each cell a histogram of gradients is calculated. This histogram is a vector of n_{bins} bins that correspond to evenly split angle ranges. A bin is selected based on the direction of the gradient and the contribution to the bin is determined by gradient magnitude.
4. **Block normalization**
   Histograms are normalized over bigger sized blocks to ensure better invariance to illumination, shadowing, and edge contrast. The overlap is controlled by the number n_{cell} such that there are $n_{cell} \times n_{cell}$ cells per block.
5. **Flattening into feature vector**
   The resulting flattened feature vector has length $(\frac{a}{n_{pix}} - n_{cell})^2 \times n_{cell}^2 \times n_{bins}$.

[1] see Section 3.1

To extract these features, I used the *scikit-image*[2] Python library, namely, the function

```
feature_descriptor = hog(img, orientations=n_bins, pixels_per_cell=(n_pix, n_pix),
                         cells_per_block=(n_cell, n_cell))
```

where I specified the values of n_{bins}, n_{pix}, and n_{cell}.

Usually, 9 orientation bins are chosen and the block size is set to twice the cell size. Thus, I set $n_{bins} = 9$ and $n_{cell} = 2$. For the block normalization I used L1-norm. The cell size should be chosen based on the scale of the features relevant for the classification. However, this is not known in advance and so I produced several feature sets by varying the value of n_{pix}. I labeled them as *hog_8*, *hog_16*, *hog_32*, *hog_64* for $n_{pix} = 8, 16, 32, 64$ respectively.

Fig. 1 shows HOG features extracted at various scales for sample images from the training set. We can see that these features can well capture the structure of the bridge and it seems that the values $n_{pix} = 16, 32$ are a good choice as they neither oversimplify nor overfit the image.

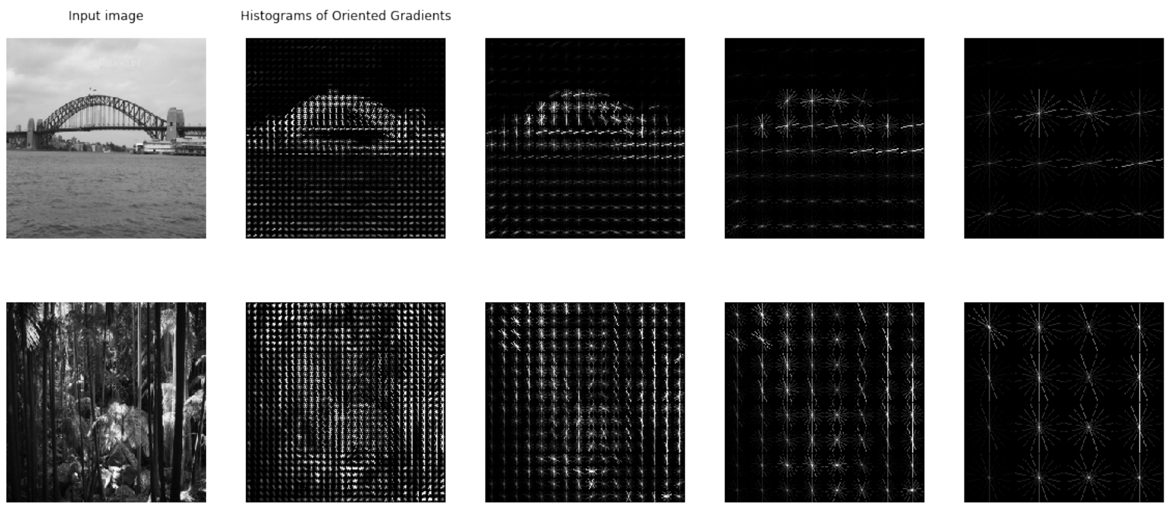

Figure 1: Extracted HOG features for $n_{bins} = 9$, $n_{cell} = 2$ and for various cell sizes $n_{pix} = 8, 16, 32, 64$ (from left to right). Classes: bridge (top row) and rainforest (bottom row).

1.2 Colour features

I started by visualizing the colour information for every image class and I plotted RGB histograms as shown in Fig. 2. As we can see, the information contained in colour channels has different character for different classes and similar one for same classes which suggests that it will be able to discriminate between them.

To create a feature vector of reasonable size I quantized each colour channel into n_{bins} bins. Usually 32–96 bins are used, but it tends to be application dependent. For this reason, I tried several values of $n_{bins} \in \{16, 32, 64\}$ and compared the classification results.

Furthermore, to capture local colour information I split image into several cells (each of size $n_{pix} \times n_{pix}$ pixels) and calculated the colour histograms for each. For each cell and for every colour channel the histogram was normalized. The final feature vector was then obtained by concatenating values of all histograms over all colour channels, hence its length was $3 \times (\frac{a}{n_{pix}})^2 \times n_{bins}$.

Similarly as in the case of local features, it was not clear how to set the value of n_{pix}. Therefore, I created several feature sets and labeled them as *col_*$\{n_{pix}\}$*_*$\{n_{bins}\}$ for $n_{pix} \in \{8, 16, 32\}$.

[2]http://scikit-image.org/

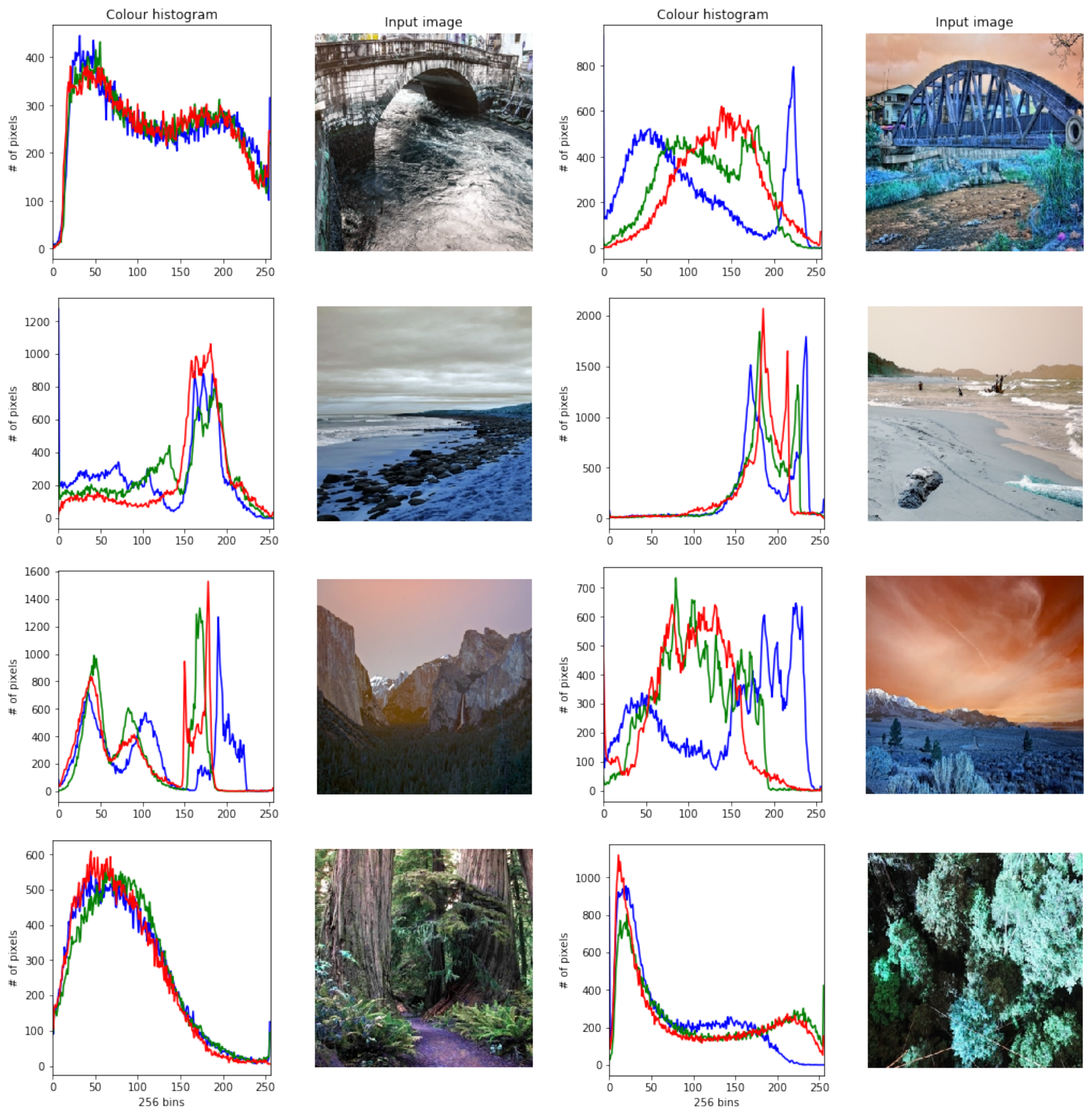

Figure 2: Colour histograms of RGB channels for the four image classes (from top): bridge, coast, mountain, and rainforest. The number of bins is $n_{bins} = 256$.

To extract these colour histograms I used the OpenCV method
`feature_descriptor = cv2.calcHist(img, channels, mask, histSize, ranges)`
where I specified which colour channels and what region of the image to use, as well as the number of bins (by histSize) and the range of possible pixel values (0–255).

1.3 Multimodal features

I created another feature set by combining local and colour feature sets. I used feature-level[3] fusion where the feature vectors of two modalities are simply concatenated resulting in a higher dimensional feature vector. I denoted this feature set by label *featureSet1Label + featureSet2Label*. To limit the

[3]also referred to as early fusion

number of possible multimodal feature sets I only combined feature sets that achieved best classification accuracies when used separately.

Additionally, I examined the effect of normalization and weighting of the two feature sets when concatenating them.

2 Classification

Once all the features were extracted for training as well as testing dataset, I used *scikit-learn*[4] Python library to train linear Support Vector Machine (SVM) classifier on the training dataset. Since there were multiple classes one-versus-the-rest[5] strategy was used. Linear SVM has only one hyperparameter: penalty C that I chose using 5-fold cross-validation, searching at logarithmic scale. I reported the results in terms of classification accuracy[6] on the testing set for all constructed feature sets in Tables 1 and 2, and in terms of confusion matrices for the best-performing feature sets in Figures 3 and 4.

2.1 Local features

Feature set	Classification accuracy %
hog_8	75.0
hog_16	76.4
hog_32	**79.3**
hog_64	77.9

Table 1: Classification accuracy on the testing set for various types of local HOG features. The number in the feature set name is the cell size in pixels (along one dimension).

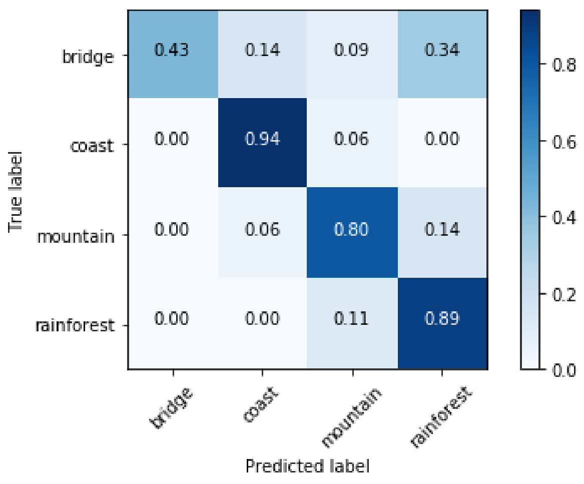

Figure 3: Normalized confusion matrix for the best-performing HOG feature set '*hog_32*'.

We can see that the cell size of 32 × 32 pixels for HOG features resulted in best performance. This implies that the features constructed at this scale can best discriminate these four classes. The confusion matrix presented in Fig. 3 shows the following:

[4]http://scikit-learn.org/stable/
[5]also known as one-versus-all
[6]accuracy was appropriate evaluation metric in this case because the dataset was balanced

- Coast images were classified very well. I did not expect that initially, but it is probably caused by very distinctive horizontal structures contained in these images.
- Rainforest and mountain images were also well classified but sometimes these two classes were mutually confused, which reflects the fact that images of mountains often contain forests.
- In contrast to my expectations, bridge images were not recognised very well and they were often confused with rainforests. This suggests that images of bridges and those of rainforests share similar local structure that was captured by HOG descriptors.

2.2 Colour features

n_{bins} \ n_{pix}	8	16	32
16	63.6	63.6	64.3
32	62.1	**65.0**	61.4
64	61.4	57.9	62.1

Table 2: Classification accuracy (%) on the testing set for various types of colour features determined by the cell size $n_{pix} \times n_{pix}$ and number n_{bins} of quantization bins.

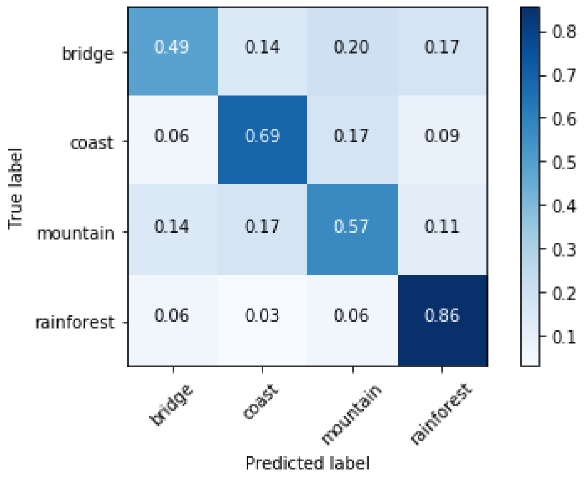

Figure 4: Normalized confusion matrix for the best-performing colour feature set '*col_16_32*'.

From the results for colour feature sets we can conclude that for this classification task the relevant number of bins is 32 and the most appropriate cell size is 16 × 16 pixels. This suggests that image regions of size 16 × 16 pixels provide most relevant colour information for classification in this case. As expected, colour features can discriminate rainforest images very well. The likely reason is that they contain very specific colour composition and different from other images.

Also, we can notice that colour features did not achieve the performance of local features. I think that the main reason is that colour information is influenced by background illumination and so it might not be always reliable. Nevertheless, the performance of all feature sets is well above the baseline, in this case a chance level of 25%.

2.3 Multimodal features

To clearly see the effect of multimodal features I summarized the performance of the best-performing local feature set, the best-performing colour feature set, and their combination in Tab. 3. We can see that a simple concatenation (without any normalization or weighting) of different types of features did not improve on the overall best result. The fact that the obtained accuracy is close to the accuracy of the colour feature set alone indicates that colour features are probably given too much weight. This is slightly mitigated by normalizing each feature set prior to the concatenation. However, the best solution is to use weighting so that the resulting multimodal feature vector f_m is given by

$$f_m = pf_h + (1-p)f_c$$

where f_h and f_c are normalized HOG and colour feature vectors respectively. The parameter $p \in [0,1]$ becomes another hyperparameter that needs to be tuned.

Feature set	Classification accuracy %
hog_32	79.3
col_16_32	65.0
hog_32 + col_16_32	67.9
hog_32 + col_16_32 (normalized)	73.6
hog_32 + col_16_32 (weighted)	74.3

Table 3: Classification accuracy (%) on the testing set for the best-performing local feature set, the best-performing colour feature set, and their combination: without normalization, with normalization, and with weighting using the optimized $p = 0.502$.

We can conclude that in this case the concatenation of different types of features resulted in worse performance when compared to the best performance obtained using a single modality. This was probably caused by the resulting feature vector having too high dimensionality compared to the relatively small number of training examples available.

3 Extensions

3.1 Image preprocessing

As suggested in the first section, when dealing with HOG features it might be beneficial to perform Gamma correction as a preprocessing step in order to reduce the effects of variations in illumination and shadowing.

To achieve this I computed the square root of each colour channel before applying the HOG algorithm. I used the cell size 32×32 pixels and labeled the new feature set as *hog_32_gamma*. Then, I performed training, cross-validation, and testing obtaining the classification accuracy 77.1% which did not improve on the original procedure without the correction.

3.2 Complex kernel SVMs

I also examined the SVM classifier with RBF kernel on both kinds of best-performing feature sets. In this case I cross-validated on two parameters: cost C and exponent γ. I obtained the accuracy 81.4% and 64.3% for HOG feature set *hog_32* and colour feature set *col_16_32* respectively. We can see that this more complex model (allowing non-linearities) achieved slightly better result for HOG feature set indicating that non-linear combinations of HOG features is relevant for the classification.

Appendix

Source code (extracted from Jupyter Notebook)

```python
import matplotlib.pyplot as plt
from skimage.feature import hog
from skimage import data, exposure
from skimage import io
from sklearn.preprocessing import normalize
from sklearn.metrics import confusion_matrix
from sklearn import svm, datasets, metrics
from sklearn.model_selection import GridSearchCV
import itertools
import numpy as np
import cv2
import glob

# Load dataset
classes = ['bridge', 'coast', 'mountain', 'rainforest'] # correspond to labels 0,1,2,3 in this order
train_images = {}
test_images = {}
for c in classes:
    train_images[c] = glob.glob('./../TrainingSet/' + c + '/*.jpg')
    test_images[c] = glob.glob('./../TestingSet/' + c + '/*.jpg')

# Image size (assume same for all images), 256 x 256
# len(cv2.imread(train_images['bridge'][0]))
width =  256
height = 256

# Number of examples per class (train, test) assuming balanced set
N_train = len(train_images['bridge'])
N_test = len(test_images['bridge'])
y_train = np.repeat([0,1,2,3], N_train) # Prepare labels
y_test = np.repeat([0,1,2,3], N_test)
X_train = {}
X_test = {}

###############################################################################################7
# Feature extraction
###############################################################################################
# Create feature vectors X_train, X_test in ascending order of class labels

###############################################################################################
# HOG features - extract
###############################################################################################
n_pix = 32
f_type = 'hog_' + str(n_pix) + '_gamma'
f_type = 'hog_' + str(n_pix)

def hog_features(img):
    return hog(img, orientations=9, pixels_per_cell=(n_pix, n_pix),
               cells_per_block=(2, 2), feature_vector=True)#, transform_sqrt=True)

X_train[f_type] = []
X_test[f_type] = []

# Over all 4 classes
for c in classes:
    # Over training images
    for img_name in train_images[c]:
        img = cv2.imread(img_name, 0)
        descriptor = hog_features(img)
        X_train[f_type].append(descriptor)

    # Over testing images
    for img_name in test_images[c]:
        img = cv2.imread(img_name, 0)
        descriptor = hog_features(img)
        X_test[f_type].append(descriptor)

X_train[f_type] = np.array(X_train[f_type])
X_test[f_type] = np.array(X_test[f_type])
```

```python
###############################################################################################
# Colour features - extract
###############################################################################################

def extract_colour_fd(img, n_pix, n_bins):
    n_cells = 256 / n_pix # cell size = n_pix x n_pix pixels
    fd = []
    for i in range(n_cells):
        for j in range(n_cells):
            for c in range(3): # For each colour channel: BGR
                hist = cv2.calcHist(img[i*n_pix:(i+1)*n_pix, j*n_pix:(j+1)*n_pix, c],
                    [c], None, [n_bins], [0, 256]).flatten()
                fd.extend(hist / np.sum(hist)) # Normalize the histogram
    return fd

def extract_colour_features(n_pix, n_bins, hist_eq=False):

    X_train_tmp = []
    X_test_tmp = []

    # Over all 4 classes
    for c in classes:
        # Over training images
        for img_name in train_images[c]:
            img = cv2.imread(img_name)
            descriptor = extract_colour_fd(img, n_pix, n_bins)
            X_train_tmp.append(descriptor)

        # Over testing images
        for img_name in test_images[c]:
            img = cv2.imread(img_name)
            descriptor = extract_colour_fd(img, n_pix, n_bins)
            X_test_tmp.append(descriptor)

    return np.array(X_train_tmp), np.array(X_test_tmp)

###############################################################################################
# Train, cross-validate and test using linear SVM classifier
###############################################################################################

def train_cv_test_linearSVM(X_train, X_test, verbose=False):
    # 5-fold cross-validation using grid search at logarithmic scale
    C_range = 2. ** np.arange(-11, 5, step=1.)
    parameters = [{'C': C_range}]
    grid = GridSearchCV(svm.LinearSVC(), parameters, cv=5, n_jobs=8)
    grid.fit(X_train, y_train)
    bestC = grid.best_params_['C']
    print "\tThe best parameter is: Cost=", np.log2(bestC)

    # Using the best hyperparameters, train on full training set
    SVM_linear_best = grid.best_estimator_
    SVM_linear_best.fit(X_train, y_train)

    # Evaluate
    y_pred = SVM_linear_best.predict(X_test)
    if verbose:
        print (metrics.classification_report(y_test, y_pred, target_names=classes))
    print "\tOverall Accuracy:", round(metrics.accuracy_score(y_test, y_pred), 3)
    return confusion_matrix(y_test, y_pred)

###############################################################################################
# Automatic feature extraction of colour features
# and classification for various feature extraction parameters
###############################################################################################

# n_pix_range =  [8, 16, 32]
# n_bins_range = [16, 32, 64]
n_pix_range =  [16]
n_bins_range = [32]
```

```python
for n_pix in n_pix_range:
    for n_bins in n_bins_range:
        f_type = 'col_'+ str(n_pix) +'_' + str(n_bins)
        print f_type
        X_train[f_type], X_test[f_type] = extract_colour_features(n_pix, n_bins)
        cm = train_cv_test_linearSVM(X_train[f_type], X_test[f_type])

###############################################################################
# Multimodal features (hog_32 + col_16_32)
###############################################################################

best_hog_fd = 'hog_32'
best_col_fd = 'col_16_32'
normalize = True

# Normalize each modality separately (L1-norm)
if normalize:
    X_train[best_hog_fd + '_norm'] = normalize(X_train[best_hog_fd], axis=1, norm='l1')
    X_test[best_hog_fd+ '_norm'] = normalize(X_test[best_hog_fd], axis=1, norm='l1')
    X_train[best_col_fd+ '_norm'] = normalize(X_train[best_col_fd], axis=1, norm='l1')
    X_test[best_col_fd+ '_norm'] = normalize(X_test[best_col_fd], axis=1, norm='l1')
    best_hog_fd = best_hog_fd + '_norm'
    best_col_fd = best_col_fd+ '_norm'
X_train['multimodal'] = np.append(X_train[best_hog_fd], X_train[best_col_fd], axis=1)
X_test['multimodal'] = np.append(X_test[best_hog_fd], X_test[best_col_fd], axis=1)
print "Multimodal features, combining ", best_hog_fd, " and ", best_col_fd
cm = train_cv_test_linearSVM(X_train['multimodal'], X_test['multimodal'], True)

# Tuning the weighting parameter p
p_range = np.linspace(0.4975, 0.506, 10)

best_hog_fd = 'hog_32_norm'
best_col_fd = 'col_16_32_norm'

for p in p_range:
    X_train['multimodal'] = np.append(p * X_train[best_hog_fd], (1.-p) * X_train[best_col_fd], axis=1)
    X_test['multimodal'] = np.append(p * X_test[best_hog_fd], (1.-p) * X_test[best_col_fd], axis=1)
    print "Multimodal features, combining ", best_hog_fd, " and ", best_col_fd
    print "\tp = ", p
    cm = train_cv_test_linearSVM(X_train['multimodal'], X_test['multimodal'])

###############################################################################
# Train, cross-validate and test using RBF kernel SVM classifier
###############################################################################

def train_cv_test_RBFSVM(X_train, X_test, verbose=False):
    # 5-fold cross-validation using grid search at logarithmic scale
    #     C_range = 2. ** np.arange(3, 4, step=0.15)
    #     g_range = 2. ** np.arange(-4, -3, step=0.15)
    C_range = 2. ** np.arange(0, 2, step=1.)
    g_range = 2. ** np.arange(-11, -7, step=1.)
    parameters = [{'gamma': g_range, 'C': C_range, 'kernel': ['rbf']}]
    grid = GridSearchCV(svm.SVC(), parameters, cv=5, n_jobs=8)
    grid.fit(X_train, y_train)
    bestC = grid.best_params_['C']
    bestG = grid.best_params_['gamma']
    print "\tThe best parameters are: gamma=", np.log2(bestG), "Cost=", np.log2(bestC)

    # Using the best hyperparameters, train on full training set
    SVM_rbf = grid.best_estimator_
    SVM_rbf.fit(X_train, y_train)

    # Evaluate
    y_pred = SVM_rbf.predict(X_test)
    if verbose:
        print (metrics.classification_report(y_test, y_pred, target_names=classes))
    print "\tOverall Accuracy:", round(metrics.accuracy_score(y_test, y_pred), 3)
    return confusion_matrix(y_test, y_pred)

cm = train_cv_test_RBFSVM(X_train['hog_32'], X_test['hog_32'])
cm = train_cv_test_RBFSVM(X_train['col_16_32'], X_test['col_16_32'])
```