Image Classification using Bag of Features Algorithm

Written by Alexander Peralta

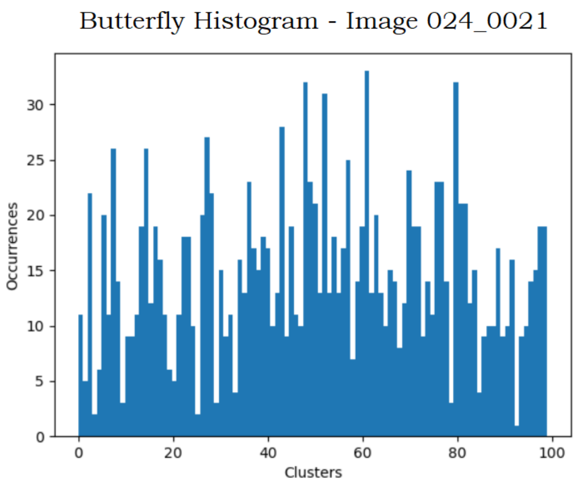
In this project a bag of features algorithm using SIFT descriptors was used for image classification. Steps taken to complete this task are broken down in the following report

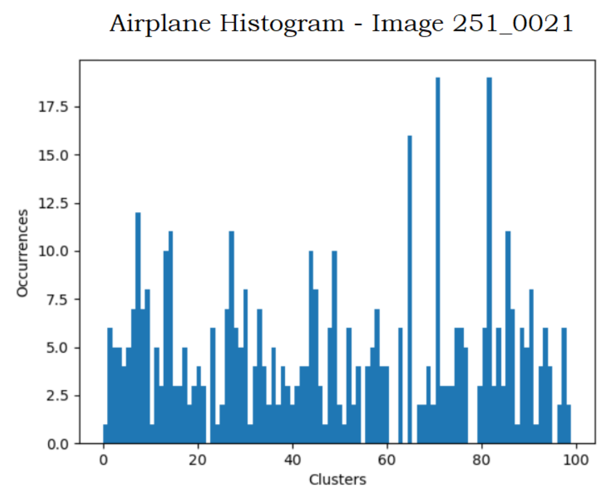
Bag of features classification uses a training set and a testing set of images. In this project, there were three different classifications of images used, a butterfly, a hat, and an airplane. The training set had 157 total images - 50 butterfly images, 50 hat images, and 57 airplane images. The testing set had 36 total images - 10 butterfly images, 10 hat images, and 16 airplane images. These numbers and ordering play an important role in keeping track of the images and feature descriptors for each image.

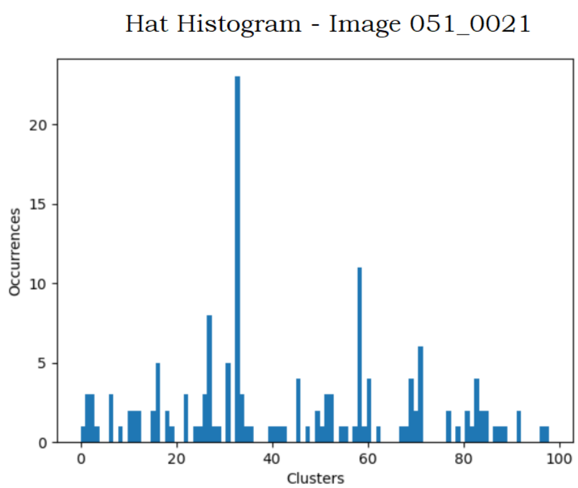
For the training set images, a for loop was used to iterate over the training set data and process the images. The 50 butterfly images were processed first, followed by the 50 hat images, followed by the 57 airplane images. In order to find the SIFT feature descriptors and keep Thtrack of which image they corresponded to, an 2D array and a list were used. The array named trainingSetDescriptors was used to store the entire collection of SIFT features for all the images. The list named numDescriptorsTrainingImages was used to keep track which sift features belonged to which image. The SIFT feature description for each image I was found using [f, d] = sift.detectAndCompute(I,None). The descriptors, d, were added to trainingSetDescriptors , using np.vstack(). The number of descriptors added was found using numDescriptors, dimension = d.shape , and this is kept track of by appending numDescriptors to numDescriptorsTrainingImages. An theoretical example to illustrate this: the first image processed has 168 descriptors that are added to trainingSetDescriptors . The second image processed has 200 descriptors that are added after the first 168 descriptors to trainingSetDescriptors . This is kept track of in numDescriptorsTrainingImages , where arr[0] = 168, and arr[1] = 200. Descriptors in image n can be found in using numDescriptorsTrainingImages [n-1].

The training set image descriptors in the trainingSetDescriptors array were clustered using sklearn.cluster.Kmeans. One hundred clusters were created. The output of Kmeans was assigned to kmns. kmns.labels_ is an array of the same length as trainingSetDescriptors, where its value at each index is the equal to the cluster corresponding to the descriptor at the index in trainingSetDescriptors. kmns.labels_ has values from 0-99 inclusive. This is used to create the histograms.

To create a histogram for each image a for loop was used to iterate over the numDesciptorsTrainingImages list. Using a counter starting from 0 and the fact that numDesciptorsTrainingImages has the number of descriptors for each image, kmns.labels_ was able to be iterated through, breaking up each descriptor section that corresponds to each image. This broken up section of the array was used to create a histogram with 100 bins for each individual image, using np.histogram. 100 bins were used to match the 100 clusters. The histograms were normalized by setting density=True inside the np.histogram function. The output is a (1x100) histogram vector that represents the image The histograms were stored in a 2D array named trainingDataHistgrams. A visual output of a histogram from an image in each classification is included on the following page.







This exact same process was completed for the testing data set, except instead of using sklearn.cluster.Kmeans of the testing data feature descriptors, kmns.predict() was used. This was done because kmns.predict() uses the clusters created from the training data. It finds which of these clusters each descriptor in the testing data set is closest to. The output of kmns.predict() on the testing data set descriptors is the same format as the output for kmns.labels_ on the training data set descriptors, so the same process of finding the histograms is used.

Classifications for the test images were conducted using three different methods. First was a K nearest neighbors classification with k=1. An imageType array was created with the first 50 entries being 0's (butterfly), the next 50 being 1's (hat), and the last 57 being 2's (airplane). The function sklearn.neighbors.KNeighborsClassifier.fit was with the trainingDataHistogram and imageType arrays. It takes these two inputs in order to match up histograms with the type of image. The function sklearn.neighbors.KNeighborsClassifier.predict was with the testDataHistogram and the output is an array of length 36 where each index corresponds to the number of test histograms/images and the predicted value is 0(butterfly), 1(hat), or 2(airplane).

The next two classification methods follow the exact same logic but use different processes in creating their predictions. LinearSVC() method creates linear boundary lines between training data histogram points that represent butterfly, hat, and airplane. The function svm.SVC() by default uses a kernel with a radial basis function to create boundary lines between training data histogram points of like kind. Confusion matrices are included below showing the results. Both the K nearest neighbor and linear SVM methods had difficulty predicting the hat images, only correctly predicting a hat image as a hat image 50% of the time. The RBF kernel SVM method showed improvement, correctly predicting a hat image as a hat image 80% of the time. All three methods were successful at correctly predicting butterfly and airplane images, with at least an 80% accuracy. These results show that using a RBF kernel SVM produces the most accurate results.

K Nearest Neighbors

Actual Classes	Predicted Butterfly	Hat	Airplane
Butterfly	100%	0%	0%
Hat	20%	50%	30%
Airplane	18.75%	0%	81.75%

Linear SVM

Actual Classes	Predicted Butterfly	Hat	Airplane
Butterfly	80%	0%	20%
Hat	20%	50%	30%
Airplane	0%	6.25%	93.75%

Radial Basis Function Kernel SVM

Actual Classes	Predicted Butterfly	Hat	Airplane
Butterfly	90%	0%	10%
Hat	10%	80%	10%
Airplane	0%	6.25%	93.75%

Potential improvements could be found by using principle component analysis on the SIFT feature descriptors before clustering. These descriptors were 128 dimensional vectors. By reducing the dimension of the vectors to the principal components, it could have created more distinction between the descriptors of different classes which could have allowed for better clustering.