CMSC426 Project #3: SVM

Group Members: Andrew Sumner (asumner1), Teimuraz Trapaidze (ttrapaid)

Contributions: As with the last project, we did pair programming via Github and Discord screen-sharing. We alternated on a regular basis between who would code and who would look up documentation while double-checking the code via video feed. We wrote up the final report at the same time via Google Docs.

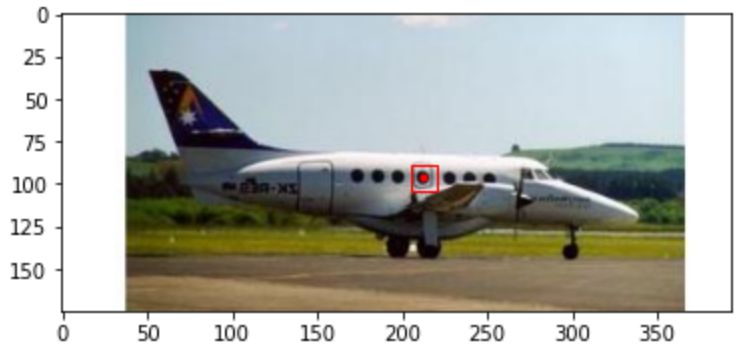
We implemented the SVM training by following the general structure outlined in the project description. The layout of our notebook is as follows: each step of the project is laid out, followed by a description of the step as well as any relevant method names and any relevant images/visuals.

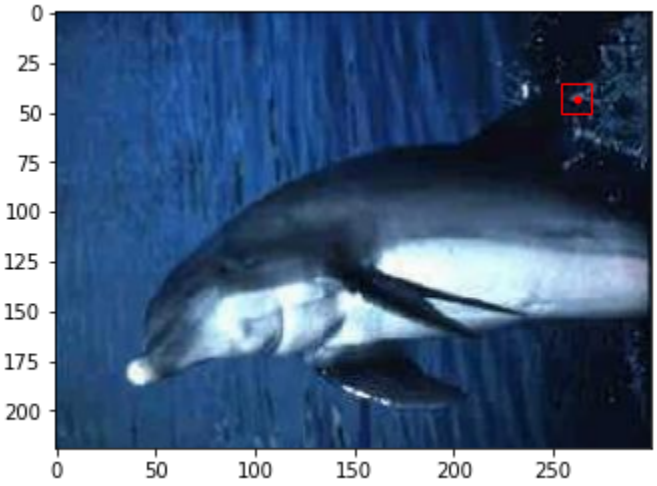
Feature Extraction

> For this project, we used the SIFT methods as provided by the OpenCV SIFT_create() library. Namely, we used SIFT_create() to construct the SIFT object and then detectAndCompute() to get key points and their respective descriptors for every image in each category. Finally, we used the numpy.vstack() method to put all of these descriptors into a single matrix for our K-Means algorithm/method.

K-Means:

> ```
> def clustering(descriptors, k)
> ```
>
> For this algorithm, we followed the lecture for Bag of Visual Words, specifically slide 43. The function we wrote picks k random descriptors to represent k initial centers from the given list of descriptors of all images. Then, our method will find the closest descriptors for each center, update the list of k centers with the center of mass of each set of points, and repeat until convergence. We initially tried using np.sum/np.linalg.norm to calculate the center of mass for a set of descriptors, but we found that the updated centers of mass would get smaller and smaller until every single descriptor was closest to the smallest center. We switched to using the mean of the stacked set of descriptors to calculate the center of mass and found much better results.

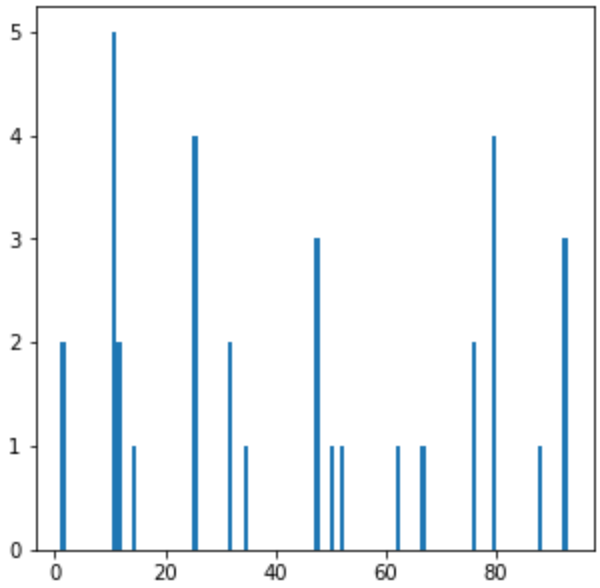


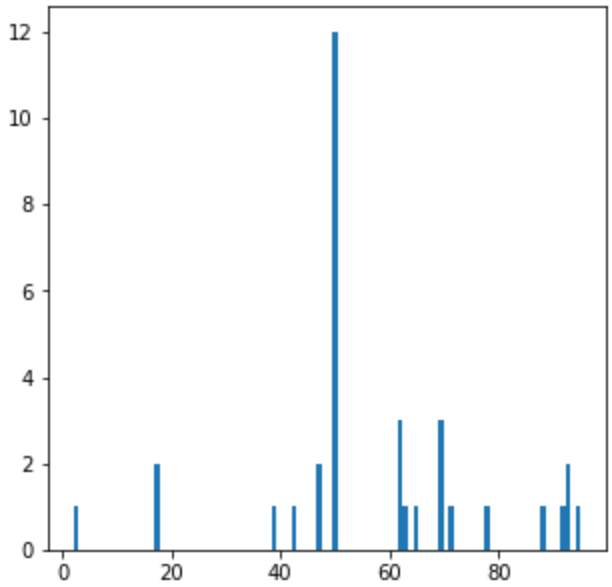


Bag of Visual Words:

```
def generate_histogram(descriptors, centers)
```

In order to acquire a bag of visual words for a given training image, we wrote the method generate_histograms(), which takes the list of descriptors for the image and the list of finalized centers/centroids from clustering() as parameters. generate_histograms() will go through the centroids for each of the given descriptors and compare their distance, and tally up the occurrences of the closest centroids into a histogram. This is the histogram that represents the bag of visual words for that training image.





SVM Training:

Lastly, we used the “One vs. All” Support Vector Machines technique to classify each image based on its histogram, then we printed the corresponding confusion matrix. These steps were relatively simple since we were allowed to utilize external libraries for this section. We chose to use sklearn for the SVM classifiers and to find the confusion matrix. This involved creating the model via sklearn’s SVC() and OneVsRestClassifier() functions, as well as fitting the model to the training histograms and training labels. The “One vs. All” technique allows SVM to use binary classifiers for each of the image categories, where one category is divided from each of the other categories.

Testing/Prediction:

We did not get particularly good results after passing in our testing data to the trained SVM classifier. The lack of good results could be attributed to our calculation or determination of the centroids; we speculate that the root of the issue is in the smaller training set size. We used the minimum number of keypoints among the images to limit the number of keypoints/descriptors across all the images; furthermore, we only had 65 images in our training set, so the combination of the limit on the number of keypoints and the limit on the number of images may have affected our results.

	precision	recall	f1-score	support
0	0.67	0.57	0.62	7
1	1.00	0.43	0.60	7
2	0.50	0.86	0.63	7
accuracy			0.62	21
macro avg	0.72	0.62	0.62	21
weighted avg	0.72	0.62	0.62	21

This model reported a 62% accuracy and was the best accuracy that we found across numerous trainings/model generations.

```
          dolphin airplane leopard
dolphin  [  4   0   3]
airplane [  1   3   3]
leopard  [  1   0   6]
```

According to the confusion matrix that this model generated, it accurately predicted 4/7 (57%) dolphins, 3/7 (43%) airplanes, and 6/7 (86%) leopards. We hypothesize that our ability to recognize leopards at a higher accuracy than the other two subsets stems from the fact that leopards have many easily-identifiable-by-SVM spots on their bodies, whereas the smooth surfaces of dolphins and airplanes are not so easy.