Project 2 Report

SIFT Features

We first preprocess each training image by converting it to grayscale and then applies the SIFT algorithm to detect key points and extract descriptors. The detected keypoints and descriptors are stored in corresponding lists, and we assign a label to each image based on its filename prefix. As this process occurs within the same iteration, the features lists are aligned with the training class.

Figure 1: Gray-scale Image of a butterfly with key points

Figure 2: Gray-scale Image of a cowboy hat with key points

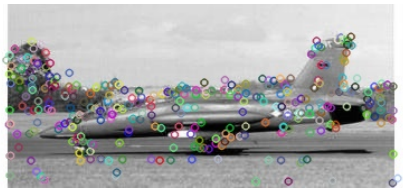

Figure 3: Gray-scale Image of an airplane with key points

The figures above demonstrate the key points obtained from the SIFT algorithm overlaid on top of the grayscale images. These key points are generally located near edges, indicating that SIFT detects them based on changes in intensity values, which are often found near edges. For each key point, there is a 128-dimensional vector that captures the local appearance of the image region surrounding the feature. As such, the descriptor is robust to changes in viewpoint and illumination.

Clustering

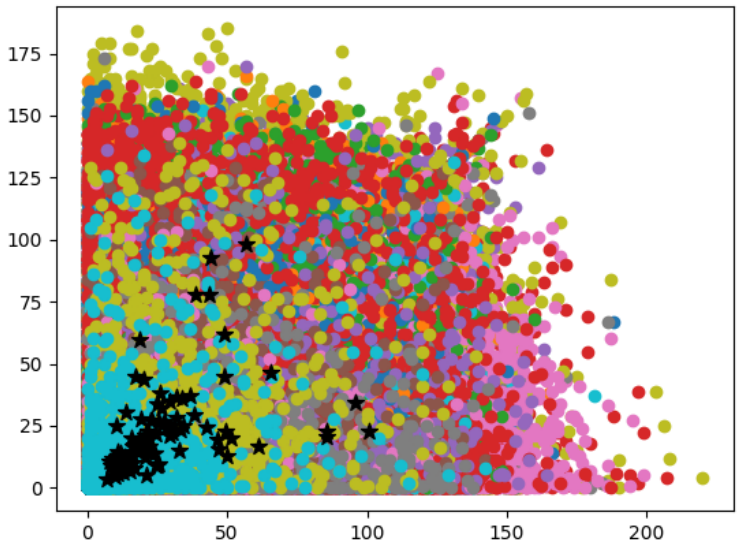

Figure 4: Visualization of K-Means clustering on Training Data Set (centroids are indicated by black stars)

Next, we establish a vocabulary of visual words by performing k-means clustering on the list of SIFT feature descriptors extracted previously. Since k-means clustering expects a 2D array of samples, the descriptors are first concatenated into a single array, where each row represents a single feature descriptor.

We initialize a k-means classifier with parameters `n_clusters=100`, `init='k-means++'`, `max_iter=100`, `n_init=1`, and `random_state=0`. This classifier will partition the descriptors into 100 distinct clusters. Furthermore, the algorithm will use a smart initialization method to speed up convergence since `init='k-means++'`. The `max_iter` parameter sets the maximum number of iterations that k-means will run before stopping while the `n_init` parameter sets the number of times the algorithm will run. The `random_state` parameter is set to 0, which ensures that the clustering is reproducible across different runs of the code. After experimenting with various `max_iter` and `n_init` values, I found that `max_iter=100` and `n_init=1` provided a good balance between run time and accuracy. For example, with `max_iter=1` and `n_init=1`, the run time is ≈ 10 seconds, while for larger values such as `max_iter=100` and `n_init=5`, the run time is much longer (i.e. ≈ 60 seconds) with minimal increase in overall testing accuracy. With `max_iter=100` and `n_init=1`, the run time is still minimal (≈ 13 seconds) and the overall testing accuracy is acceptable. However, the parameters can be more carefully tuned to obtain more optimal outcomes. This would involve trying different combinations of parameters and evaluating their performance. For example, we could use a validation set to evaluate the performance of different parameter values and find the one that does the best.

After initializing the classifier, we fit to it the array of concatenated descriptors. The output of the k-means clustering algorithm is then a set of 100 cluster centers (i.e. visual words), and a set of labels that indicate which cluster each descriptor belongs to. These labels are assigned to the corresponding images in the training set, creating a bag of visual words representation for each image. We see the visualization of this in Figure 4. We see that the cluster centers are close together, suggesting that features belonging to different clusters are similar to each other. This can potentially lead to poor performance because the visual words are not very distinct.

Histograms

The images can now be represented as histograms based on the frequency of visual words present in them. By using these image histograms, we can determine which images are similar by using them as features for training classifiers. To create the histogram for each training image, we use the descriptors and cluster centers found previously. Initially, we create a histogram of zeros, and for each descriptor, we find the nearest centroid based on the Euclidean distance. We increment the corresponding bin in the histogram for that centroid. Finally, we normalize the histogram by dividing each bin value by the total number of SIFT features binned for the image.

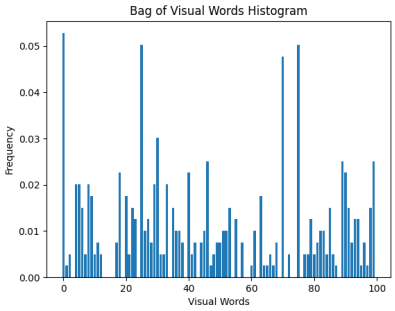

Figure 5: Histogram of Figure 1

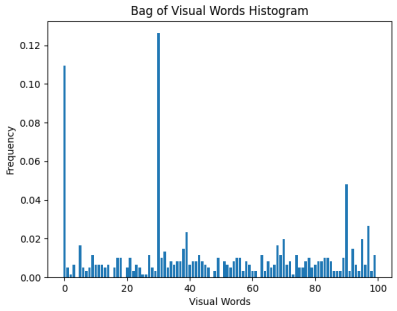

Figure 6: Histogram of Figure 2

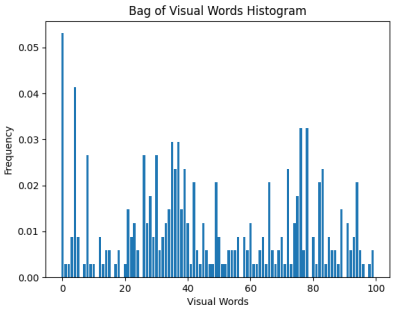

Figure 7: Histogram of Figure 3

The figures depicted above illustrate the formation of descriptors for images using normalized histograms. The x-axis denotes the 100 visual words, while the y-axis represents the frequency at which these words occurred in the image. These figures gives a rough idea of the distribution of visual words in each class of image.

As previously mentioned, there is a high degree of similarity between features assigned to different clusters. Consequently, there may be considerable ambiguity in assigning the descriptors. Some points of possible improvement would be to incorporate term frequency-inverse document frequency (TF-IDF) calculations to weight the histograms. TF-IDF involves computing a term frequency (TF) factor, which represents how frequently a word occurs within an

image, and an inverse document frequency (IDF) factor, which reflects how frequently a word appears across the entire dataset. By using TF-IDF, the histogram can be weighted such that visual words that occur frequently across all images are down-weighted, while rarer words are up-weighted. Reference

Classification Preparation

Similar to the preparation of training images, we perform analogous steps to prepare the test images for classification. This involves identifying SIFT feature descriptors, assigning the descriptors to appropriate clusters, and using these assignments to generate normalized histograms.

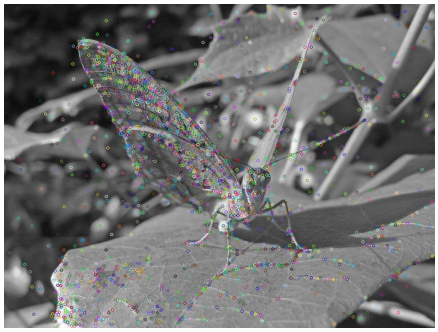

Figure 8: Gray-scale Image of a butterfly with key points

Figure 9: Gray-scale Image of a cowboy hat with key points

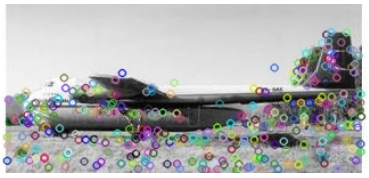

Figure 10: Gray-scale Image of an airplane with key points

The above figures show some similarities to Figures 1, 2, and 3. The key points are still concentrated near edges and distinctive features. However, it is apparent that the SIFT algorithm faces difficulty in handling cluttered backgrounds. Specifically, the butterfly and airplane images exhibit a background with distinctive features, resulting in key points that may not be relevant in determining the image class.

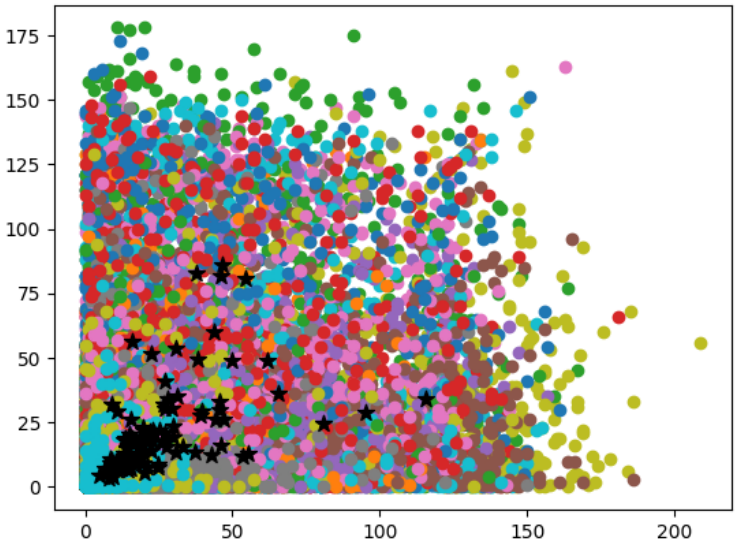

Figure 11: Visualization of K-Means clustering on Testing Data Set (centroids are indicated by black stars)

The predicted labels for the testing dataset are from the k-means clustering model trained on the training dataset. We observe that the resulting visualization is very similar to the one shown in Figure 4.

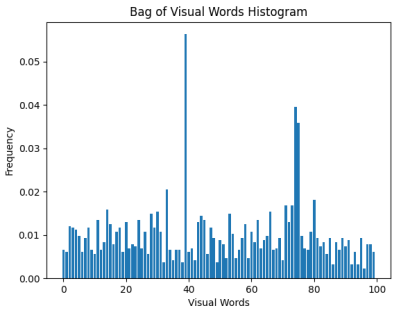

Figure 12: Histogram of Figure 8

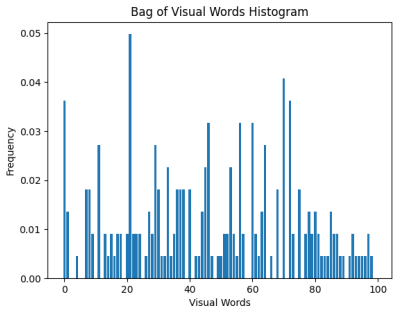

Figure 13: Histogram of Figure 9

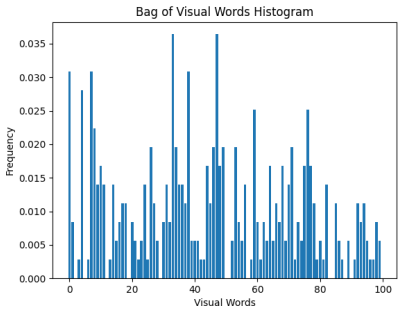

Figure 14: Histogram of Figure 10

The histograms show variations in the frequency of visual words in different images. For instance, Figure 8 has some visual words that occur more frequently compared to others, while Figures 9 and 10 have similar frequencies for most words. These differences highlight the necessity of weighting the histograms.

Classification

We will now train different classifiers (i.e. KNN, SVM, and Kernel SVM) on the previously calculated histograms in order to predict the labels of the test images.

K = 1 Nearest Neighbor

The nearest neighbor classifier finds the closest training example and assigns the test image its class label. We create the classifier using K-nearest neighbors (KNN) algorithm with a single neighbor. The histograms and their corresponding labels are then used to fit the KNN classifier. The training time for this classifier is approximately 0.000998 seconds, which is the shortest time among the three classifiers. Once the classifier has been trained, it is used to predict the labels of the test histograms. The resulting confusion matrix for this classifier is presented below.

		Predicted Class		
	Classes	Butterfly	Hat	Airplane
Actual class	Butterfly	100%	0%	0%
	Hat	30%	50%	20%
	Airplane	31.25%	0%	68.75%

The classifier demonstrated perfect prediction accuracy for butterfly images and performed well for airplane images. However, only 50% of hat images were classified correctly. As a result, the overall accuracy was $\approx 69.4\%$.

Linear Support Vector Machine

Next, we train and test a support vector machine (SVM). We create the SVM classifier with a linear kernel. The histograms and their corresponding labels are then used to fit the classifier. The training time for this classifier is approximately 0.004017 seconds, which is the longest time among the three classifiers. Once the classifier has been trained, it is used to predict the labels of the test histograms. The resulting confusion matrix for this classifier is presented below.

		Predicted Class		
	Classes	Butterfly	Hat	Airplane
Actual class	Butterfly	0%	0%	100%
	Hat	0%	10%	90%
	Airplane	0%	0%	100%

The classifier demonstrated perfect prediction accuracy for airplane images but performed very poorly for both butterfly and hat images. In fact, all of the butterfly images were wrongly classified as airplanes. As a result, the

overall accuracy was only $\approx 47.2\%$.

Kernel Support Vector Machine

Lastly, we train and test a SVM classifier with a radial basis function (RBF) kernel. We initialize the SVM classifier with a RBF kernel. The histograms and their corresponding labels are then used to fit the classifier. The training time for this classifier is approximately 0.002994 seconds, which is faster than the linear SVM but still much slower than the nearest neighbor. Once the classifier has been trained, it is used to predict the labels of the test histograms. The resulting confusion matrix for this classifier is presented below.

		Predicted Class		
	Classes	Butterfly	Hat	Airplane
Actual class	Butterfly	90%	0%	10%
	Hat	10%	60%	30%
	Airplane	0%	6.25%	93.75%

The classifier performed very well for most of the classes, achieving over 90% accuracy for both butterfly and airplane images. However, it still struggles with correcly identifying hats. As a result, the overall accuracy was the highest at $\approx 83.3\%$.

Extra Credit: Visualizing the Data

We create a 3D plot that visualizes the histograms projected onto the first three principal components. First, we reduce the dimensionality of the histograms from 100-dimensions to 3-dimensions using principal component analysis (PCA). To do so, we create a PCA object and set the number of components to 3. Then, we fit the object with the training histograms, which gives us the eigenvectors of the covariance matrix of the histograms and sorts them in descending order based on their corresponding eigenvalues. Next, we obtain the transformed histograms by taking the dot product of the histograms with the transpose of the first three principal components. Finally, we use the transformed histograms to create the 3D plot, with the first three principal components as the x, y, and z axes.

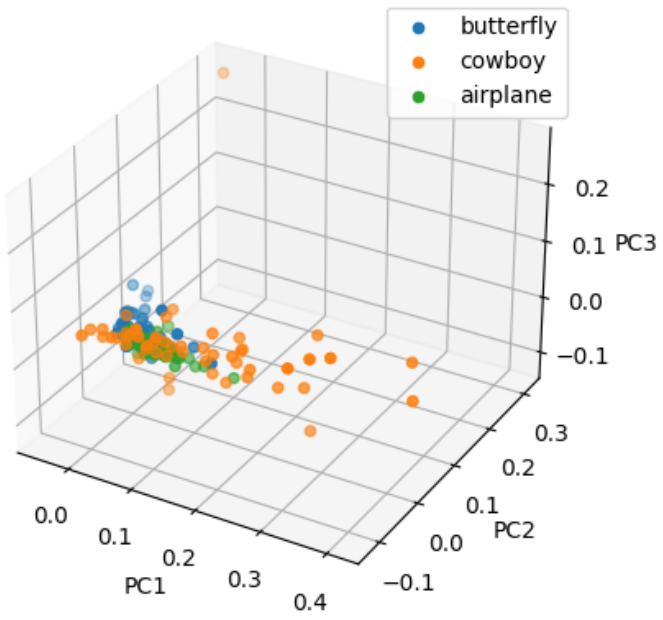

Figure 15: 3D visualization of histograms associated with the training dataset

Figure 15's visualization of the histograms enables us to analyze the separability of the different classes and detect patterns and clusters. It is evident from the plot that the visual words are closely located to each other. Nonetheless, the cowboy hat histograms are found to have considerably more outliers, indicating that the hats possess distinctive features that do not appear consistently in other classes or even among themselves. This observation could potentially explain why all of the trained classifiers struggle to accurately predict the labels for cowboy hats. One possible solution to overcome this challenge is to increase the size of the dataset, as this may help to include more diverse features of the hats.