CS 484 – Image Analysis Term Project on Object Localization and Recognition

Ali Onur Geven, Bilkent University, Alp Güvenir, Bilkent University

***Abstract*—In this project we have aimed to perform object recognition by figuring out the animal in the image dataset based on 10 different classes and object localization by giving the boundaries of the interest object.**

***Index Terms*—Object Localization, Object Recognition, Machine Learning, SVM, VLFEAT, LIBSVM**

I. INTRODUCTION

THE aim of this project is to figure out the class of the object in an image within giving the boundaries of where the object is located in the image. There are 398 training images and 100 labeled test images.

II. PREPROCESSING STAGE

When the images in the data are visualized we have figured out that the size of the images are highly varying. For this purpose we have decided to fix the size of the images to 300 pixels to 300 pixels by using the resize function predefined in MATLAB. After that we have decided to make grids on our training images on size 50 pixels to 50 pixels. Therefore we obtain 36 visual-words from each image we have. There are 398 training images and we extract 36 grids from each image where we obtain a bag-of-visual-words structure having 14328 grids in total.

Since we have resized each of the training image, the scale of the features have varied however within using `SIFT` we are able to overcome the scale problem. As a recall, `SIFT` stands for scale invariant feature transform where within using the function call `vl_sift()` we obtain so called interesting points in an image that can be extracted to provide a feature description on the object in an image [3].

The interest points marked on the image using `vl_sift()`can be shown as the examples on images given as in Figure1, Figure2 and Figure3 for different classes.

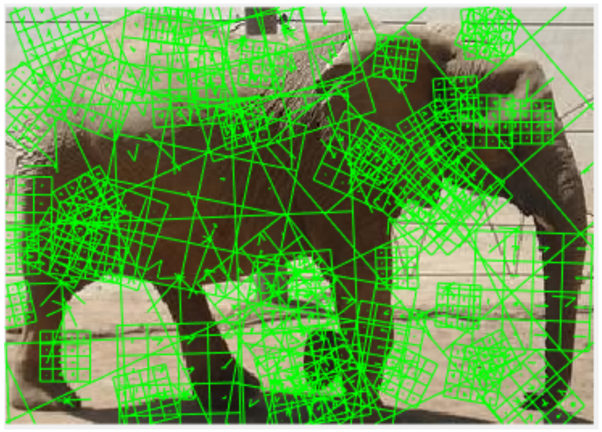

Figure1

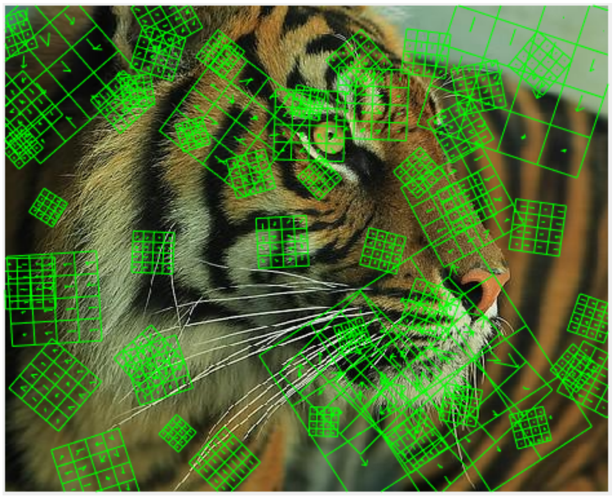

Figure 2

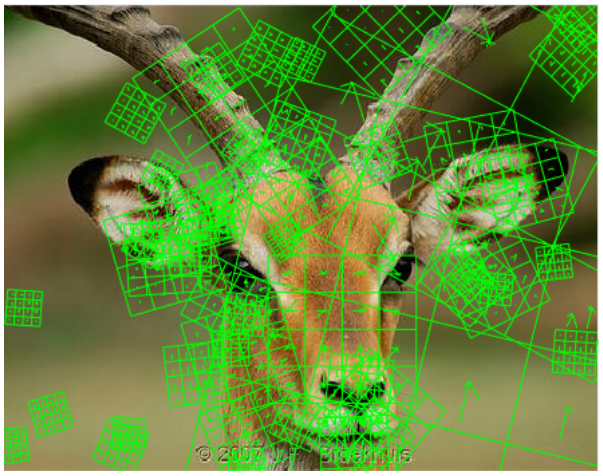

Figure 3

From the training data we know that there are 10 different classes and as stated before we extract 36 grids from each training image therefore we decided on to perform `kmeans()` with 360 clusters using the built-in MATLAB function. The return values of `kmeans()` includes the codebook that has clustered the local descriptors in grids that we have formed from our training data. Thus in our codebook we have an array with size one row to 14328 columns where each cell gets a value between 1 to 360, which is the respective cluster value.

Sample bag-of-words for class 'n02422699' can be given as the following in Figure 4:

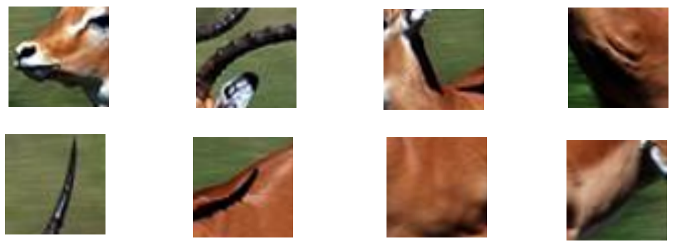

Figure 4

There are total of 36 grids for each image, whereas here we have displayed only 8 of the bag-of-visual-words that can be considered as the most important ones.

Here we know that first 36 columns are the respective grids of the first image. Respectively the next 36 columns of the codebook represent the next image and so on.

Later on we have counted the number of classes as to be the number of folders and number of images in each folder which are the training data for each class. Here we have saw that the training images are uniformly distributed with 41 or 42 instances of images for each class.

Next we have created object histogram where for each image we have visited the codebook and found the cluster id (which is between 1 to 360) for each 36 grid and incremented the occurrence of the cluster id. Thus we have hold an array of 360 elements and the total of elements in the array makes 36 for each image.

III. Training Stage

Next we have created an instance matrix, which holds the object histograms, and an object labels matrix, which has 1 for the respective class and -1 mask for the other classes. Thus within using these two matrixes we have called the `svmtrain()` method from the `libsvm`. In the training process we have used the linear kernel function by setting the -t flag to 0 and performed a support vector classification by setting the -b flag to 0. Thus in the training process we have created 10 different models in a cell array where each model is for a different object class so that we would be able to use each model in the prediction of an image an look for the highest accuracy.

IV. Testing Stage

Next for the testing stage we have bounding boxes in an image by using the predefined method called `edgeBoxes()` [2]. Here we have used the pre-trained model and optimized values calculated before. Within the returning top 10 bounding boxes we have cropped the respective section of the box from the main image, resized it 300 pixels to 300 pixels and performed a gridding operation on the new image as in slices of 50 pixels by 50 pixels thus in total having 36 visual words for our edge boxes. Here we have chosen the top 10 bounding boxes due to computation time. Within these visual words we have used the function call `vl_sift()` we obtain so called interesting points in an image that can be extracted to provide a feature description on the object in an image.

With the return value from the `vl_sift()` we have called `kmeans()` in order to cluster grids of the edge boxes to 360.

Below we can see 3 of the returning bounding boxes using the `edgeBoxes()` method under Figure 5 for 23rd image in the test data set whereas Figure 6 is the original image.

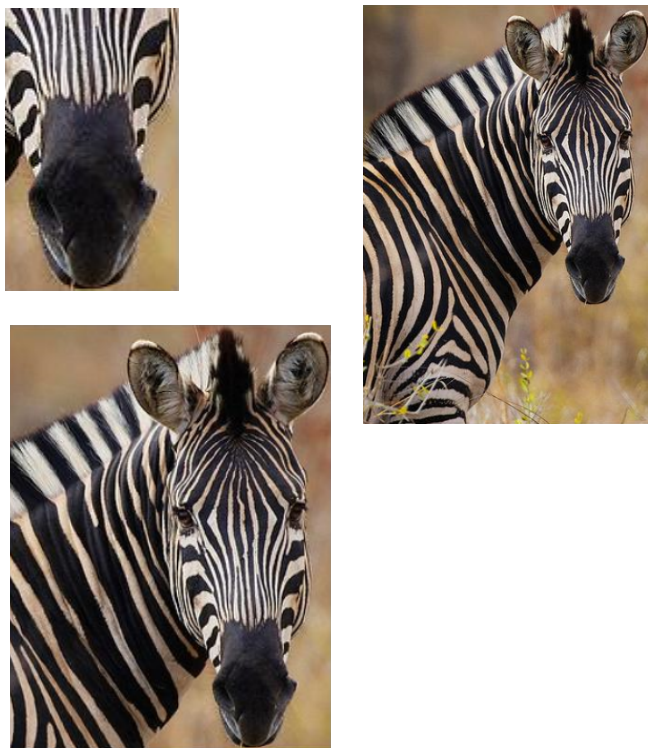

Figure 5

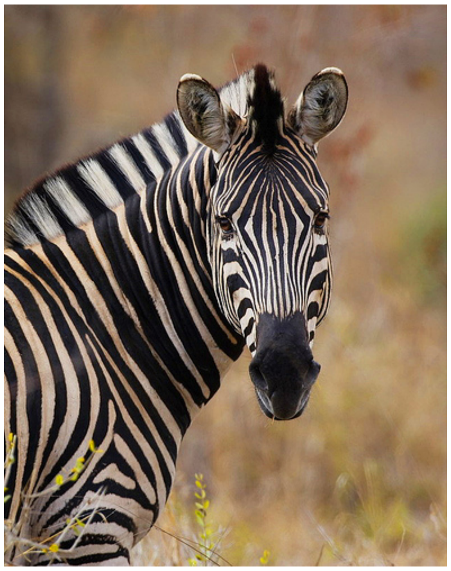

Figure 6

Another example can be given for the 42nd image in the test data set where Figure 7 provides 3 of the 10 candidate windows of the original image denoted in Figure 8.

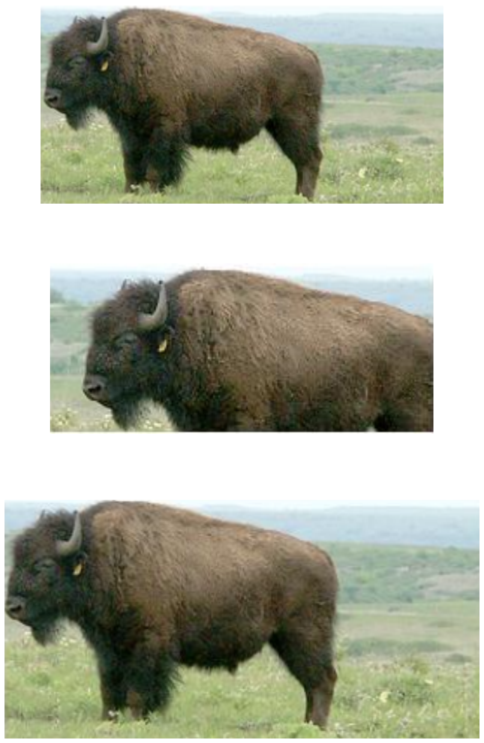

Figure 7

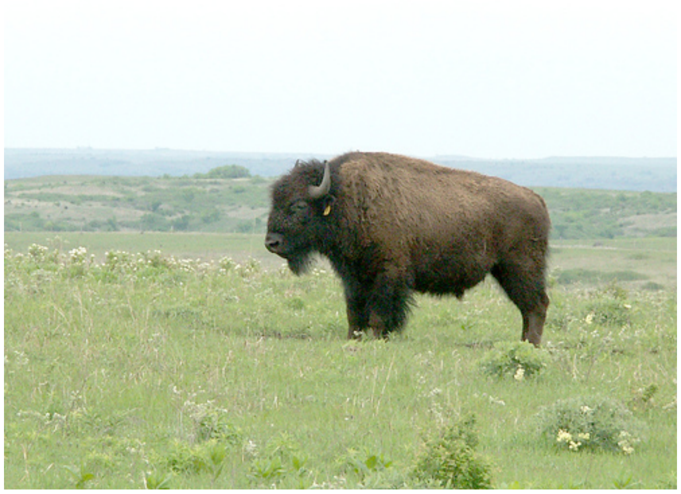

Figure 8

The test images include a great variety on background scenery. Thus using `edgeBoxes()` successfully detect the interest objects that correspond to our class labels. As we can see from the examples given below the method extracts the object from the scenery.

Within using the return value from the `vl_sift()` we have constructed another codebook for the test images. Thus furthermore we have calculated the object histograms of our test images using the test codebook. This codebook has been used for reconstruction of the test matrix of size 10 row to 360 columns which stands for the object histogram of 10 candidate windows of an image.

The ground truth tables were constructed from the labels in the `bounding_box.txt`. For this purpose we have assigned value 1 to for the respective label and the rest is kept to be as -1 in the ground truth array.

Next we have called `svmpredict()` to find the accuracy of each SVM model to predict the ratio of that class similarity for the respective test image.

The results that we have hold from the prediction process is given below in Table 1:

Class labels	Number of instances	Number of test images classified correctly
'n01615121'	10	8
'n02099601'	10	4
'n02123159'	10	0
'n02129604'	10	2
'n02317335'	10	3
'n02391049'	10	4
'n02410509'	10	5
'n02422699'	10	4
'n02481823'	10	5
'n02504458'	10	3

Table 1

V. Interpretation of Data

As we can see from the Table 1, most of the classification is done within 40%-50% interval. There are some exceptional categories where the classification process is successful and poor instead of a mediocre precision.

Such high rates of error are caused potentially because of the selection of `edgeBoxes()` method returning a selection of the image that leads to a misclassification by the SVM models. This is a reason for an error since the training data consist of images that do not include background scenery that cause variety in the classification model.

The success rates of the edge boxes can be found throughout the intersection area of the best candidate window and the bounding box values given in the `bounding_box.txt` file given as the labels of the test images. Here, a small ratio of intersection area shows that the candidate window detection is not successful enough. Intersection rates above 50% are considered as successful detection of the interest object. But most of our intersection data ended up not exceeding the 50% threshold. Some of the intersection percentages are given below in Table 2.

```
For image 82.JPEG Intersection area percentage is: 48.88 %
For image 83.JPEG Intersection area percentage is: 8.23 %
For image 84.JPEG Intersection area percentage is: 35.00 %
For image 85.JPEG Intersection area percentage is: 21.98 %
For image 86.JPEG Intersection area percentage is: 56.06 %
```

Table 2

VI. Conclusion

First of all, the scale variety of the training images was a challenge in order to extract useful features. We have overcome this problem by resizing the images and using `SIFT` which is a scale invariant feature detection algorithm.

In order to create a bag-of-visual-words model we have generated a codebook by applying `kmeans` clustering algorithm to the detectors returned from `SIFT`.

Furthermore we have modeled 10 different SVM models in order to overcome the challenge of multivariable SVM with binary SVM.

However the training images where tightly cropped around the interest object whereas the test images where not. Therefore we have overcome this problem by extracting the candidate windows for each image.

Then in order to classify the images more accurately we generated bag-of-visual-words for each candidate window. With using the bag-of-visual-words we have extracted object histograms of the candidate windows of the test images. Furthermore we have predicted the candidate window label by using 10 different SVM models trained before hand. By selecting the most accurate candidate window we have also predicted the label of respective test image. Also with using the most accurate candidate window we tried to localize the object by bounding boxes.

In total we have predicted 38 out of 100 test images correctly. During this process the bounding boxes generated using the ready code where approximately 40%-50% consistent in detecting the interest object intersection area. This has generally caused our precision of classification to be poor.

REFERENCES

[1] Zitnick, C. Lawrence, and Piotr Dollr. ”Edge boxes: Locating object proposals from edges.” European Conference on Computer Vision. Springer International Publishing, 2014

[2] Pdollar. "Pdollar/edges." GitHub. N.p., 06 July 2016. Web. 26 May 2017.

[3] Vlfeat. "Vlfeat/vlfeat." GitHub. N.p., 09 Dec. 2015. Web. 26 May 2017.

[4] "VLFeat.org." VLFeat - Tutorials SIFT Detector and Descriptor. N.p., n.d. Web. 26 May 2017.

Ali Onur Geven. Author Peer coding, Reporting, SVM model usage, Test Object Histogram

Alp Güvenir. Author Peer coding, Reporting, Codebook construction, Object Histogram