Homework 2 Writeup

Objective of the work

The main goal of the work is to implement visual recognition of different environments based on feature descriptors. There are 15 labels of images and there are 100 images from each category in training, as well as test set.

Figure 1: Exemplary images from the dataset [1]

Process of performing visual recognition of depicted environments could be seen as a sequence of following processes:

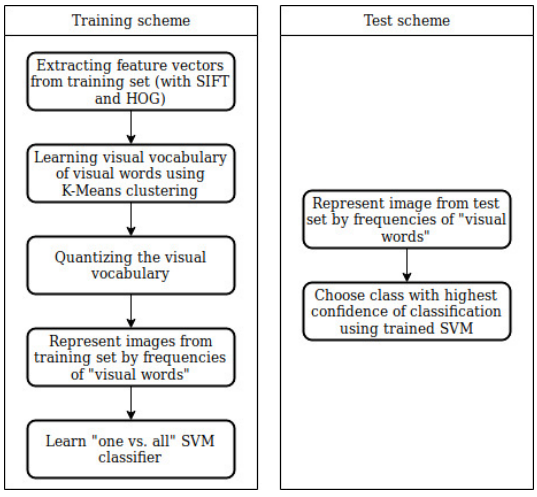


Figure 2: Flow chart of scene recognition scheme

Building vocabulary by k-means clustering

Feature extraction

In the exercise I used two feature extractors: Histogram of oriented gradients (HOG) and Scale-invariant feature transform (SIFT) algorithm.
HOG algorithm breaks image to cells, compute histogram of oriented gradients in the cells, normalize output across the block formed from cells and add up all gradients. I used `cv2.HOGDescriptor()` function, which returns concatenated list for all feature, which has to be re-scaled to size $(num\_features \times 36)$.
SIFT algorithm uses scale-space extrema detection with gaussian kernel ans keypoint localization to find relevant features. I used `cv2.xfeatures2d.SIFT_create()` which returns list of features of size $(num\_fatures \times 36)$. Code which performs feature extraction is shown below:

```
def feature_extraction(img, feature):
    """
    This function computes defined feature (HoG, SIFT) descriptors of the i

    :param img: a height x width x channels matrix,
    :param feature: name of image feature representation.

    :return: a N x feature_size matrix.
    """

    if feature == 'HoG':

        win_size = (32, 32)
        block_size = (32, 32)
        block_stride = (16, 16)
        cell_size = (16, 16)

        nbins = 9
        deriv_aperture = 1
        win_sigma = 4
        histogram_norm_type = 0
        l2_hys_threshold = 2.0000000000000001e-01
        gamma_correction = 0
        nlevels = 64

        # Your code here. You should also change the return value.

        # sample visualizing
        # cv2.imshow('img', img)

        gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

```
        hog = cv2.HOGDescriptor(win_size,
                                block_size,
                                block_stride,
                                cell_size,
                                nbins,
                                deriv_aperture,
                                win_sigma,
                                histogram_norm_type,
                                l2_hys_threshold,
                                gamma_correction,
                                nlevels)

        hist = hog.compute(gray)
        hist_resized = np.resize(hist, (int(len(hist)/36), 36))
        hist_resized
        return hist_resized

    elif feature == 'SIFT':

        gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        sift = cv2.xfeatures2d.SIFT_create()
        kp, des = sift.detectAndCompute(gray, None)

        return des
```

Learning visual vocabulary using K-means clustering

After feature extraction there are multiple features for every image. To build ”viusal vocabulary” from ”visual words”. ”Visual vocabulary” is a set of unique features derived from the training set. With the assumptions that there are regions where density of real

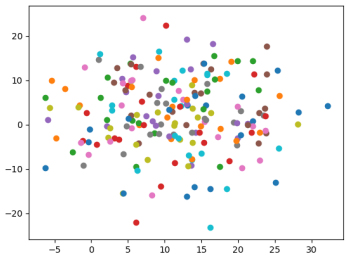
features are higher, we could find those regions by K-means clustering.

Data: Visual words
Result: List of code vectors
parameter : max_iter = 100, epsilon = 1e-4
Find maximum (max_feat) and minimum value (min_feat)
of each feature
Randomly assign centroids in the range max_feat, min_feat
while *iter* $<$ *max_iter* **do**
  assign points to nearest centroids;
  for *every centroid* c **do**
    if *there are any points assigned to centroid* **then**
      compute mean of assigned points $mean_c$;
      compute distance between mean of assigned points and centroid d_c;
      change position of c to mean: $c = mean_c$;
    else
      leave centroid in the same place;
    end
    if d_c *for every centroid* $< epsilon$ **then**
      return c
    else
  end
  return c
end

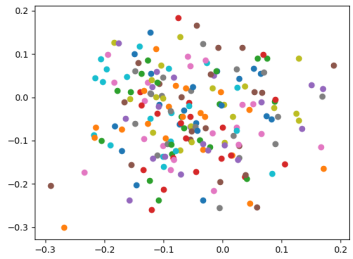
Algorithm 1: Building visual vocabulary with K-means

Principle component analysis(PCA) for vocabulary

Feature space for HOG algorithm equals 36 and for SIFT algorithm 128. Since it is impossible to visualize in a meaningful way such high dimensional data, we could use PCA for such task. PCA could be also perform as a preprocessing step to reduce the dimensionality of the data and to decrease the variance of our model.

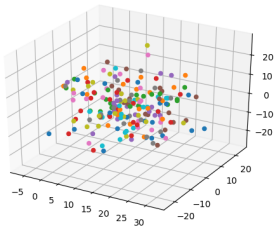


(a) First 2 PC for SIFT algorithm

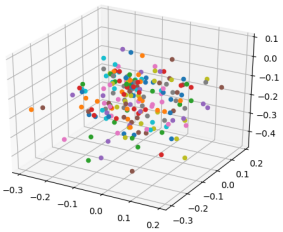


(b) First 2 PC for HOG algorithm

Figure 3: First 2 Principal Components for SIFT and HOG algorithm



(a) First 3 PC for SIFT algorithm



(b) First 3 PC for HOG algorithm

Figure 4: First 3 Principal Components for SIFT and HOG algorithm

Function which performs PC extraction is shown below:

```
def get_features_from_pca(feat_num, feature):
    """
    This function loads 'vocab_sift.npy' or 'vocab_hog.npg' file and
    returns dimension-reduced vocab into 2D or 3D.

    :param feat_num: 2 when we want 2D plot, 3 when we want 3D plot
    :param feature: 'Hog' or 'SIFT'

    :return: an N x feat_num matrix
    """

    if feature == 'HoG':
        vocab = np.load('vocab_hog.npy')
    elif feature == 'SIFT':
        vocab = np.load('vocab_sift.npy')

    def _get_PCA_vectors(feat_num, vocab):

        mean = vocab.mean(axis=0, keepdims=True)
        vocab_normalized = vocab - np.multiply(np.ones([vocab.shape[0],
```

```
            mean.shape[0]]), mean)
        #TEST: mean unit test
        #mean = vocab_normalized.mean(axis=0, keepdims=True)

        cov_matrix = np.cov(np.transpose(vocab_normalized))
        sigma, V = np.linalg.eig(cov_matrix)
        order_sigma = np.argsort(sigma)

        PCA_vectors = []
        i = 1
        for f in range(len(order_sigma)):
            eigen_vector = V[:, order_sigma[i]]
            if all(True for _ in np.isreal(eigen_vector)):
                PCA_vectors.append(np.real(eigen_vector))
                i += 1
            if len(PCA_vectors) == feat_num:
                break

        return np.array(PCA_vectors)

    #MAIN
    PCA_vectors = _get_PCA_vectors(feat_num, vocab)

    d = np.dot(vocab, np.transpose(PCA_vectors))

    return np.dot(vocab, np.transpose(PCA_vectors))
```

Representing images from training set by frequencies of ”visual words”

I implemented 2 approaches (Bag of words and Spatial Pyramid) to combine features obtain from every image into meaningful low-dimensional input into SVM classifier.

Bag of words representation of scenes

Bag of words implementation involve representing features of every image by code vectors of the ”visual vocabulary” and combining those codevectors into normalized histogram of occurrences. Bag of words implementation doesn't contain any information about spatial location and proximity between features.

Spatial pyramid representation

Spatial pyramid representation is a more complex form than Bag of words, which involves also information about spatial location of features obtaining for every image. In this approach histogram of features is obtained by weighted sum of histograms for multiple levels of images. Each level of images involve 4^l equal sized subimages of the original image.

Multi-class SVM

Having representation of images from training set by frequencies of "visual words" the next step is to learn SVM classifier which would be capable to recognize environments from [1]. SVM classifier is a binary classifier, so to obtain the classification task between 15 classes in the model I trained 15 "one vs. all". For every image there are 15 probabilities that the class is the choosen one. In the training set for every image is chosen class which has the highest probability obtained by "one vs. all" classifier for this class. Code which executes "one vs. all" classificator is shown below:

```
categories = np.unique(train_labels)

categories_dict = dict(zip(np.arange(15), categories))

all_predicted_proba = np.empty([1500,1])

for c in range(len(categories)):
    one_vs_all_labels = [1 if n == categories[c]
        else -1 for n in train_labels]
    clf = svm.SVC(probability=True, gamma='auto',
        kernel=kernel_type)
    clf.fit(train_image_feats, one_vs_all_labels)
    predicted_proba = clf.predict_proba(test_image_feats)
    predicted_proba = predicted_proba[:,1].
        reshape((len(predicted_proba[:,1]),1))
    all_predicted_proba = np.hstack((all_predicted_proba,
                                     predicted_proba))

all_predicted_proba = all_predicted_proba[:,1:]

max_proba = np.argmax(all_predicted_proba, axis=1)

out_labels = np.array([categories_dict[x]
    for x in max_proba])

return out_labels
```

The kernel trick

SVM classification could be obtain by solving Dual problem:

$$maxL_D(a_i) = \Sigma_{i=1}^{l} - \frac{1}{2}\Sigma_{i=1}^{l} a_i a_j y_i y_j (\mathbf{x_i} \cdot \mathbf{x_j}) s.t.$$

$\Sigma_{i=1}^{l} a_i y_i = 0$ & $a_i >= 0$ We obtain coefficients a by nonlinear optimization and in the training process to obtain the classification we only have to compute $(\mathbf{x_i} \cdot \mathbf{x_j})$, between all training examples for which $a_i = 0$ and new point (a_i is non-zero only for support vectors). This term could be wraped into to different kernel functions $K(\mathbf{x_i} \cdot \mathbf{x_j})$ to imitate feature expansion to get non-linear SVM decision boundary.

We could use radial basis function (RBF) kernel, which we get by computing:

$$K_{RBF}(\mathbf{x_i} \cdot \mathbf{x_j}) = e^{-\gamma||\mathbf{x_i} - \mathbf{x_j}||^2}$$

When parameter γ is higher we place lower impact on the difference between training points and new observation, so the "area of influence of support vectors is lower" and model is prone to overfitting.

Accuracy of the model

Comparsion of SIFT and HOG performance

In the model I used two feature exctractors SIFT and HOG.

Feature extractor	Kernel	Representation	Accuracy
SIFT	Linear	Bag of words	43.0%
HOG	Linear	Bag of words	41.9%
SIFT	Linear	Spatial pyramid	46.4%
HOG	Linear	Spatial pyramid	46.3%

Table 1: Comparsion of overall accuracy of the model between SIFT and HOG

As shown in table 1 SIFT features has 1.1% higher accuracy on the used dataset than HOG using Bag of words and 0.1% higher with use Spatial pyramid. The reason could be fact that, as mentioned above SIFT features include information about spatial location of the feature. Though, in the model especially with use of Spatial pyramid increase in the accuracy is not siginificant and depending on the task Bag of words could be better choice since computational complexity for this algorithm is significantly lower than for Spatial Pyramid.

Comparsion of linear and RBF kernel

In the model I used linear and RBF kernels SVM classifier.

Feature extractor	Kernel	Representation	Accuracy
SIFT	Linear	Bag of words	43.0%
SIFT	RBF	Bag of words	17.7%
HOG	Linear	Bag of words	41.9%
HOG	RBF	Bag of words	23.2%
SIFT	Linear	Spatial pyramid	46.4%
SIFT	RBF	Spatial pyramid	46.2%
HOG	Linear	Spatial pyramid	46.3%
HOG	RBF	Spatial pyramid	46.7%

Table 2: Comparsion of overall accuracy of the model between Linear and RBF kernel

Table 2 shows the comparsion between RBF and linear kernels. For the RBF kernel I used γ in the 'auto' mode which uses value $\frac{1}{number\_of\_features}$. Radial kernel has much higher variance than linear kernel and SVM classifier is much more unstable than in case of linear kernel. One could see it for the Bag of words approach, when for dafault values of γ mode has 23.3% and 18.7% lower accuracy!
With use of Bag of words RBF kernel has similar accuracy to linear kernel, with the best accuracy among all models which is 46.7%

Comparsion of Bag of Words and Spatial Pyramid

In the model I used Bag of Words and Spatial Pyramid representation.

Feature extractor	Kernel	Representation	Accuracy
SIFT	Linear	Spatial pyramid	46.4%
SIFT	Linear	Bag of words	43.0%
HOG	Linear	Spatial pyramid	46.3%
HOG	Linear	Bag of words	41.9%
SIFT	RBF	Spatial pyramid	46.2%
SIFT	RBF	Bag of words	17.7%
HOG	RBF	Spatial pyramid	46.7%
HOG	RBF	Bag of words	23.2%

Table 3: Comparsion of overall accuracy of the model between Bag of Words and Spatial Pyramid

Table 3 shows that on average Bag of Words has on average 2.4% lower accuracy using linear kernel than Spatial Pyramid and 26% lower using RBF kernel. It is a significant difference and in many more complex task, Spatial Pyramid could outperform Bag of Words.

Future works

In the further development of the model one could use Cross Validation scheme to try multiple kernels, and γ values. Also more training points could be used during the

training process.

References

[1] Svetlana Lazebnik, Cordelia Schmid, and Jean Ponce. “Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories”. In: *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR’06)*. Vol. 2. IEEE. 2006, pp. 2169–2178.