Project 3 Report: Image Classification

Using Bag of Features and Support Vector Machines

CMSC 426 Spring 2021

Hongyu Tu 115323568

1. Load & Prepare data

For this project, we are using the three categories: airplane, dolphins, and leopard from the Caltech-101 dataset. Since these three sets each have 800, 65, 200 images, to avoid bias from the input training data, I end up decided to read in 80 images for airplane, leopard, and all 65 for dolphins (I will discuss how I come up to the numbers in step 4). A total of 265 images are all then passed into the openCV's SIFT function to obtain the SIFT keypoints and their corresponding SIFT descriptors. Image patches of size 10 pixel by 10 pixels, corresponding to each SIFT keypoints, are saved into a dictionary with their keys equal to their descriptor for later use. Also, for balancing, the image with minimum number's feature size was found to be 36, so I kept the feature size for all image to be 36 as well.

After obtaining all the above data and before feeding them into the next step, the list of data will be subsampled again, evenly between each category, since we to keep the number of SIFT descriptors from each category similar as well. As the instruction required, I'm using a train/test ratio of 9:1, which leaves me with 72, 58, 72 images in the training set for each category. The final output of this step will be a list of SIFT descriptors randomly selected from three categories. Here below are two examples with feature size set to 1500 and 4500:

```
 --- classification starts [k = 200, feature size = 1500] ---
Number of images in each training set:           [72, 58, 72]
Number of features in each training set:         [2592, 2088, 2592]
Number of features in training after subsample:  [502, 495, 503]
Shape of training set after random sampling:     (1500, 128)
 --- classification starts [k = 200, feature size = 4500] ---
Number of images in each training set:           [72, 58, 72]
Number of features in each training set:         [2592, 2088, 2592]
Number of features in training after subsample:  [1487, 1501, 1512]
Shape of training set after random sampling:     (4500, 128)
```

Fig 1. Results from subsampling the descriptor list to obtain the training set for next step

2. Creating bag of visual words

2.1 K-Means

In this step, I made two functions to implement the K-Means algorithm. When the list of features (SIFT descriptors) of size **S** and the number of clusters **C** are passed in, the main function will first randomly pick **C** number of descriptors (shape 128 by 1) as the initial centroid. Then, the feature list, along with the centroid list, will be passed into a second function, where

the features will be divided into **C** clusters, depending on each descriptor's distance to the centroids. The centroid list will then be replaced by the mean of descriptors in the same cluster. To minimize the internal distance within each cluster and maximize the external distance between different clusters, we feed to new centroid list back into the second function again until there's no change to the list of centroids.

For my original implementation, I used a nested loop in python to find the distance from all the features to all the centroids, and that takes, 400 (number of clusters, **C**) times 6000 (number of centroids, **S**), 2,400,000 iterations. Python is notoriously known for its slow loops and my K Means takes an hour to run, which is pretty much unusable. Therefore, I rewrote my function by using broadcasting, Boolean indexing, and spatial data structures: creating a **C** by **S** distance matrix, then use the "min", "argmin", along with "where" functions in NumPy to avoid loop in python. That worked well and cut down the runtime from over 1 minute to about 10 seconds, and my K-Means overall can be done within 2 minutes instead of hours.

2.2 bag of visual words

From step 2.1, I obtained the centroid list that is already most optimized. Each centroid in that list will be a visual word that has a size 128 by 1. Since all the centroids in that list are the result of averaging all the features in a cluster, we can't find an exact match to the original list of features, but we can still find the closest. That will give us two things: 1. Reshape each centroid to 16 by 8 to get a representation of each visual word, and 2. Match the centroid to the closest feature and obtain the image patch that we made in step 1. Below are some examples of visual words that I found:

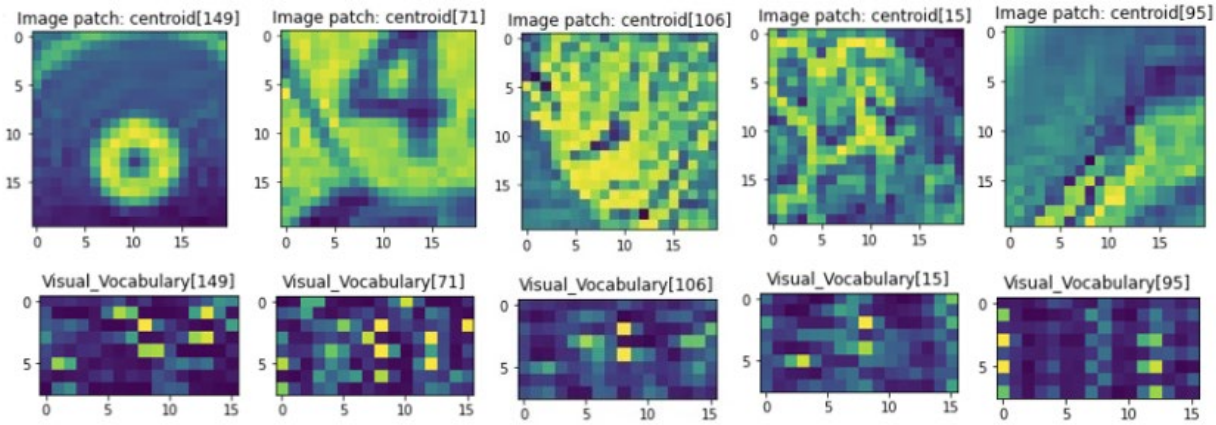

Fig 2. Plot of visual words and their corresponding image patch

From the five images above, I think #149 and #71 are features that we expect only to find on airplanes, circular shapes, and characters/numbers. For #106 and #15, I think those are features of leopards as they usually have spots on their body that sort of looks like camouflage. And lastly, #95 looks like the face of a dolphin.

For the bag of visual words, I simply loop through the entire image list (size of 265), to convert them all into visual words that will later be used for SVM. For each image, I first get the list of descriptors in that image, then I spread those descriptors into **C** number of bins according to the centroid list just like what we did in K-Means. The bag of the visual words will be size **C** (Again, number of clusters) by 1 and each digit represents the count of features that fall into that one specific cluster. For each category, the first 90% of images get assigned to the train set for SVM and the other 10% get sent to the test set. Here are some histograms that I made, the difference in distribution for both the three on the left and three on the right are pretty obvious:

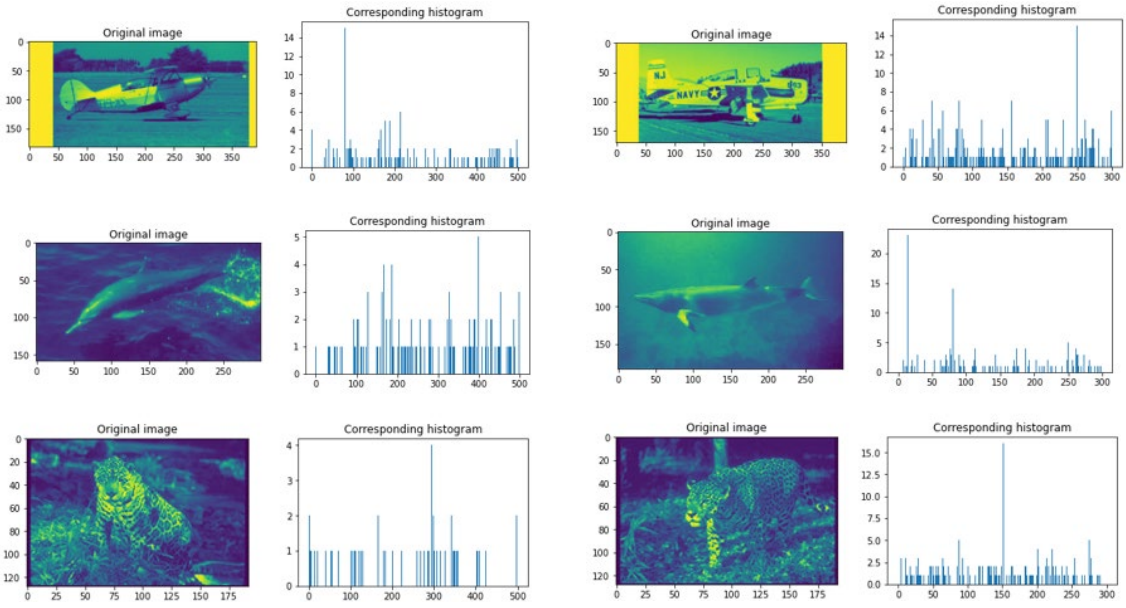

Fig 3. Plot of histogram for images in different categories (Left k = 500, Right k = 300)

3. SVM Classifier Training

For this part, I'm just using the SVM classifier from the scikit-learn library. X is the list of all images. And for y, since we want the classifier to be one vs. all, when training each category's classifier, the images from the other categories will be indicated as 0 and the images from this category will be 1. I made a list hosting the three classifiers for each category, and a for loop to go through the train set I made in step 2, to train the classifiers one by one. From this, I get a list of one vs. all classifiers.

4. Test model

To try out my models and test different paraments (number of clusters, number of features), I made a for loop to run the overall classify function. I made four tables to indicate the output under different conditions. The four tables each indicate the number of correct predictions, number of wrong predictions, both in the training set and test set. Here is the result:

--- Train Correct ---
airplanes classifier (72)

	1500	2500	3500	4500	5500
200	71	71	71	71	71
300	71	71	71	71	71
400	71	71	71	72	72
500	71	71	71	72	71

dolphin classifier (58)

	1500	2500	3500	4500	5500
200	54	54	55	55	57
300	56	54	55	56	58
400	55	56	56	56	57
500	57	57	57	56	56

Leopards classifier (72)

	1500	2500	3500	4500	5500
200	71	71	72	72	72
300	72	72	72	72	72
400	71	72	72	72	72
500	72	72	72	72	72

--- Train Wrong ---
airplanes classifier (72)

	1500	2500	3500	4500	5500
200	0	0	0	0	0
300	0	0	0	0	0
400	0	0	0	0	0
500	0	0	0	0	0

dolphin classifier (58)

	1500	2500	3500	4500	5500
200	0	0	0	0	0
300	0	0	0	0	0
400	0	0	0	0	0
500	0	0	0	0	0

Leopards classifier (72)

	1500	2500	3500	4500	5500
200	1	0	1	0	1
300	1	0	0	0	0
400	2	0	0	0	0
500	0	0	0	0	0

--- Test Correct ---
airplanes classifier (8)

	1500	2500	3500	4500	5500
200	7	7	7	7	7
300	7	7	7	7	8
400	8	7	7	6	7
500	7	6	8	8	7

dolphin classifier (7)

	1500	2500	3500	4500	5500
200	6	6	6	7	6
300	6	5	6	6	5
400	7	7	5	6	5
500	4	4	5	5	5

Leopards classifier (8)

	1500	2500	3500	4500	5500
200	8	7	6	7	6
300	7	7	7	8	7
400	7	7	7	7	7
500	7	6	5	7	7

--- Test Wrong ---
airplanes classifier (8)

	1500	2500	3500	4500	5500
200	0	0	0	0	0
300	0	0	0	1	0
400	0	0	1	0	0
500	0	0	0	1	0

dolphin classifier (7)

	1500	2500	3500	4500	5500
200	0	1	1	0	0
300	1	0	1	0	0
400	0	1	0	0	0
500	0	0	0	0	0

Leopards classifier (8)

	1500	2500	3500	4500	5500
200	0	0	0	0	0
300	0	0	0	0	0
400	0	0	0	0	0
500	0	0	0	0	0

Fig 4. Screenshot of the model test summary

From the four tables above, it's obvious that going for huge feature size and huge k doesn't provide improvement and quite opposite, which is a sign of overfitting, and also the runtime for larger k and larger feature size is longer. Therefore, I decided to settle with k = 300 and feature size = 3500 as the parameter for my model.

Another attempt of mine is to try to use 200 images for airplane and leopard instead of 80 in step 1. Although I balanced the input by subsampling those 120 more images brings a more generalizable pool of features and that gave airplane and leopard a huge advantage, but dolphin's classifier struggles to with a limited pool of features. For the overall performance, I decided that it's best to keep the image number similar as well on top of the balancing in feature sizes from each category.

5. Result

After all the steps above, I did classification with 80 images each from airplane and leopard and 65 from the dolphin. The cluster size is 300 and the feature size is 3500. For the 3500 features in the training set, airplane, dolphin, leopards each has 1167, 1159, 1174 features. Those will be feed into K-Means and then get bags of visual words.

```
 --- classification starts [k = 300, feature size = 3500] ---
Number of images in each training set:           [72, 58, 72]
Number of features in each training set:         [2592, 2088, 2592]
Number of features in training after subsample:  [1167, 1159, 1174]
Shape of training set after random sampling:     (3500, 128)
```

Fig 5. Data preparation for final model

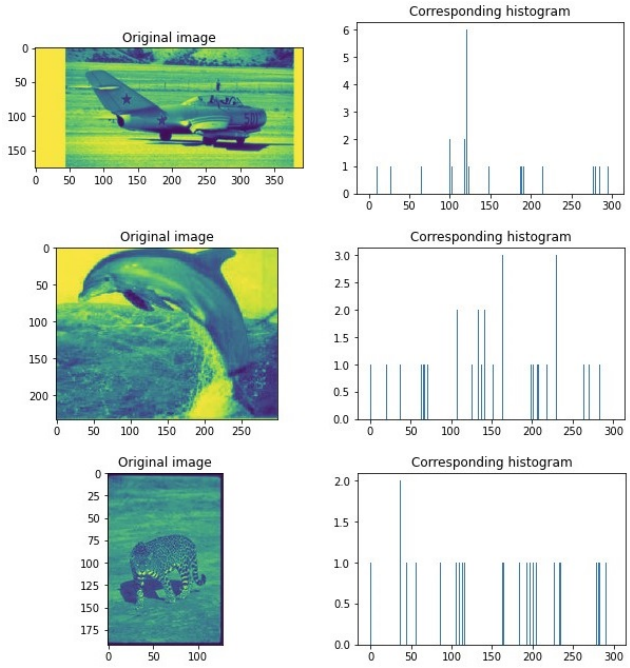

Fig 6. A plot of the histogram from the final model

```
Result of classification with
k = 300, feature size = 3500

train confusion matrix
                 airplanes  dolphin  Leopards
airplanes (72)          71        0         0
dolphin   (58)           0       56         0
Leopards  (72)           0        0        72

test confusion matrix
                 airplanes  dolphin  Leopards
airplanes (8)            8        0         0
dolphin   (7)            0        7         0
Leopards  (8)            0        0         7
```

Fig 7. Confusion matrix of the final model

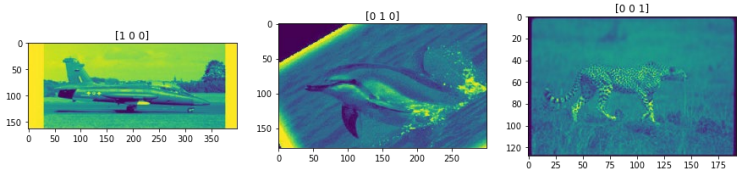

Fig 8. Output of the classifiers for picture in each category (Form: [airplane, dolphin, leopard])

Overall, I think the classifier performed pretty well, with this limited amount of training data, and the run time is about 15 seconds. None of the classifiers predict wrong in the testing set, and both classifiers for airplane and dolphin are 100% correct, the leopard has only one wrong. Although the test set is really small, and I think the result might not be super generalizable. As for the training, 2 images of dolphin weren't picked up by the dolphin's classifier and the leopard's classifier, and only 1 image that was missed for airplane. Since my code randomize my training when picking the images, and the output isn't always this good. With that said, I think this combination of three 1 vs all classifiers did pretty well.