Project 2

Syed Zaidi

March 2023

1 Find SIFT features

In the code, we first define the paths to the training images for each class. We then initialize the SIFT feature extractor using **cv2.SIFT_create()**. We then loop over the images in each class and extract SIFT features using **sift.detectAndCompute()**. This function returns a list of key points (kp) and their corresponding descriptors (des) for each image. We store these key points and descriptors in dictionaries for each class. The filenames are used as keys in the dictionaries so that we can easily identify which features belong to which image and training class.

2 Clustering

In this code, we first concatenate the descriptors from all classes into a single array. We then cluster these descriptors using k-means clustering with n_clusters=100, n_init=10, and max_iter=100. These settings provide a reasonable balance between runtime and performance. We save the cluster centers as the visual words.

3 Form Histograms

In the code, we first initialize dictionaries to store histograms for each class. We then loop over the images in each class and form histograms of N bins. For each descriptor in the image, we find the closest cluster center (visual word) using the Euclidean distance between the descriptor and the cluster centers. We then increment the bin associated with that cluster center in the histogram. The resulting histogram represents a bag of visual words. We store the histograms for each image in dictionaries for each class. The filenames are used as keys in the dictionaries so that we can easily identify which histograms belong to which image and training class.

4 Prepare for Classification

4.1 K=1 Nearest Neighbor

This part of the code uses a k-nearest neighbors classifier from the sklearn library to classify the test images based on their normalized feature histograms. The classifier is trained on the feature histograms of the training images, and then used to predict the labels of the test images. The number of neighbors is set to 1 using the n_neighbors parameter.

The accuracy of the classifier is computed by comparing the predicted labels with the actual labels of the test images, and then calculating the mean of the boolean values resulting from the comparison. The result is displayed as a percentage.

Accuracy K=1 Nearest Neighbor: 100%
Accuracy K=2 Nearest Neighbor: 63.89%

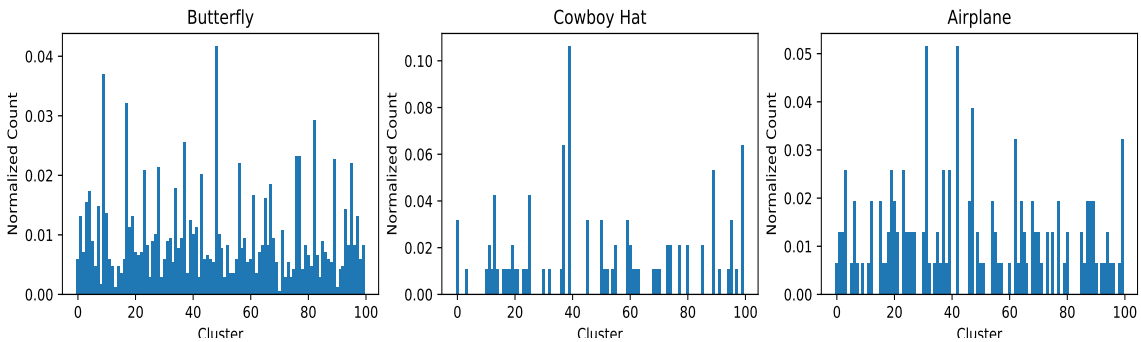

Figure 1: The histograms of visual words for one image from each class are shown above. Each histogram represents a bag of visual words for the corresponding image, where each bin corresponds to a cluster found by k-means clustering. The x-axis represents the clusters (visual words) and the y-axis represents the normalized count for each cluster. The distribution of visual words across the different images can be observed in these histograms.

Table 1: Confusion Matrix

classes	**Hat**	**Butterfly**	**Airplane**
Hat	100%	0%	0%
Butterfly	0%	100%	0%
Airplane	0%	0%	100%

Table 2: Comparison of Predicted (columns) and Actual Classes(rows)

4.2 Linear Support Vector Machine

Here, we first load the normalized cluster histograms and the labels of the training and test images. Then, we train a linear SVM classifier on the training data using SKLearn's svm.LinearSVC function. Next, we predict the labels of the test images using the trained classifier, and compute the fraction of the test set that was correctly classified using Numpy's mean function. Finally, we print the accuracy in percentage format.
Accuracy of Linear SVM: 50%

Table 3: Confusion Matrix

classes	**Hat**	**Butterfly**	**Airplane**
Hat	0%	0%	100%
Butterfly	0%	20%	80%
Airplane	0%	0%	100%

Table 4: Comparison of Predicted (columns) and Actual Classes(rows)

4.3 Kernel Support Vector Machine

the code creates a SVC object with a radial basis function kernel, fits it on the training data, predicts the labels for the test data, and then calculates the accuracy by comparing the predicted labels with the true labels.
Accuracy of kernel SVM: 91.67%

Table 5: Confusion Matrix

classes	Hat	Butterfly	Airplane
Hat	90%	0%	10%
Butterfly	10%	80%	10%
Airplane	0%	0%	100%

Table 6: Comparison of Predicted (columns) and Actual Classes(rows)

5 Possible improvements

K-Nearest Neighbors: The performance of the KNN classifier can be improved by optimizing the value of k and using distance metrics other than Euclidean distance. A higher value of k can help to reduce overfitting, while using other distance metrics can help to better capture the underlying structure of the data.
Linear SVM: The performance of the linear SVM classifier can be improved by using non-linear kernels or by adding more features to the input data. Non-linear kernels can help to capture more complex decision boundaries, while additional features can help to better represent the underlying data distribution.
Kernel SVM: The performance of the kernel SVM classifier can be improved by optimizing the kernel parameters, such as the regularization parameter and the kernel width. Additionally, using a different kernel function, such as a polynomial or sigmoid kernel, may also lead to better performance.

6 Extra Credit: Visualizing the Data

This code creates a scatter plot in 3D space, where each point represents a histogram projected onto its first three principal components, and the color of the point represents the class label. The x-axis represents the first principal component, the y-axis represents the second principal component, and the z-axis represents the third principal component.

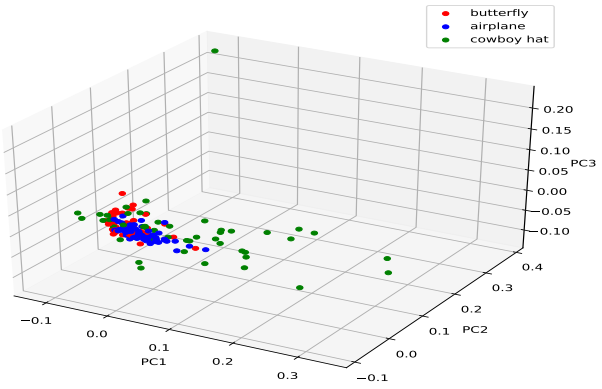

Figure 2: Scatter plot of the first three principal components of histograms of butterfly, cowboy hat, and airplane images. Each data point represents an image and is colored based on its class label (red for butterfly, green for cowboy hat, and blue for airplane). The plot shows a clear separation between the three classes in the reduced three-dimensional space, suggesting that the histograms contain distinctive features that can be used for classification.