

Identifying Feathers

Using BOW & SVM

Nicholas Carpenetti

Goals

The goal of this project was build a program using OpenCV that can identify feathers and notify the user what birds they came from. This would be done with the use of a Bag of Words (BOW), and a scaled vector machine (SVM) for classification. In addition, it was also the goal of this project to determine what impact choices such as feature extraction methods used and number of words used in the BOW had on the effectiveness of feather identification.

Compilation Notes

To use a Bag of Words model, we need to use feature extraction methods, such as SIFT and SURF. These do not come with OpenCV 3.3.1 by default, so it had to be compiled in combination with the `opencv_contrib` repository. The project assumes this is located at `C:/opencv-3.3.1`

Using The Program

This project produced a program that can load groups of images in categories to train a BOW & SVM, and allows for images to test against the trained BOW & SVM to see how well it performs.

Arguments

When run, the program must be fed 3 arguments:

1. The name of a file detailing training information
2. The name of a file detailing testing information
3. An optional directory where the files are located. If none is given, the directory is assumed to be the current one.

.TRN and .TST Files

.TRN and .TST files are both text files that contain the necessary information to train and test the Identifier. .TRN files and .TST files are formatted as follows:

FIGURE 1. .TRN FILE

```
<feature extractor>      <# words>
<category name 1> <# train images> <directory>
<category name 1> <# train images> <directory>
<category name 1> <# train images> <directory>
...
```

FIGURE 2: .TST FILE

```
<category name 1> <# test images> <directory>
<category name 1> <# test images> <directory>
<category name 1> <# test images> <directory>
```

...

Directory Structure

For the program to be able to access the files it needs, the files need to be organized into a specific directory structure. The root of this structure is either the same as the executable, or specified by the optional third argument to the executable.

FIGURE 3: EXAMPLE DIRECTORY STRUCTURE (DIRECTORIES IN BOLD)

- **WorkingDirectory**
 - **TEST**
 - **Category1**
 - Category1_0.jpg
 - Category1_1.jpg
 - ...
 - **TRAIN**
 - **Category1**
 - Category1_0.jpg
 - Category1_1.jpg
 - ...
 - ____TST
 - ____TRN

Batch Files

The batch files for running the program follow a naming scheme. They start with an indicator for if the feathers have clean backgrounds or noisy backgrounds (CL or BG), followed by the feature extractor (SIFT or SURF), followed by the number of words (100 or 1000). For example, the batch file for testing feathers with backgrounds, using SIFT and 1000 words is as follows: BL_SIFT_1000.BAT.

The one exception to this rule is the one for testing on images that aren't feathers. This one is called VerifyTest.BAT

All of these batch files are located in the exe folder with the executable.

Data Gathering

To provide the feather data to be used for training Feathers, images of feathers were acquired from the US Fish & Wildlife Forensics Lab Feather Atlas (<https://www.fws.gov/lab/featheratlas/browse.php>).

Helen also visited the PAWS Wildlife Rehabilitation Center, and took pictures to use as samples for testing.

These images were edited and placed into two groups: Feathers with blank backgrounds, and feathers with varied backgrounds, as I wanted to see what difference the background would make on the performance of the Identifier.

FIGURE 4: PROCESSED FEATHER IMAGES



Program Design

The process in the program can be divided into three major steps: Loading training data, training the BOW & SVM with the training data, Loading testing data, and testing the testing data on the trained SVM.

Loading Training Data

The loading process is started by opening the .TRN file specified by the program argument. This file contains information for what settings to use for the BOW, which categories to use, and where the images for each category are to be found.

The training data, when loaded is stored in a vector of structures called ImageSets. Each ImageSet object represents one category of images. As such, it consists of a vector of Mat objects, a name of the set, and a label to use for the SVM training/testing.

FIGURE 5: IMAGESET STRUCT

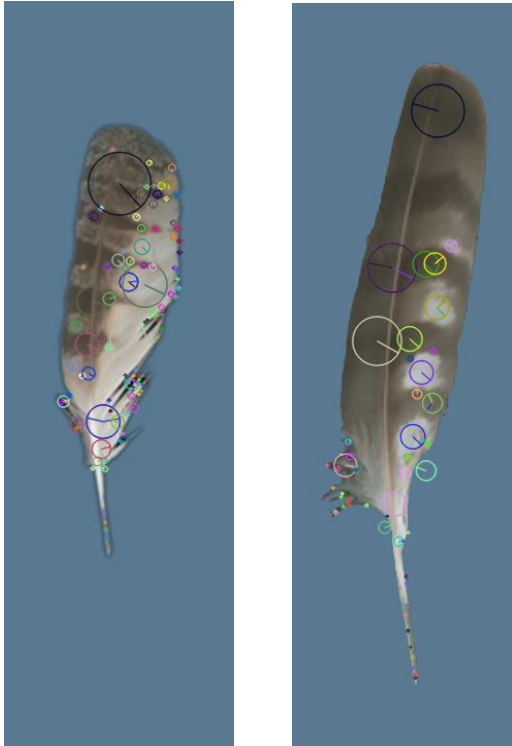
```
typedef struct
{
    vector<Mat> images;
    string name;
    int label;
} ImageSet;
```

Training The BOW & SVM

Once data structures to hold the training data have been created, the Identifier starts using all of the images to train the BOW.

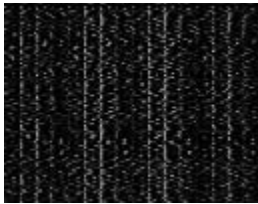
For each image being trained, the Identifier finds all the keyPoints in the image, and then extracts features at each of these keyPoints. These features are the visual “words” to be used in the dictionary. Each time these descriptors are found they are added to an array holding every word found in the entire set.

FIGURE 6: KEYPOINT DETECTION ON FEATHERS(SIFT)



Once all of the words in the set are found, a `BOWKmeansTrainer` class the OpenCV library provides performs a K-means segmentation algorithm in feature space on the entire array of features. In this case, the number of centers is the number of words in the dictionary. Once the K-means algorithm is done, we have a complete dictionary Mat object, which looks like Figure 7. In this case there are 100 words and 128 values per SIFT feature, so you end up with a 128 x 100 greyscale image.

FIGURE 7: A 100 WORD BOW DICTIONARY USING SIFT



This image can be saved and retrieved for reuse so you don't have to perform a time-consuming enormous k-means every test.

Now that a dictionary has been created, it is time to train an SVM. This is done by first making histograms of occurrences of dictionary words in each image in the training set. These histograms are added to the SVM, along with an array of tags indicating the category each histogram belongs to. Once all of the histograms have been added to the SVM, you train the SVM with the data. At this point the SVM is ready to predict input image's categories.

Testing Input

To test the input, you first need to load the input images along with their categories, into a vector of ImageStructs. This process is similar to the loading of training data, except without a need to load data for setting up the BOW & SVM.

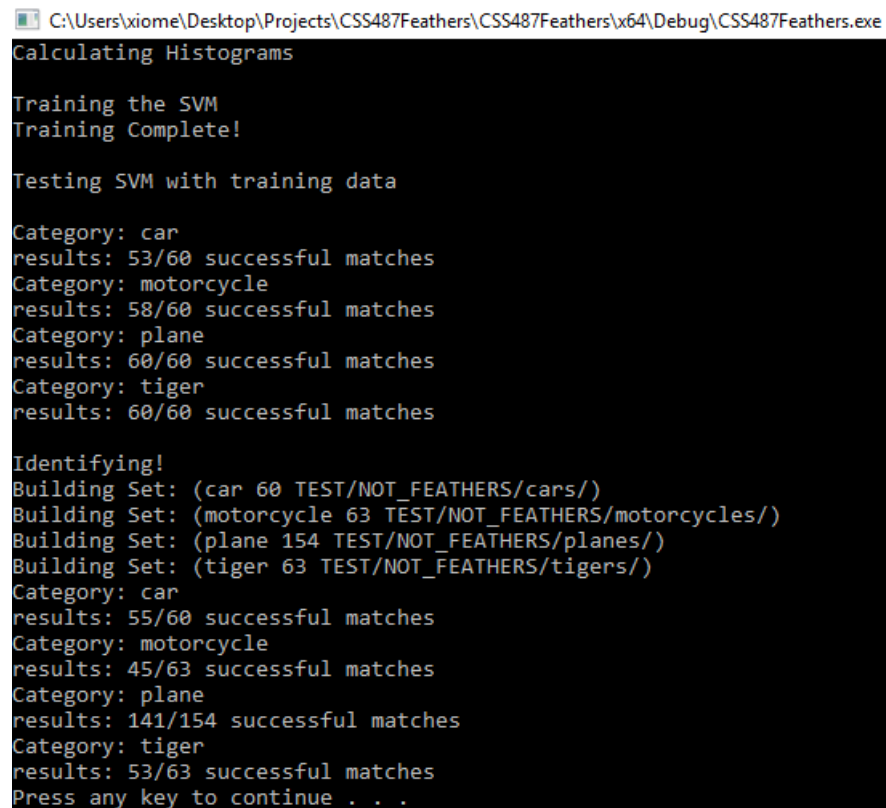
Once this data is loaded, the Identifier takes each image, and performs the same process on the testing images as for the training images, except instead of adding their histograms to the SVM as training data, it passes it into an SVM::predict() function. This function returns the SVM's best guess as to the category of the image.

Results

Using Another Dataset

The Identifier was first tested on an existing dataset consisting of cars, planes, motorcycles, and tigers to determine that it was functioning properly. Generally speaking, the Identifier seemed to work rather well. However, the motorcycle category only succeeded in being identified properly 71% of the time.

FIGURE 8: OTHER DATASET RESULTS



```
C:\Users\xiome\Desktop\Projects\CSS487Feathers\CSS487Feathers\x64\Debug\CSS487Feathers.exe
Calculating Histograms

Training the SVM
Training Complete!

Testing SVM with training data

Category: car
results: 53/60 successful matches
Category: motorcycle
results: 58/60 successful matches
Category: plane
results: 60/60 successful matches
Category: tiger
results: 60/60 successful matches

Identifying!
Building Set: (car 60 TEST/NOT_FEATHERS/cars/)
Building Set: (motorcycle 63 TEST/NOT_FEATHERS/motorcycles/)
Building Set: (plane 154 TEST/NOT_FEATHERS/planes/)
Building Set: (tiger 63 TEST/NOT_FEATHERS/tigers/)
Category: car
results: 55/60 successful matches
Category: motorcycle
results: 45/63 successful matches
Category: plane
results: 141/154 successful matches
Category: tiger
results: 53/63 successful matches
Press any key to continue . . .
```

Regardless of performance, the data succeeded in fulfilling its purpose – verifying the functionality of the algorithm.

Using Clear Images

Using a provided dataset with hundreds of images is one thing, but using a dataset of only dozens that we created ourselves is a very different situation. Each category of images we had for training had different numbers of images, and we first tried using the maximum number of images available in each category. However, this led to a terribly skewed SVM. The category with the most images had the most influence on which words were most common, and thus the Identifier tended to say every testing image was in the category with the most training images.

Once this issue was discovered, I had to limit which feathers and categories were used for training. I removed all categories that had less than 8 training images, and removed all feathers past the 8th in categories that had more. The results were much better:

FIGURE 9: VERIFICATION AND TESTING OF PLAIN-BACKGROUND FEATHERS

```
Testing SVM with training data
-----
Category: BANO
results: 8/8 successful matches
Category: COHA
results: 8/8 successful matches
Category: LEOW
results: 8/8 successful matches
Category: NOHA
results: 8/8 successful matches
Category: OSPR
results: 8/8 successful matches

Identifying!
-----

Making Testing Sets
-----
Opening testing File "resources/CLTEST.tst"
Building Set: (BANO 2 TEST/CL/BANO/)...Done!
Building Set: (COHA 6 TEST/CL/COHA/)...Done!
Building Set: (LEOW 6 TEST/CL/LEOW/)...Done!
Building Set: (NOHA 2 TEST/CL/NOHA/)...Done!
Building Set: (OSPR 2 TEST/CL/OSPR/)...Done!

Testing Data
-----
Category: BANO
results: 2/2 successful matches
Category: COHA
results: 5/6 successful matches
Category: LEOW
results: 3/6 successful matches
Category: NOHA
results: 1/2 successful matches
Category: OSPR
results: 0/2 successful matches
Press any key to continue . . . _
```

While the training data worked on its own Identifier (unlike before the equalization of category size), the results of the new image testing is not too promising. COHA and BANO seem to have good match rates, but the rest are somewhat disappointing. However, due the small number of feathers to use for testing, its hard to tell if the bad match rates are just because of the small number of feathers used, or because of the settings of the Identifier itself.

Using Images With Backgrounds

The images with backgrounds seemed to work even worse. This makes sense in retrospect, as the backgrounds create noise that can be interpreted as important words, just because the backgrounds are very noisy. This is made more severe by the fact that the backgrounds were artificially added, which made the extra information more regular. This may have the SVM test more for which background is used than which feather is used. While it did not fail completely, it did much worse. Even many of the training images failed to be properly identified in their own classifier.

FIGURE 10: VERIFICATION AND TESTING OF NOISY BACKGROUND FEATHERS

```
Testing SVM with training data
-----
Category: BANO
results: 3/20 successful matches
Category: NOHA
results: 19/20 successful matches
Category: RTHA
results: 10/20 successful matches
Category: SSHA
results: 10/20 successful matches

Identifying!
-----

Making Testing Sets
-----
Opening testing File "resources/BGTEST.tst"
Building Set: (BANO 5 TEST/BG/BANO/)...Done!
Building Set: (NOHA 5 TEST/BG/NOHA/)...Done!
Building Set: (RTHA 5 TEST/BG/RTHA/)...Done!
Building Set: (SSHA 5 TEST/BG/SSHA/)...Done!

Testing Data
-----
Category: BANO
results: 2/5 successful matches
Category: NOHA
results: 4/5 successful matches
Category: RTHA
results: 2/5 successful matches
Category: SSHA
results: 1/5 successful matches
Press any key to continue . . .
```

Further Improvements

While the Identifier program works properly at this point, there are some improvements that can be made, for both better testing, and better functionality.

1. Get more training and testing images. While the Identifier seemed to somewhat work with our data, the small number of images in each of the categories made it difficult to tell if this was because the Identifier was working properly, or just chance. These new images should be more varied as well
2. Add a feature to save and load the Identifier state. Creating the dictionary by calculating an enormous K-means algorithm, and scales terribly. If one was using this on a regular basis, the

would not want to have to remake the dictionary every time. The functions for saving and loading are easily available in OpenCV, but I did not have time to add them, and they were not necessary for the assignment, although they may have made testing faster.

3. Test with different feature extraction methods. I had originally intended to measure the difference in performance and accuracy with different extraction methods, but there was too little training and testing data to find a noticeable difference.

Conclusion

While the original goals were to see if factors such as different sizes of dictionaries and different feature extractors affected the performance of a BOW Identifier, the lessons ended up being more about how the dataset you are using makes the biggest impact on how the Identifier works.

If you don't have many training or testing images, it does not matter much which settings you use for the rest of your algorithm, as if you don't have many training images, it is difficult to create a comprehensive enough dictionary of words to encompass a majority of the features that category of feather contains.

If your images for training and testing contain repeated extraneous data, the Identifier becomes one that detects background information, instead of the information you are actually concerned about. This makes it practically useless. Perhaps with a larger training data set with more varied backgrounds, this would help, but with the data I had at hand, there was no way for me to know.

At the end of the day, if your training data is bad, it makes no difference which feature extraction method, or dictionary size you use – your BOW Identifier will not work well.