ELEC 345 Assignment 6: Bag of Features

Ryan Li

April 28, 2016

Contents

Chapter 1

Bag of Features Classification with SIFT Descriptors

1.1 Introduction

1.1.1 Bag of Features

On this section of the assignment, one is asked to implement Bag of Features Algorithm in order to classify images. Starting from the reduced data set(3 classes), one is expected to perform machine learning techniques on the data set, and deliver accurate classification of test data set.

1.1.2 Scale Invariant Features Transform (SIFT)

In order to retrive features from the images, SIFT is implemented in this part of the assignment [1]. SIFT yeilds a $128 \times M$ dimension vector, while M descripes the number of feature points in the image, and each features points are descriped with a 128-dimension descriptor. Using a prewritten image processing toolbox developed by VLFeat [2], I am able to use a function (*vl_sift*) to find the SIFT features.

1.1.3 K-Means Clustering

Instead of using Matlab's built-in *kmeans()*, I used *vl_kmeans* function to compute my codebook since *kmeans()* is much more time consuming. It is worth noting that *vl_kmeans* has three different implementation of finding K-means - Lloyd(by default), Elkan, and ANN. I used Elkan's algorithm to due to it's reasonable speedup without lose of accuracy

From the following chart provided by VLFeat (Figure 1.1), it is clearly seen that three algorithm's takes various amount of time for computation. In this example of computing clusters for the three classes, I have the following result - Lloyd: 45.470475 seconds; Elkan: 35.932392 seconds; ANN: 28.137212 seconds [1]. The reason that I choose not to use ANN in this section is that instead of computing distance between data points to cluster centers exhaustively, ANN produces an estimate of cluster centers that is fairly ro-

[1] Running on Matlab 2014a, Windows 10, i7-4770 @3.40GHz

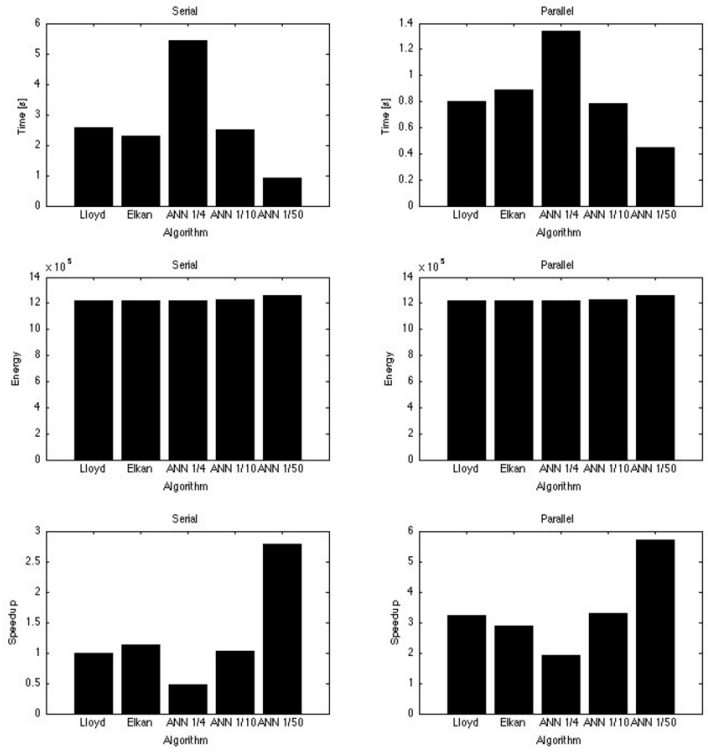

Figure 1.1: K-Means Computing Algorithm Comparison

bust for most of the classes. Thus ANN does lose accuracy, which is not suitable to use for the smaller amount of data.

1.2 Algorithm Implementation

1.2.1 Set Up

Before we dive into importing images and finding SIFT, I first have to deal with the code compatibility issue with my two systems - Mac and Windows. Thus, instead of having two versions of the code, I used *try..catch* structure

to import image directory and VLFeat's library so that the program will not fail when I switch work spaces.

1.2.2 Importing Images and Finding SIFT

The images are read into Matlab with command *imread()* and they are converted into gray-scale images as requested by VLFeat's *vl_sift* function for finding the SIFT descriptors. I encountered a problem here by simply using command *rgb2gray()*, and Matlab gave me an error message *"MAP must be a m x 3 array."* It was then when I realized that some of the test images are already in grayscale - thus I used an *if...else...* statement to skip all the already grayscale images.

Within the same loop of importing, I also did SIFT descriptor calculation in order to save time improve efficiency. For each iteration which operates on one images from the same class, after converting the images to date type *single*, the data is stored into data type *struct* under the label "SIFTDescriptor". Furthermore, these two information are also stored for future references: the name of the file and the metadata of the image. A sample class data should look like this:

airplanes

61x1 struct with 5 fields

Fields	name	metadata	SIFTDescriptor	SURFDescriptor	HOGDescriptor
1	'251_0001.jp...	4x209 double	128x209 single	64x275 double	21x50x31 single
2	'251_0002 (1...	4x238 double	128x238 single	64x239 double	23x50x31 single
3	'251_0002.jp...	4x238 double	128x238 single	64x239 double	23x50x31 single
4	'251_0003.jp...	4x169 double	128x169 single	64x260 double	21x49x31 single
5	'251_0004.jp...	4x234 double	128x234 single	64x322 double	21x49x31 single
6	'251_0005.jp...	4x234 double	128x234 single	64x316 double	22x49x31 single
7	'251_0006.jp...	4x155 double	128x155 single	64x173 double	18x49x31 single
8	'251_0007.jp...	4x152 double	128x152 single	64x210 double	19x49x31 single
9	'251_0008.jp...	4x263 double	128x263 single	64x247 double	19x49x31 single
10	'251_0009.jp...	4x179 double	128x179 single	64x240 double	20x49x31 single
11	'251_0010.jp...	4x176 double	128x176 single	64x191 double	17x50x31 single
12	'251_0011.jp...	4x218 double	128x218 single	64x248 double	20x50x31 single
13	'251_0012.jp...	4x171 double	128x171 single	64x233 double	19x50x31 single
14	'251_0013.jp...	4x248 double	128x248 single	64x268 double	22x49x31 single
15	'251_0014.jp...	4x175 double	128x175 single	64x173 double	19x50x31 single
16	'251_0015.jp...	4x338 double	128x338 single	64x414 double	25x51x31 single
17	'251_0016 (1...	4x260 double	128x260 single	64x264 double	21x49x31 single
18	'251_0016.jp...	4x260 double	128x260 single	64x264 double	21x49x31 single
19	'251_0017.jp...	4x231 double	128x231 single	64x295 double	22x50x31 single
20	'251_0018.jp...	4x243 double	128x243 single	64x313 double	23x50x31 single

Figure 1.2: Sample Data Example

1.2.3 Clustering Analysis Using K-Means

I used the suggested vocabulary size N = 1000 [2] In order to perform K-Means clustering, the data is required to be concatenated into a big matrix - in this case, I created a matrix named $bagofFeatures$ with Matlab command *horzcat()* to concatenate each image's SIFT descriptor.

Now with the matrix $bagofFeatures$ that contains all the features from the training set, I used the following line to compute the cluster centers (vocabularies)

[2]Vocabulary size is computed by the following reasoning: 3 classes, 50 images each, and each image is estimated to have around 5-7 vocabulary.

```
[centers,assignment] = vl_kmeans(bagofFeatures, N, ...
    'Initialization', 'plusplus', 'Algorithm', 'Elkan');
```

In order to understand the time needed for each clustering algorithm (Llyod, Elkan, ANN), I used Matlab command *tic...toc* to output the time needed for clustering with different algorithms by wrapping it around the lines of code I want to analyze [3].

1.2.4 Computing Histogram Model for Each Class

A histogram is generated for each of the three classes as a model, which serves a metric of describing the class computationally. Each histogram has a size as large as the number of clusters, meaning each histogram is a 1×1000 matrix. The histogram is computed with *Algorithm 1.*

There are two things that worth noting here: *knnsearch()* and my method of thresholding, namely discarding the extreme outlier samples. First, *knnsearch()* is a build-in Matlab function that does that K-Menans clustering in reverse. In another word, *knnsearch()* takes a SIFT feature from the given image, and tie the feature back to one of the cluster which is computed previously. The output of the knnsearch is the cluster index and the distance to the cluster. Second, the distance thresholding is done by computing the mean and standard deviation of the entire distances matrix as two metrics, then compare the absolute difference between the average distance within a single

[3] Refer to Section 1.1.3 for computational time to compute the cluster centers for the reduced data set with different methods.

result [4] with the mean of the model. If the result is larger than the standard deviation of the entire set, this particular index is discarded.

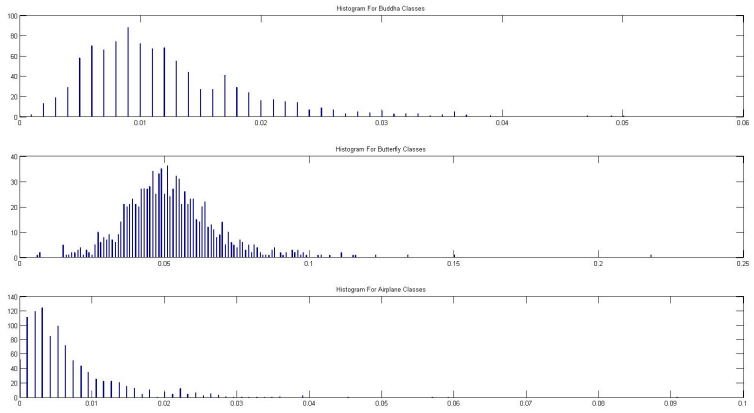

Figure 1.3: Histogram Model for the Reduced Data Set - "buddha", "butterfly", "airplane" classes from top to bottom

Later, the models (visualized in Figure 1.3) will be used for testing accuracy of recognition.

1.2.5 Testing the Accuracy of the Established Model

The testing is rather straightforward - what I did was simply modify *Algorithm 1* so that it compute the histogram for each individual image, and I used Matlab function *corr2()* (Figure 1.4) [3] to compute the correlation of the computed histogram with the three models that I have from Section 1.2.4 - the one with the highest matching percentage will get its corresponding counter increases.

[4] Refer to line 17 of Algorithm 1

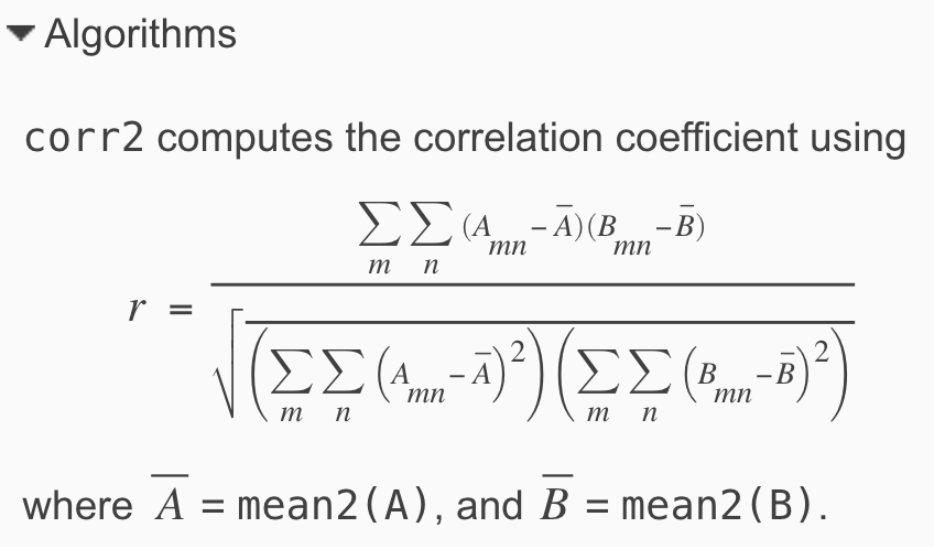

Figure 1.4: Algorithm Detail on *corr2()*

1.2.6 Confusion Matrix and Overall Accuracy

$$confusionMatrix = \begin{bmatrix} 0.6429 & 0 & 0.3571 \\ 0.3000 & 0.6000 & 0.1000 \\ 0.0625 & 0 & 0.9375 \end{bmatrix}$$

$$Accuracy = \frac{trace(confusionMatrix)}{3} = 72.68\%$$

The result of my algorithm on the reduced data set is printed above - the label for the Confusion Matrix is "buddha", "butterfly", and "airplane" from both left to right counting column-wise, and top to bottom counting row-wise.

It is worth noting here that the algorithm does not score as high as expected on differentiating "buddha" and "butterfly" class. Yet it is not that surprising, since the histogram for these two class are quite similar(see

Figure 1.3). This problem of ambiguous histogram model construction will further exacerbate when I scaled the data to 25 classes. I will further explore the scaling of data and the accuracy result in next chapter.

```
Data: A Struct type data of the desired class, imageClass; Cluster
      centers, centers
Result: A 1 × 1000 histogram model of the class, model
finalIdx ⟵ [];
codebook = transpose(centers);
for i ≤ length(imageClass) do
    queryPt = transpose(imageClass(i).descriptor);
    [tempIdx, tempDistance] = knnsearch(codebook, queryPt);
    imageClass(i).idx = tempIdx;
    imageClass(i).distance = tempDistance;
end
for i ≤ length(imageClass) do
    distanceMean(i) = mean(imageClass(i).distance)
end
modelStd = std(distanceMean);
modelMean = mean(distanceMean);
for i ≤ length(imageClass) do
    j ⟵ 1;
    while j ≤ length(imageClass.distance) do
        if abs( imageClass(i,1).distance(j) - modelMean ) ≥ modelStd
         then
            imageClass(i,1).idx(j) = [];
            imageClass(i,1).distance(j) = [];
        else
            pass;
        end
        j ++;
    end
    finalIdx = [finalIdx;imageClass.idx];
end
ux = [1:1000];
counts = hist(finalIdx,ux);
bins = unique(finalIdx);
normalizeFactor = length(bins);
model = counts / normalizeFactor;
```

Algorithm 1: Computing Histogram Model For Desired Class

Chapter 2

Can I Fix it? Yes I can!

In this chapter, I will be discussing the steps I took to improve the accuracy of image recognition and how I finally score a 55.4795% on the overall accuracy.

2.1 Attempt 1: SIFT and K-Means Clustering

In order to cope with the large amount of data that need to be imported and computed, instead of repeating the code to complete the process, I wrapped the codes I used in the reduced data set and packaged them into separate functions so that I can execute the process more elegantly. After repeating the process in Chapter one without doing any sort of optimization, I got 28.98% accuracy, and some of the classes are competently not recognized. Unsatisfied with the result of the algorithm I have, I started to try the meth-

ods mentioned in the assignment document.

2.2 Attempt 2: SURF and K-Means Clustering

2.2.1 Speeded-Up Robust Features (SURF)

Similar to SIFT, SURF is also a local feature detector in Computer Vision. Claimed to be inspired by SIFT algorithm, SURF is reported to perform several times faster and yet more robust in accuracy under certain cituations [4]. The advantages of SURF that make it superior to SIFT in some ways are as follow: integral images, Fast Hessian Detector, as well as Haarwa Velet Response [5].

Integral Image

Integral Image is the foundation of SURF as I see it - since SURF involves in computing Hessian Matrix and approximation of with box filters, the speed up from SURF is mainly due to the use of integral image for the above computation. The value of integral image at point (x,y) is given by the summation of all the pixels above and to the left of that point.

$$I_{\sum}(x,y) = \sum_{i=1}^{i \leq x} \sum_{j=1}^{j \leq y} I(i,j)$$

It is immediately clear that with Integral Image, the computational complexity is dramatically reduced. Let us look at one example here. Consider Figure 2.1, The sum of pixels within rectangle D can by computed with

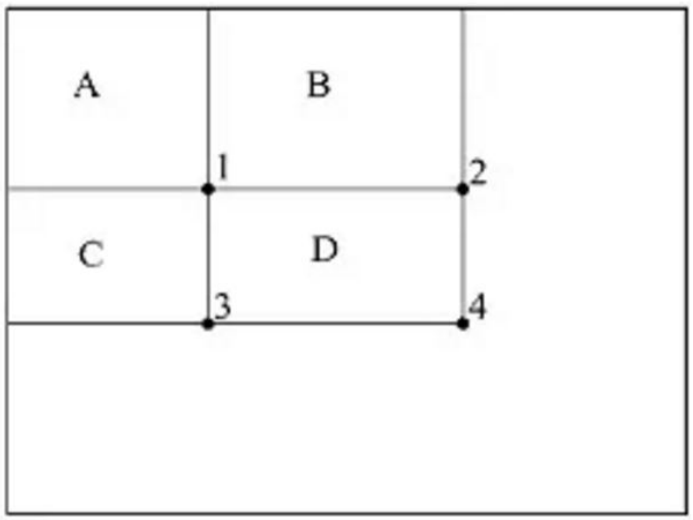

Figure 2.1: Example Illustrating the Computational Complexity with Integral Image

four arrays that are precomputed and stored in one matrix, thus rectangle D can be computed as: $D = intergralImage(4) + intergralImage(1) - intergralImage(2) - intergralImage(3)$ [6]. With the use of Integral Image, to compute Lyy value at a pixel, it requires 8 additions and 1 multiplication, which is significantly faster than that without Integral Image(44 additions and 1 multiplication).

Fast Hessian Detector

Fast Hessian Detect is a speeded-up version of the more traditional Hessian Detector (see definition below) to find interest point in the given image - usually, the interest points are at the corer pixels. This computation time is reduced by the use of Integral Images.

The Hessian Matrix is defined as below:

$$\mathscr{H}(x,\sigma) = \begin{bmatrix} L_{xx}(x,\sigma) & L_{xy}(x,\sigma) \\ L_{xy}(x,\sigma) & L_{yy}(x,\sigma) \end{bmatrix}$$

Each entry of the Hessian Matrix, L_{aa}, is a second order partial derivative of the pixel after smoothed by a Gaussian filter in a particular direction(xx, xy, or yy). If the pixel is a local extreme, it is preserved as a feature point. Figure 2.2 visualized the reason why SURF is faster than SIFT with the use of Integral Image to approximate Second Order Gaussian Derivatives.

Haar Wavelet Response

Different from SIFT which uses an orientation histogram, SURF uses the sum of Haar Wavelet Response around the point of interest. Consider Figure 2.3, where the scattered points are the points of interest and the arrow represents the sum of haar wavelet response. After scanning the the whole circle and summing up the haar wavelet response of all the the sections, we can find the orientation by picking the largest summation value.

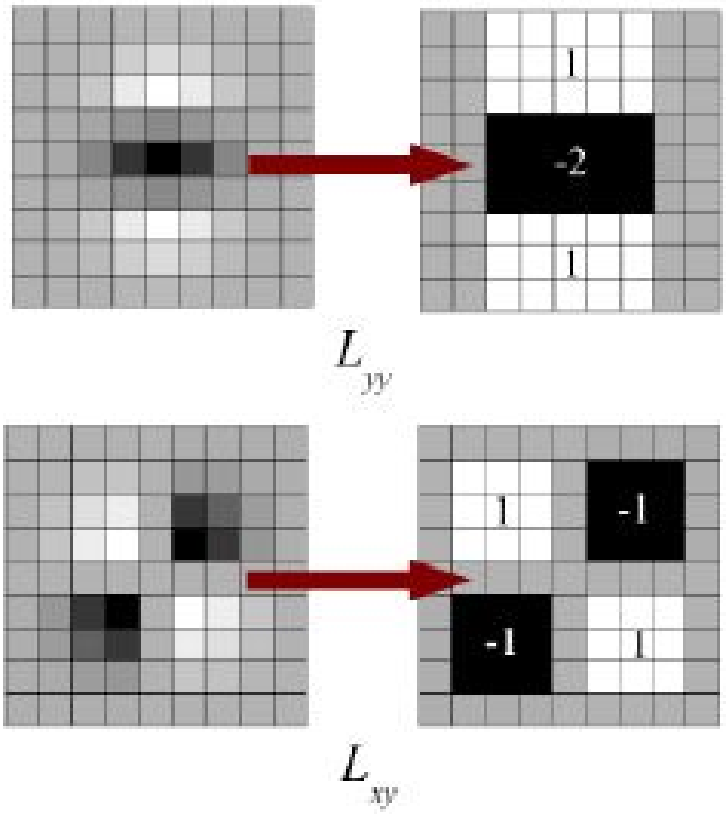

Figure 2.2: Approximating Second Order Gaussian Derivatives with Integral Images

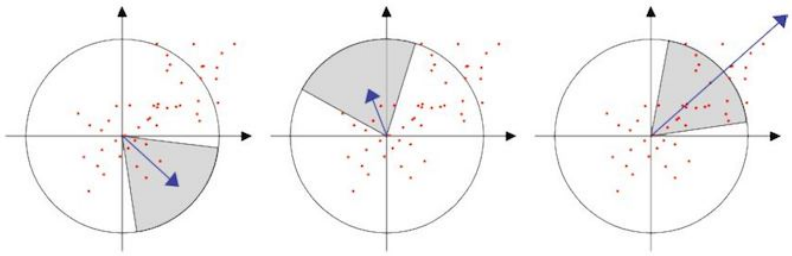

Figure 2.3: Using Sum of Haar Wavelet Response to Determine the Main Orientation of SURF Descriptor

Drawbacks of SURF Detectors

Despite of all the advantages of SURF, there are definitely some drawbacks. For example, as reported in Juan and Gwun's paper on the comparison of SURF and SIFT, they concluded that SURF *is not stable to rotation and illumination changes*. Figure 2.2 summarized their findings in a table. Thus, using SURF or SIFT is a purely application-based decision.

Method	Time	Scale	Rotation	Blur	Illumination	Affine
SIFT	common	best	best	best	common	good
PCA-SIFT	good	common	good	common	good	good
SURF	best	good	common	good	best	good

Figure 2.4: Table of Comparison Between SIFT, SURF and PAC-SIFT

2.2.2 Implementation of SURF

In the code, I used a library *OpenSurf* [7], and I stored the SIFT feature descriptors in the same struct data type for each class. An example of the importing and feature extraction process looks like this:

```
1 for i = 1:length(fileNames_basketball)
2     imageTemp = imread(fileNames_basketball{i});
3     if size(imageTemp, 3) > 1
4         imageTemp = rgb2gray(imageTemp);
5     end
6     imageTemp = single(imageTemp);
7     % SIFT
```

```
8       [f,d] = vl_sift(imageTemp);
9       % SURF
10      ipts = OpenSurf(imageTemp);
11      basketball(i,1).name = fileNames_basketball{i};
12      basketball(i,1).metadata = f;
13      basketball(i,1).SIFTDescriptor = single(d);
14      basketball(i,1).SURFDescriptor = ...
            reshape([ipts.descriptor],[64 length(ipts)]);
15  end
```

The outputted SURF descriptor is $64 \times N$ matrix, where N is the number of interested points in the given image. Each column is one entry of the SURF descriptor.

2.2.3 Performance and Confusion Matrix

To improve the performance of the modeling, I implement a hybrid of SIFT and SURF by computing the histogram and model testing once for each descriptor, then determine the best match based on the two similarity score yielded from both methods.

The reason for doing so is that I noticed that SURF is better at detecting some classes while not as ideal for the other classes, and SIFT can compensate for some of the bad matches. I determine the match index based upon the following algorithm (Algorithm 2),

With the above method, I scored 38.90%. The confusion matrix (Figure 2.5) marks the matched percentage in a heatmap, where lighter the color

Data: Accuracy from SIFT classification, *accuracySIFT*; Accuracy from SURF classification, *accuracySURF*
Result: The matched class index counter, *classMatchCounter*

```
overallAccuracy = max(accuracySIFT, accuracySURF);
[maxVal, maxIdx] = max(overallAccuracy);
classMatchCounter(maxIdx)++;
```

Algorithm 2: Hybrid SIFT and SURF for Image Recognition

in block, the higher the matching percentage, and reversely, the darker the block, the lower the match percentage. At this point, it is already visually visible that the diagonal of the confusion matrix is much lighter than the rest of the test result. However, I did not stop here; I kept asking myself "Can I do better?" As it turns out, yes I can still improve.

2.3 Attempt 3: Spatial Pyramid Matching and Intersectional Kernel Support Vector Machine

In this section, I will be discussing Spatial Pyramid Matching as well as Intersectional Kernel Support Vector Machine (SVM). With these two methods combined, I got an accuracy of 55.4795% on the expanded class (Figure 2.6).

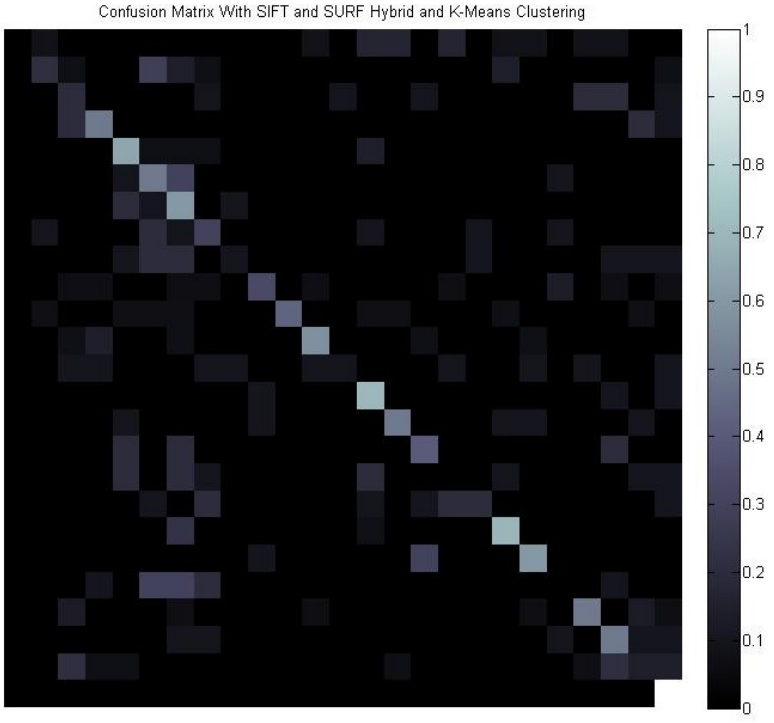

Figure 2.5: Confusion Matrix for the Expanded Data Set with Hybrid SIFT and SURF

2.3.1 Spatial Pyramid Matching

So far, Bag of Features has been performing not so robustly for the expanded class- one of the possible reason is that spatial information of the local features are not taken into account. Consider Figure 2.6, where the author of the Spatial Pyramid Matching gave an example of the algorithm. To accomplish spatial pyramid matching, the image is divided into several regions and apply BoW on each region. Then, we can compute the histogram for each region with their according spatial information coded within. The ad-

```
Command Window
Building Sift Descriptors

Descriptor has already been computed for this settings
Building Dictionary using Training Data

Dictionary has already been computed for this settings
Computing assignments
Recomputing assignments for this settings
Building Spatial Pyramid
 Pyramid has already been computed for this settings

Classification using BOW rbf_svm
Accuracy = 45.2055% (132/292) (classification)

Classification using histogram intersection kernel svm
Accuracy = 50.6849% (148/292) (classification)

Classification using Pyramid BOW rbf_svm
Accuracy = 44.5205% (130/292) (classification)

Classification using Pyramid BOW histogram intersection kernel svm
Accuracy = 55.4795% (162/292) (classification)
```

Figure 2.6: Console Output with Spatial Pyramid and Intersectional Kernel SVM

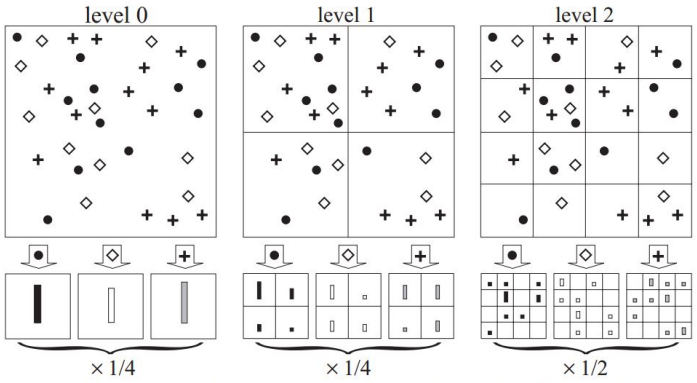

Figure 1. Toy example of constructing a three-level pyramid. The image has three feature types, indicated by circles, diamonds, and crosses. At the top, we subdivide the image at three different levels of resolution. Next, for each level of resolution and each channel, we count the features that fall in each spatial bin. Finally, we weight each spatial histogram according to eq. (3).

Figure 2.7: Toy Example of Constructing a Spatial Pyramid [8]

vantage of this method is that when the spatial information is related with visual words, the object/scene recognition is able to distinguish background and foreground. For example, consider an image of a meadow (Figure 2.8) - the histogram of part 3 will be more accurately representing meadow class with the use of Spatial Pyramid. This innovative way of feature extraction actually solved my problem in Chapter 1, where I realized that the reason for lower accuracy on buddha v. butterfly class is the histogram for these two are too similar to each other.

Figure 2.8: Example of Using Spatial Pyramid to Identify Meadow Scene

2.3.2 Intersectional Kernel SVM

Support Vector Machine(SVM) is a supervised learning method that serves as a classification method. Given a dataset containing two classes, a binary SVM builds a model that draws the line that separates the two classes, achieving a method of classification.

Intersectional Kernel SVM runs faster and more efficiently than the standard SVM. The paper showed that *"one can build histogram intersection kernel SVMs (IKSVMs) with runtime complexity of the classifier logarithmic in the number of support vectors as opposed to linear for the standard approach."* Furthermore, the paper showed that by precomputing auxiliary tables, IKSVMs is able to construct an approximate classifer with constant runtime and space requirement [9].

2.3.3 Implementation

I first try to code Spatial Pyramid Matching by myself but I later realized that the actual coding is much more complicated than I initially thought. Due to the time constriat of the project, with the help of GitHub, I found an implementation of the two methods for image classification [10] by Piji Li.

I modified the initialization code to get it work with my database, and I also modified his implementation of K-Means since I ran into an error with his exit condition for K-Means Clustering.

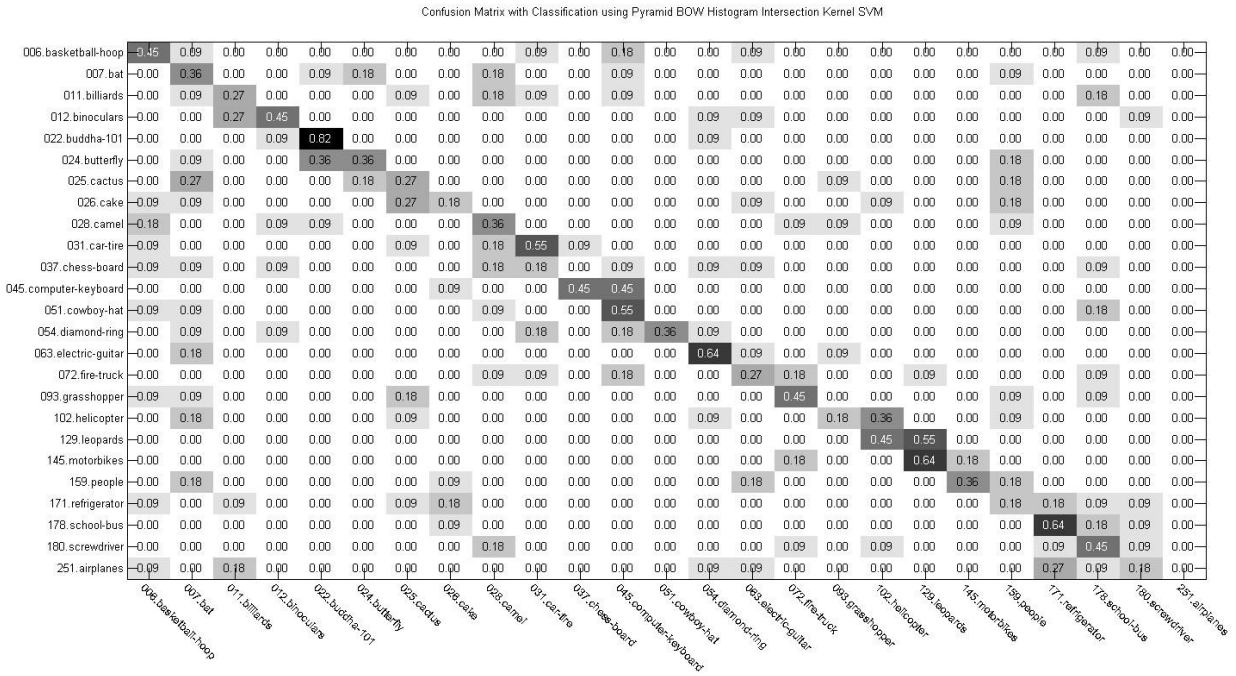

Figure 2.9: Confusion Matrix from a modified version of Piji Li's Implementation of Spatial Pyramid and Intersectional Kernel SVM

2.4 Summary and Learning Experience

Summary of Experiments on the Expanded Data Set	
Algorithm	Accuracy
SIFT	28.98%
Hybrid SIFT & SURF	38.90%
Spatial Pyramid and Intersectional SVM	55.4795%

The table above summarized my process of trial and error for the second part of the assignment. Compared to part one of the assignment, I found this part of more engaging and more challenging. The endless process of seeking perfection really drives the technology forward - it is amazing to retrace the steps from SIFT to the state of the art Spatial Pyramid Matching as well as Intersectional Kernel SVM. Besides the technical training I got from this assignment, I think personally the bigger gain is the ability to research and learn by myself.

Thank you for this amazing semester!

Bibliography

[1] T. Lindeberg, "Scale invariant feature transform", *Scholarpedia*, vol. 7, no. 5, p. 10 491, 2012.

[2] A. Vedaldi and B. Fulkerson, "Vlfeat: An open and portable library of computer vision algorithms", in *Proceedings of the 18th ACM international conference on Multimedia*, ACM, 2010, pp. 1469–1472.

[3] Matlab. (Apr. 27, 2016). Corr2, 2-d correlation coefficient, [Online]. Available: `http://www.mathworks.com/help/images/ref/corr2.html?refresh=true`.

[4] L. Juan and O. Gwun, "A comparison of sift, pca-sift and surf", *International Journal of Image Processing (IJIP)*, vol. 3, no. 4, pp. 143–152, 2009.

[5] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool, "Speeded-up robust features (surf)", *Computer vision and image understanding*, vol. 110, no. 3, pp. 346–359, 2008.

[6] A. Soman. (Apr. 27, 2016). How integral image help in speeding up the process in surf algorithm?, [Online]. Available: https://www.quora.com/How-integral-image-help-in-speeding-up-the-process-in-SURF-algorithm.

[7] D.-J. Kroon. (Apr. 27, 2016). Opensurf (including image warp), [Online]. Available: http://www.mathworks.com/matlabcentral/fileexchange/28300-opensurf--including-image-warp-.

[8] S. Lazebnik, C. Schmid, and J. Ponce, "Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories", in *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, IEEE, vol. 2, 2006, pp. 2169–2178.

[9] S. Maji, A. C. Berg, and J. Malik, "Classification using intersection kernel support vector machines is efficient", in *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, IEEE, 2008, pp. 1–8.

[10] P. Li. (Oct. 24, 2011). Image classification using bag of words and spatial pyramid bow, Shandong University, China, [Online]. Available: https://github.com/lipiji/PG_BOW_DEMO.