Problem 1- Create a panorama of images (3 images) using Detect, Describe, Match and wrap.

(In the first submission I forgot to add code file for this problem, so I had done another submission next day after deadline.)

Approach:

The idea is to deal with two images at a time and the result of that will be stitched with the third image.

First of all find key points (feature points) in both the images using **SIFT** or **SURF** feature extractor as they are **invariant to scale and transformations**. And than we need to match this both images feature points. To do so we can use cv2.FLANN matcher or cv2.BFMatcher. Then as there are **many unwanted feature points** in the images, we need to remove such feature points. So we **apply ratio test** using the top 2 matches obtained above. After this we need to **map corresponding** features of both images, and to do so we will have to calculate **homography matrix**. To calculate homography matrix we can use cv2.findHomography() function. And finally using this calculated homography matrix, we will wrap the source image into destination image.

Algorithm:

Step 1: Load images from right to left ordering and convert into gray scale.

Step 2: First stitch the left image and middle image together. So using SIFT, find interest points and descriptors for both image.

Step 3: The obtained descriptors in one image are to be recognized in the other image. To do that we have 2 options: FLANN matcher or BFMatcher. As we are dealing with only 3 images, we can go with BFMatcher (it is Brute force and more time consuming). BFMatcher takes the descriptor of one feature in first set and is matched with all other features in second set using some distance calculation. And the closest one is returned.

Step 4: The match returned by above step has many unwanted matches. So to deal with it filter out good matches. Apply ratio test using the top 2 matches obtained above

Step 5: Now calculate homography matrix using cv2.FindHomography().

Step 6: Using homography matrix, warp and stitch both images together.

Step 7: Now considering this output image as left image and last image as right, repeat from step 2 to step 6.

NOTE:- When we are stitching two images, we are like increasing size of smaller image and then stitching with bigger one. So in the output of stitched image sometime there remains black portions in boundary. So to remove that I have defined a function called **trimBlack()** which takes image as input and trims all black portion surrounding that image.

Input images:

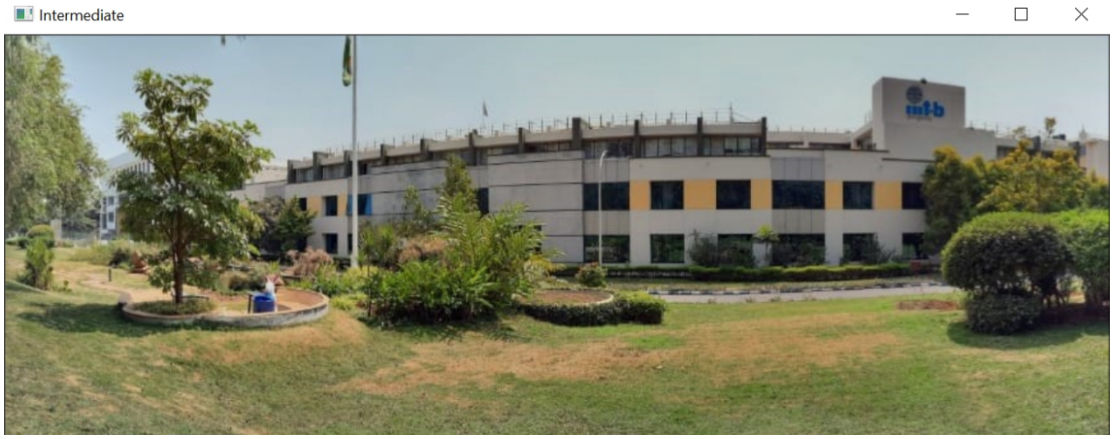
(Left)

(Middle)

(Right)

Output:

(Output by stitching left and middle image)



(Final output)

Difference between SURF and SIFT.

Major difference between SURF and SIFT is, SURF returns 64 dimensional feature vector whereas SIFT returns 128 dimensional feature vector i.e double information compared to SURF.

Other thing is SURF is claimed to be several times faster than SIFT and more robust against different image transformations compared to SIFT.

Briefly explain the main principles of FLANN matching and RANSAC.

FLANN stands for Fast Library for Approximate Nearest Neighbors. It contains a collection of algorithms optimized for fast nearest neighbor search in large datasets and for high dimensional features.

While performing matching we take one feature from set S1 and look for nearest neighbor in the set S2 and using Flann we can find nearest neighbors optimally and faster .

RANSAC algorithm is a learning technique to estimate parameters of a model by random sampling of observed data. The main principle of RANSAC is to find out optimal fitting result given a dataset whose data elements contain both inliers and outliers.

Problem-2: Implement Bag-of words and VLAD approach to do classification of CIFAR 10 dataset.

Solution:

About data set: CIFAR-10 contains 60,000 images belonging to 10 classes. And size of each image is 32*32*3 . This complete dataset is divided into Train and Test data, with train size as 50000 and test as 10000. And each class contains 5000 images in Train data and 1000 images in test data.

Approach:

The idea is similar to that of NLP where we develop bag of words. But here rather than words, we will develop **Bag-of-Visual Words and using that BOVW encoding each image in a vector which consists of count of occurrences of features in that images.**

Vocabulary:

So our vocabulary will be consisting of this visual words, like in an image how many nose are there, eyes are there and so on, **that is for each image there will be a vector which maintains count of such feature** and we will feed this vector to any multi class classification algorithm like SVM.

Developing vocabulary:

To develop visual words, we will use SIFT like interest point extractors which will return descriptors. And using this descriptors we will club all liked descriptors together(using clustering) where size of **k** will define the s**ize of vocabulary words**.

Pipeline:

Load images class wise --> Extract interest points --> Apply clustering on descriptors list and create vocabulary --> Encode each image in a vector --> Classification Algo --> Test accuracy.

Problem in this approach: As we know SIFT will return **(number of points * 128)** descriptors and this number of points will totally depend on the image. For example for some image there will be 0 interest points and for other there can be 1000 interest

points, hence we need to handle this **dimension mismatch problem.** And to do so, I have stacked all the feature points on top of each other, like depth of the stack will be 128 and size will be **(total no of descriptor * 128). And at the time of accessing this image wise descriptors, we will use an offset.**

Initial run:

To do initial testing, I have used only 3 class that are "Aeroplane, Bird and Automobile" as train images. And our vocabulary size as 20 (K=20).

Algorithm [Training code]:

Step 1: [Read training images]

Create dictionary of input images as ,

key: Class_label and **Value: List of images of that class.** And also create a dictionary mapping label name to an integer.

Step 2: [Calculate interest points in each images.]

Using SIFT, calculate feature points in each images and if there are descriptors in that image than **append in global list of descriptors.**

Step 3: [Now stack all the descriptors.]

Stack all the descriptors using numpy's vstack() function. The shape of this stack will be **#descriptor * 128.**

Step 4: [Do k-means clustering.]

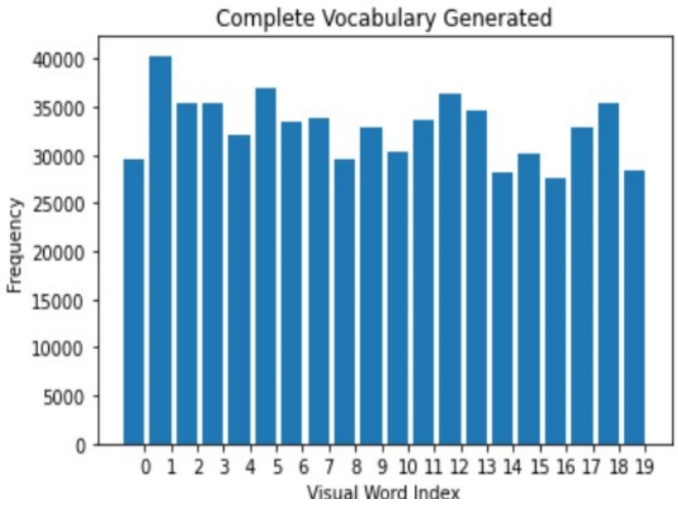
Using K as 20, apply k-means clustering.For this I have used kmeans() from sklearn.cluster class.

Step 5: [Develop vocabulary and encode images.]

Length of vocabulary will be equal to K and and each image will be encoded such that it will be contain number of occurrence of each word from vocab. Here I have **used offset to access** the descriptor belonging to that image as i have stacked all descriptors together.

(Here is the plot of vocabulary)

```
Plotting histogram
[29586 40332 35432 35364 32158 37010 33535 33853 29650 32920 30276 33564
 36389 34661 28222 30241 27562 32891 35437 28394]
```



Step 6: [Apply standard scaler to normalize the data.]

Step 7: [Train SVM on this data]

I have used svc.svm class with kernel as 'rgf'. This will return trained model object that will be used in predictions.

Testing part:

To do testing, I had done same steps till step 5 and than I had used predict method of SVM to finally predict the outputs. To do testing I have used some random 5 images from each class.

Testing Algorithm:

Step-1: [Load test data**]**

Create dictionary of test data.

Step-2: [Calculate interest points and generate vocab for test data and encode images]

Using SIFT descriptor calculate interest points and using **pre-trained kmeans** object generate vocabulary and encode testing images using this vocab.

Step-3: [Apply standard scaler.]

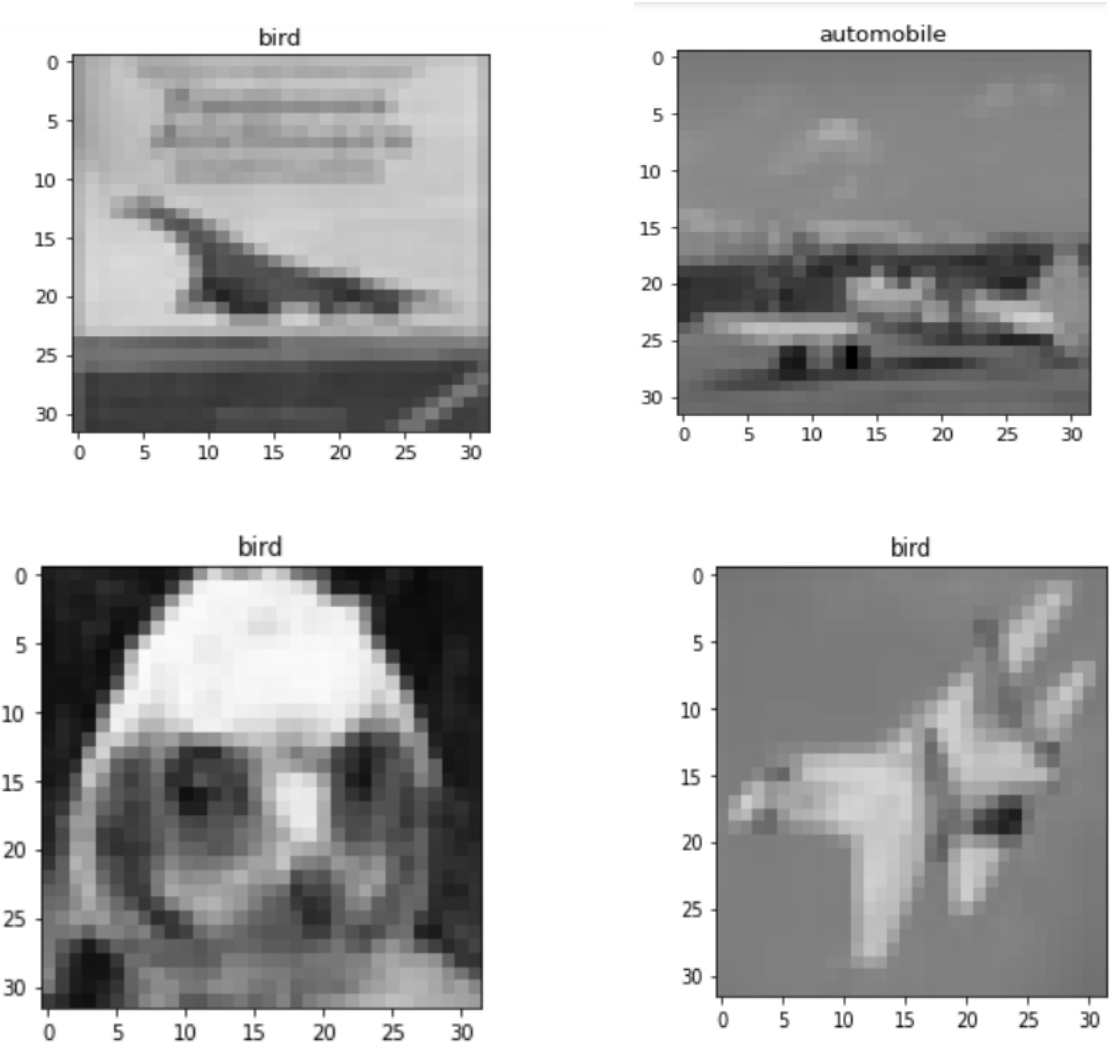
Step-4: [Predict the class]

Using pre-trained svm object, predict the class of the image and maintain count of true predictions. Moreover maintain an dictionary of predictions which will store image and it's predicted label.

Step-5: [Print out accuracy of model and images]

Now using that predictions dictionary, print out the images.

Here are the results. (Accuracy on this small test set was 50%)



Observations:

As we can see all flying objects are classified as **Bird.** So it may be because of small vocabulary (K=20) which has lead to clustering airplane wings and birds wings in same cluster. So will check by increasing size of K.

Experiments on complete data using this algorithm:

Using same algorithm as discussed above but with bit modification such that it shows **class wise accuracy** on complete data and different value of K and different learning algorithms following results are obtained.

SVM and vocab size (K) as 20: (Acc. 23.89)

```
airplane true predictions: 387 total image: 1000 Accuracy of airplane is 38.7
automobile true predictions: 286 total image: 1000 Accuracy of automobile is 28.599999999999998
bird true predictions: 150 total image: 1000 Accuracy of bird is 15.0
cat true predictions: 118 total image: 1000 Accuracy of cat is 11.799999999999999
deer true predictions: 111 total image: 1000 Accuracy of deer is 11.1
dog true predictions: 219 total image: 1000 Accuracy of dog is 21.9
frog true predictions: 244 total image: 1000 Accuracy of frog is 24.4
horse true predictions: 225 total image: 1000 Accuracy of horse is 22.5
ship true predictions: 366 total image: 1000 Accuracy of ship is 36.6
truck true predictions: 283 total image: 1000 Accuracy of truck is 28.299999999999997
```

```
print(true_count/total_count)
```

```
0.2389
```

SVM and vocab size (K) as 200: (Acc. 29.3)

```
airplane true predictions: 404 total image: 1000 Accuracy of airplane is 40.400000000000006
automobile true predictions: 352 total image: 1000 Accuracy of automobile is 35.199999999999996
bird true predictions: 188 total image: 1000 Accuracy of bird is 18.8
cat true predictions: 180 total image: 1000 Accuracy of cat is 18.0
deer true predictions: 171 total image: 1000 Accuracy of deer is 17.1
dog true predictions: 303 total image: 1000 Accuracy of dog is 30.3
frog true predictions: 253 total image: 1000 Accuracy of frog is 25.3
horse true predictions: 322 total image: 1000 Accuracy of horse is 32.2
ship true predictions: 398 total image: 1000 Accuracy of ship is 39.800000000000004
truck true predictions: 359 total image: 1000 Accuracy of truck is 35.9
```

```
print(true_count/total_count)
```

```
0.293
```

SVM with polynomial kernel and vocab size (K) as 200: (Acc. 24.37)

```
airplane true predictions: 292 total image: 1000 Accuracy of airplane is 29.2
automobile true predictions: 204 total image: 1000 Accuracy of automobile is 20.4
bird true predictions: 309 total image: 1000 Accuracy of bird is 30.9
cat true predictions: 121 total image: 1000 Accuracy of cat is 12.1
deer true predictions: 132 total image: 1000 Accuracy of deer is 13.200000000000001
dog true predictions: 165 total image: 1000 Accuracy of dog is 16.5
frog true predictions: 123 total image: 1000 Accuracy of frog is 12.3
horse true predictions: 177 total image: 1000 Accuracy of horse is 17.7
ship true predictions: 701 total image: 1000 Accuracy of ship is 70.1
truck true predictions: 213 total image: 1000 Accuracy of truck is 21.3
```

```
print(true_count/total_count)
```

```
0.2437
```

Observation:

As we can see from the class wise accuracy that **cat and deer** are having quite low accuracy compared to other classes. This might be because cat and deer have

similar features and because of this model might be miss classifying both of this like classifying cat as deer and deer as cat . So to deal with it I have tried increasing K to 400.

SVM and vocab size (K) as 400: (Acc. 29.43)

```
airplane true predictions: 396 total image: 1000 Accuracy of airplane is 39.6
automobile true predictions: 382 total image: 1000 Accuracy of automobile is 38.2
bird true predictions: 212 total image: 1000 Accuracy of bird is 21.2
cat true predictions: 167 total image: 1000 Accuracy of cat is 16.7
deer true predictions: 172 total image: 1000 Accuracy of deer is 17.2
dog true predictions: 283 total image: 1000 Accuracy of dog is 28.299999999999997
frog true predictions: 251 total image: 1000 Accuracy of frog is 25.1
horse true predictions: 342 total image: 1000 Accuracy of horse is 34.2
ship true predictions: 388 total image: 1000 Accuracy of ship is 38.800000000000004
truck true predictions: 350 total image: 1000 Accuracy of truck is 35.0
```

```
print(true_count/total_count)
```

```
0.2943
```

Observations:

As we can see, with increase in K value is not improving accuracy with SVM. Hence I had tried with other learning algorithm i.e KNN.

KNN and vocab size (K) as 200: (Acc. 14.17)

```
airplane true predictions: 584 total image: 1000 Accuracy of airplane is 58.4
automobile true predictions: 81 total image: 1000 Accuracy of automobile is 8.1
bird true predictions: 199 total image: 1000 Accuracy of bird is 19.900000000000002
cat true predictions: 51 total image: 1000 Accuracy of cat is 5.1
deer true predictions: 89 total image: 1000 Accuracy of deer is 8.9
dog true predictions: 33 total image: 1000 Accuracy of dog is 3.3000000000000003
frog true predictions: 46 total image: 1000 Accuracy of frog is 4.6
horse true predictions: 12 total image: 1000 Accuracy of horse is 1.2
ship true predictions: 307 total image: 1000 Accuracy of ship is 30.7
truck true predictions: 15 total image: 1000 Accuracy of truck is 1.5
```

```
print(true_count/total_count)
```

```
0.1417
```

KNN and vocab size (K) as 400: (Acc. 13.88)

```
processing  airplane
airplane true predictions: 613 total image: 1000 Accuracy of airplane is 61.3
processing  automobile
automobile true predictions: 58 total image: 1000 Accuracy of automobile is 5.800000000000001
processing  bird
bird true predictions: 228 total image: 1000 Accuracy of bird is 22.8
processing  cat
cat true predictions: 32 total image: 1000 Accuracy of cat is 3.2
processing  deer
deer true predictions: 100 total image: 1000 Accuracy of deer is 10.0
processing  dog
dog true predictions: 17 total image: 1000 Accuracy of dog is 1.7000000000000002
processing  frog
frog true predictions: 24 total image: 1000 Accuracy of frog is 2.4
processing  horse
horse true predictions: 1 total image: 1000 Accuracy of horse is 0.1
processing  ship
ship true predictions: 312 total image: 1000 Accuracy of ship is 31.2
processing  truck
truck true predictions: 3 total image: 1000 Accuracy of truck is 0.3
```

Observations:

As we can see, KNN is not performing better compared to SVM. So the best accuracy so far obtained is **29.43** .

NOTE: Even though we are training on complete data set, but actually it is not being trained on complete data set because when there is 0 interest points we are not considering that image (Training algo step-2). Hence total train data size is 49911.

Overall observations:

As we saw even by doubling K value i.e vocabulary size, accuracy is marginally increasing. The reason behind it might as image size is 32*32 which is very small and because of this even with increase in K value feature extractor is not able to extract features as it's so much blurred.

And also for classes like deer, cat, dog, horse all share similar features like 4 legs, 2 eyes, etc and we are encoding in vector by simply counting their number of occurrences which is not affective in distinguishing such liked classes.

So this are the reasons which will make bottle neck on accuracy with BOV encoding.

Classification using VLAD approach:

In above discuss approach, each image was encoded in a form of vector which contains count of occurrence of each word from vocabulary which is present in that image. But in VLAD, this vector is made up of **sum of difference of descriptors in that image to all centers obtained from knn**. That is we match a descriptor to its closest cluster, then for each cluster, we store the sum of the differences of the descriptors assigned to the cluster and the centroid of the cluster.

Things that will change in code part compared to BOV:

For VLAD approach, the pipeline will remain same, but the change will be in encoding the images and in building vocabulary, as we need centroids hence I have used **cv2.kmeans()** rather then sklearns **kmeans().**

Also as we need to match descriptor to its closest cluster, I have defined a function which **calculates the norm** (using cv2.norm() function)of descriptor with all centroids and finds min distance centroid.

Algorithm for encoding part (As rest all is same from BOV):

Step-1: Iterate over each descriptor in image.

Step-2: For each descriptor, find the closest centroid (use any distance measure like L2 norm).

Step-3: Now sum up the difference between each descriptor and closest centroid.

Code snippet in python:

```
for des in descriptors:
        nearest_center = find_nearest_center(des, centers)
        for i in range(len(descriptors[0]):
                vlad_vector[i]+= (des[i] - nearest_center)
return vlad_vector
```

Trying on small data set with 3 classes "Airplane, Automobile, Bird":

First to test the code, I had trained it on subset of CIFAR-10 data set. On this subset of data, **the accuracy obtained is 68.7% which is more** then the one obtained using BOVW approach which was around 50%.

And with trying with KNN algorithm, the accuracy obtained was around 18%.

```
airplane true predictions: 3 total image:1000 Accuracy of airplane is 0.3
automobile true predictions: 0 total image:1000 Accuracy of automobile is 0.0
bird true predictions: 0 total image:1000 Accuracy of bird is 0.0
```

```
print("Acc.", true/len(true_label)*100)
```

```
Acc. 18.75
```

As we can see KNN is not able to identify Automobile and Bird.

Note: On complete data, it took to long to run and at the end memory exhausted error came.

Observed difference between BOVW and VLAD:

Computational time:

VLAD is more time consuming as compared to BOVW. This is because in VLAD we are assigning closest cluster to each descriptor and then calculating the difference whereas in BOVW we are simply counting occurrence of visual words.

Efficiency in terms of accuracy:

VLAD is more accurate compared to BOVW. This might be because in BOVW we are counting visual words, so for example if in an image there is 2 eyes then **encoded vector will be same as for Bird, Dog, Deer etc** and this will lead to miss classification. But in case of VLAD we are taking distance into account and not just occurrence.