Due: 8th March 2023 3:30 PM

In this project you need to classify images. You will learn a classifier from the training data provided in `Project_02.zip`, then use it to classify test images, as described in Section 2.

The provided dataset includes both training and test data. The first 3 digits of each image's file name provides its label: 024 denotes butterflies, 051 denotes cowboy hats, and 251 denotes airplanes.

As always, you will document your work and show your results in a pdf report.

1 Restricted Functions

You may import numpy, matplotlib, PIL, sklearn, cv2 (with contrib), os, and time. Any and all functions in these packages are fair game. Please do not import additional packages.

2 Bag of Features Classification with SIFT Descriptors

A *Bag of Features* algorithm uses image features for classification[1]. It operates under the key assumption that the presence or absence of certain features within an image indicate the class of the image. For instance, the presence of "tire" features would indicate membership of the "vehicle" class. SIFT[2] features are in general robust to a wide range of non-ideal imaging situations, and therefore are what you will use for your implementation. This means you will need to find SIFT features and compute their associated feature descriptors for a large pool of images. You can use OpenCV-Python to detect and compute SIFT features.
You will need to install opencv-contrib to have access to SIFT. The easiest route to do this is using `pip install opencv-contrib-python`.
After instantiating the SIFT feature detector with `sift = cv2.SIFT_create()`, SIFT features and descriptors can be found for a uint8 grayscale image `I` using
`[f,d] = sift.detectAndCompute(I,None)`. The output `f` stores the location, size, and orientation of the keypoints while `d` stores the actual (128×1) feature descriptors.

[1] Also sometimes referred to as a *Bag of Words* algorithm, from the techniques by the same name used to classify written documents.

[2] D. G. Lowe. Distinctive image features from scale-invariant keypoints. IJCV, 60(2):91–110, 2004.

3 Instructions

Complete the following steps to build your own bag of features classification algorithm.

3.1 Find SIFT features

10 points

Use SIFT to find features (and their descriptors) in all of the training set images. Do this carefully, so that you can easily identify which features belong to which image and training class.

3.2 Clustering

15 points

Cluster all the SIFT feature descriptors you found using the k-means clustering algorithm. With three image classes, $N = 100$ clusters provide a reasonable balance between runtime and performance. These clusters represent your visual words and, conceptually, you can imagine that each cluster represents the presence of certain distinct "things" in your images. `KMeans` has two optional arguments `n_init` and `max_iter` which can be reduced from their default values to speed up the clustering process.

3.3 Form Histograms

15 points

1. For each training image, form a histogram of N bins, where each bin corresponds to a cluster found above. When forming an image's histogram, the bin associated with a cluster j is incremented every time one of the image's SIFT features falls closest to cluster j. The resulting histogram represents a bag of visual words.

2. Normalize the bin counts by dividing by the total number of SIFT features binned for that particular image. This normalized histogram now forms a descriptor for that particular training image.

Please include visualizations of a few of these histograms in your report.

3.4 Prepare for Classification

5 points

1. Find the SIFT feature descriptors within each test image.

2. Assign these features to clusters as you did when creating the class descriptors.

3. Use these assignments to calculate a normalized cluster histogram for each test image.

3.5 Classification

3.5.1 K=1 Nearest Neighbor

10 points

Assign each test image to one of the possible classes by comparing its cluster histogram to the cluster histograms of the various images you trained with previously. One conceptual way to do this comparison is to think of each normalized cluster histogram as a point in an N dimensional space. Finding the class is akin to finding the class associated with the image whose histogram vector is closest to the one associated with image being tested. This task can be implemented using SKLearns's `KNeighborsClassifier` function.
Compute and report the fraction of the test set that was correctly classified. If implemented correctly, you should expect an accuracy over 60%.

3.6 Linear Support Vector Machine

10 points

Use a linear support vector machine to classify the data, again using the normalized histograms. To do this, you will first need to learn linear decision boundaries (in the normalized histogram space) using the training set. Once you have determined these boundaries, you can use them to classify the test data. This task can be implemented using SKLearns's `svm` function.
Compute and report the fraction of the test set that was correctly classified. If implemented correctly, you should expect an accuracy over 40%.

3.7 Kernel Support Vector Machine

10 points

The linear SVM's performance may be disappointing. This is because the normalized feature histograms are not linearly separable.[3] We will overcome this issue by (implicitly) lifting the data, using the kernel trick, to a higher dimensional space where it can be linearly separated.

Use a kernel SVM with a radial basis function kernel to classify the data, again using the normalized histograms. You will again first need to learn decision boundaries using the training set and then use these to classify the test data. Kernel SVMs are also suppored by SKLearns's `svm` function.
Compute and report the fraction of the test set that was correctly classified. If implemented correctly, you should expect an accuracy over 80%.

3.8 Technical Write-up: Results and Discussion

25 points

[3] Last year some students reported better-than-expected linear SVM performance.

- Clearly and cogently document your methods and results. From your PDF report, it should be clear what you did, how/why you did it, and how well it worked, without needing to run code or sift through 300 figures.
- Indicate points of possible improvement and provide conceptual solutions to the extent you are able.
- Include and interpret three distinct (3×3) confusion matrices (one for each classifier). See below for an example:

		Predicted class		
	Classes	Hat	Butterfly	Airplane
Actual class	Hat	55%	31%	14%
	Butterfly	20%	70%	10%
	Airplane	12%	5%	83%

4 Extra Credit: Visualizing the Data

15 points

The SVM classifiers are implicitly classifying the data by placing hyperplanes in an $N(=100)$ (linear SVM) or infinite (kernel SVM) dimensional space. This is obviously difficult to visualize.

To help visualize the decision process, create a plot that projects the histograms associated with the training data onto its first three principal directions. Color code each point by its class. Your plot that looks something like Figure 1.

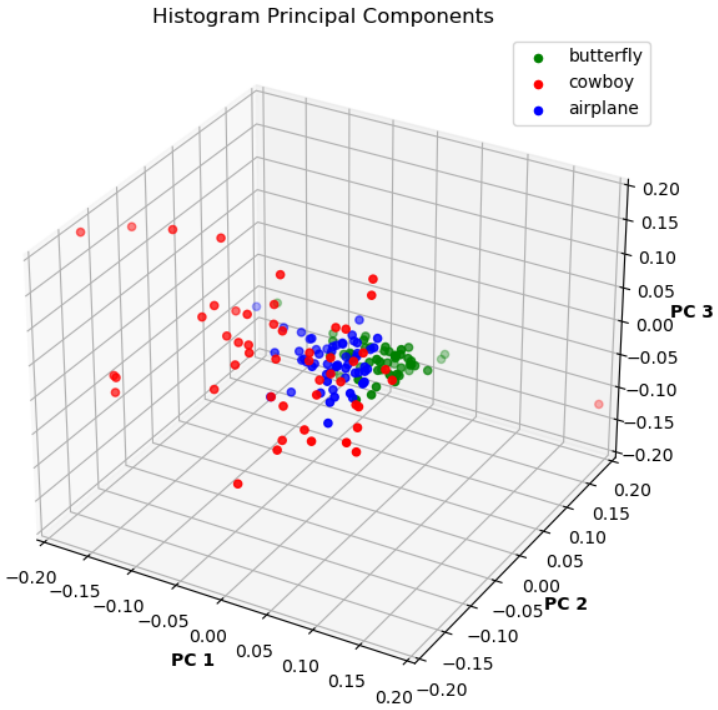

Figure 1: 3D dimensional visualization of the data

Submission Instructions

There are 2 submission portals on Gradescope. You need to submit the PDF report to the submission portal named "Project 2 PDF" and also submit a ZIP file to another portal named "Project 2 Code". Your files should be named **YourDirectoryID_Report.pdf** and **YourDirectoryID_Project1.zip**, respectively.

Your zip file should contain:

- Project2.py.

Do not include the datasets in your submission. Rather use relative pathing assuming the code and Project2_data directory are in the same parent directory.
E.g., load your training data from "`./Project2_data/TrainingDataset/`".

Only one person per group needs to submit the assignment, but be sure to submit as a group assignment.

Collaboration Policy

You may (optionally) work in groups of up to two people. You are encouraged to discuss ideas with peers outside your group. However, the code should be your group's own and should represent your understanding of the assignment. Code should not be shared or copied. If you reference anyone else's code in writing your project, you must properly cite it in your code (in comments) and in your report.

Plagiarism

Plagiarism of any form will not be tolerated. You are expected to credit all sources explicitly. If you have any doubts regarding what is and is not plagiarism, talk to me.

Credit

Thanks to Ashok Veeraraghavan and Mohammad Teli for sharing their course resources.