INTERNATIONAL INSTITUTE OF INFORMATION TECHNOLOGY – BANGALORE

MACHINE PERCEPTION
ASSIGNMENT – 2

ARUN M GEORGE – MT2016025
MITHUN MATHEW – MT 2016085

QUESTION 2A

Segmentation

- Pick 5 images from Berkeley Segmentation dataset
- Run Kmeans and Mean shift
- Show results along with the ground-truth

ANSWER

KMEANS

k-means clustering is a method of vector quantization that is popular for cluster analysis. k-means clustering aims to partition *n* observations into *k* clusters in which each observation belongs to the cluster with the nearest mean, serving as a prototype of the cluster.

Given a set of observations ($\mathbf{x}_1$, $\mathbf{x}_2$, …, $\mathbf{x}_n$), where each observation is a *d*-dimensional real vector, *k*-means clustering aims to partition the *n* observations into k ($\leq n$) sets $\mathbf{S} = \{S_1, S_2, \ldots, S_k\}$ to minimize the within-cluster sum of squares (WCSS) (sum of distance functions of each point in the cluster to the K center).

In our approach, we first smoothen the image by applying median filtering using the Python OpenCV function medianBlur() with a kernel size of 9. Then, we reshape the image such that every pixel is a point in three-dimension having its RGB value. We apply kmeans clustering on this image using the Python OpenCV function kmeans(). The result is reshaped back to the shape of the original image. This image will have colors of the cluster centers. On this image, we apply Canny edge detector algorithm, using the Python OpenCV function Canny(), to obtain the segments in the image. We plot the segments back onto the original image to get the final output.

The OpenCV-Python function kmeans() has the following input parameters:

- samples: It should be of np.float32 data type, and each feature should be put in a single column. The reshaped image is used in our approach.
- nclusters(K): Number of clusters required at end. We used 3 and 5 clusters.
- criteria: It is the iteration termination criteria. When this criterion is satisfied, algorithm iteration stops. It should be a tuple of 3 parameters. They are (type, max_iter, epsilon):
    - type of termination criteria: It has 3 flags as below:

        - cv2.TERM_CRITERIA_EPS - stop the algorithm iteration if specified accuracy, epsilon, is reached.
        - cv2.TERM_CRITERIA_MAX_ITER - stop the algorithm after the specified number of iterations, max_iter.
        - cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER - stop the iteration when any of the above condition is met. We used this in our approach.
    - max_iter - An integer specifying maximum number of iterations. We used a value of 10.
    - epsilon - Required accuracy. We used an accuracy of 1.0.

- attempts: Flag to specify the number of times the algorithm is executed using different initial labellings. The algorithm returns the labels that yield the best compactness. This compactness is returned as output. We used a value of 10.
- flags: This flag is used to specify how initial centers are taken. Normally two flags are used for this:
  cv2.KMEANS_PP_CENTERS and cv2.KMEANS_RANDOM_CENTERS.
  Here, we used cv2.KMEANS_RANDOM_CENTERS to select centers randomly.

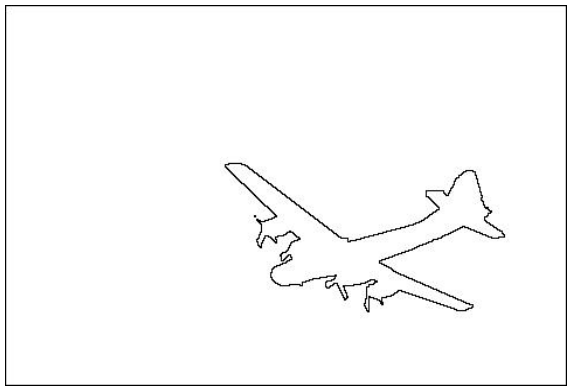
Its output parameters are:

- compactness: It is the sum of squared distance from each point to their corresponding centers.
- labels: This is the label array where each element marked ‘0’, ‘1’,...
- centers: This is array of centers of clusters.

OUTPUT

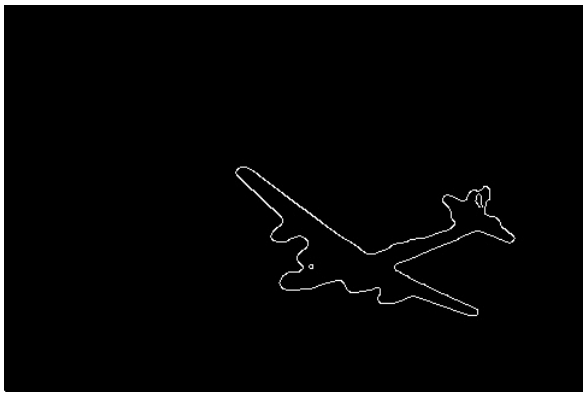
1)

Original Image



Ground Truth

After Kmeans

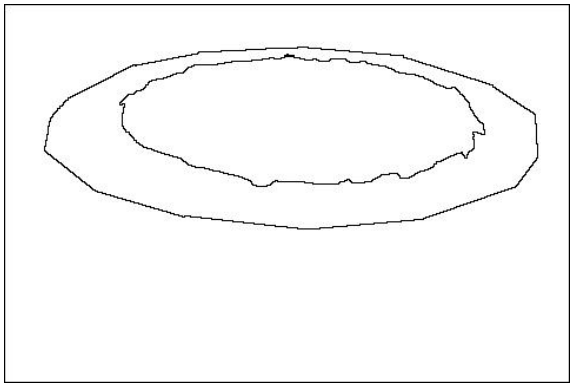


Output of Canny

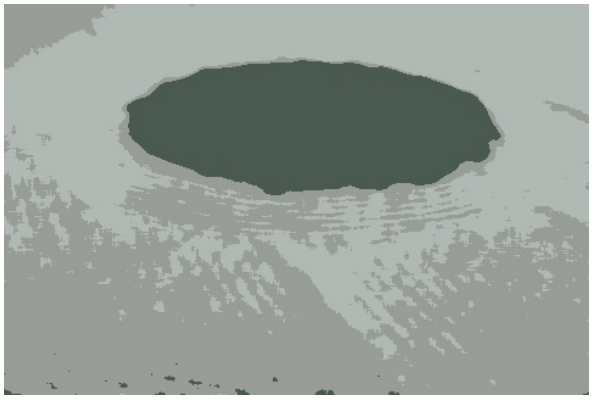
Final Output

2)

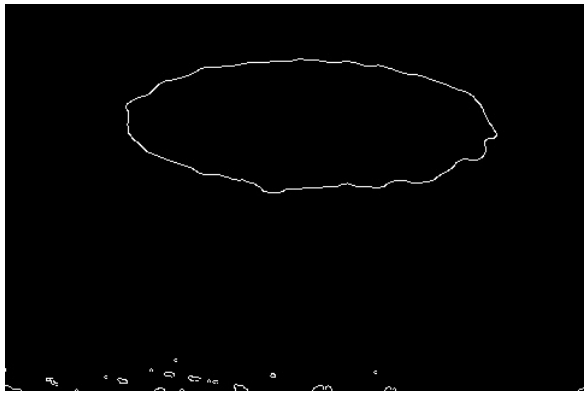
Original Image



Ground Truth



After Kmeans

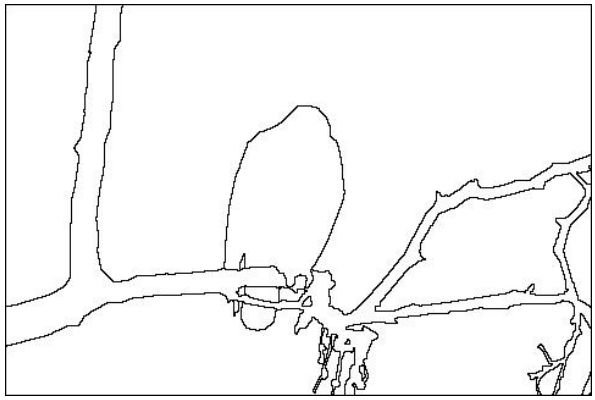


Output of Canny

Final Output

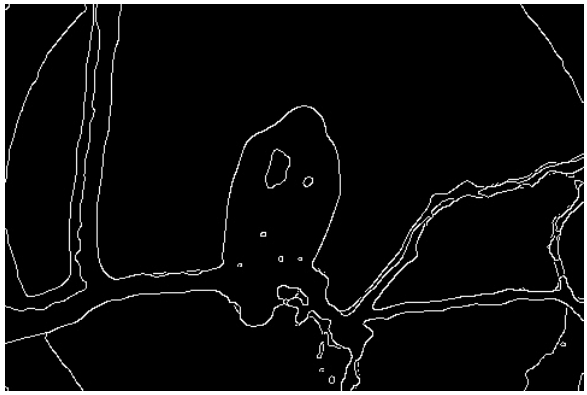
3)

Original Image



Ground Truth

After Kmeans

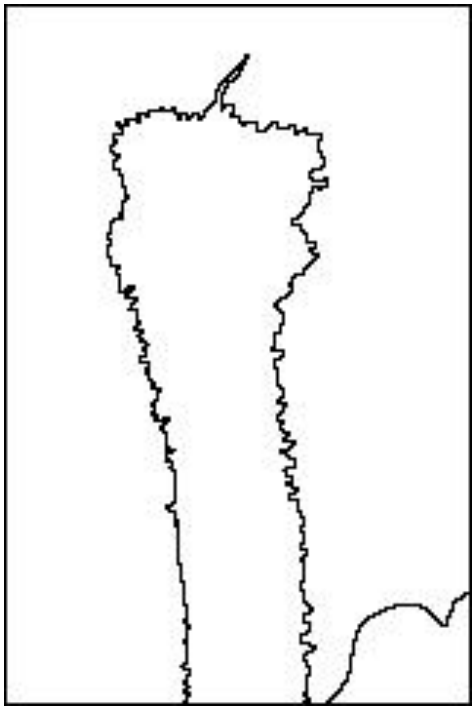


Output of Canny

Final Output

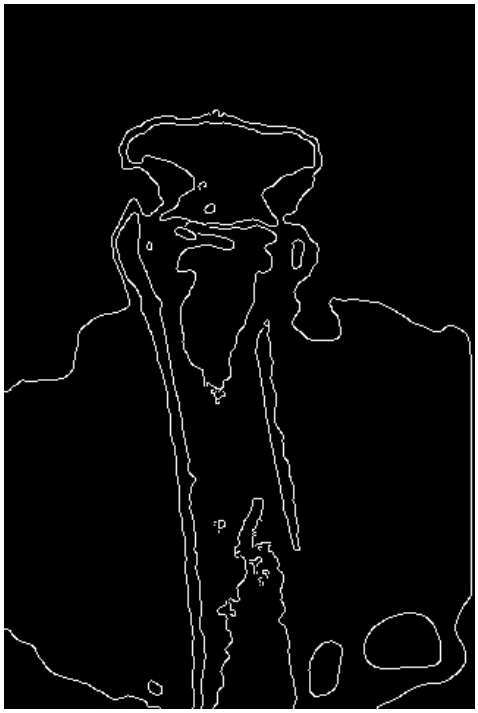
4)

Original Image



Ground Truth

After Kmeans

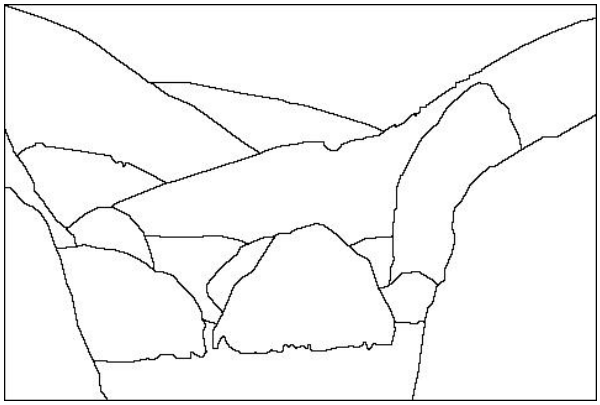


Output of Canny

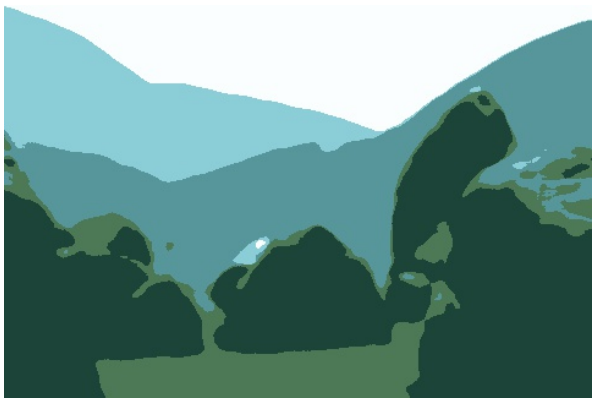
Final Output

5)

Original Image



Ground Truth



After Kmeans

Output of Canny

Final Output

MEAN SHIFT

Mean shift is a procedure for locating the maxima of a density function given discrete data sampled from that function. It is useful for detecting the modes of this density. This is an iterative method, and we start with an initial estimate.

Consider a set of points in two-dimensional space. Assume a circular window centered at C and having radius r as the kernel. Mean shift is a hill climbing algorithm which involves shifting this kernel iteratively to a higher density region until convergence. Every shift is defined by a mean shift vector. The mean shift vector always points toward the direction of the maximum increase in the density. At every iteration, the kernel is shifted to the centroid or the mean of the points within it. The method of calculating this mean depends on the choice of the kernel. If a Gaussian kernel is chosen, then every point will first be assigned a weight which will decay exponentially as the distance from the kernel's center increases. At convergence, there will be no direction at which a shift can accommodate more points inside the kernel.

In our approach, we first smoothen the image by applying median filtering using the Python OpenCV function medianBlur() with a kernel size of 9. Then, we apply Mean shift clustering on this image using the Python OpenCV function pyrMeanShiftFiltering(). The output of the function is the filtered image with color gradients and fine-grain texture flattened. On this image, we apply Canny edge detector algorithm, using the Python OpenCV function Canny(), to obtain the segments in the image. We plot the segments back onto the original image to get the final output.

The OpenCV-Python function pyrMeanShiftFiltering() has the following input parameters:

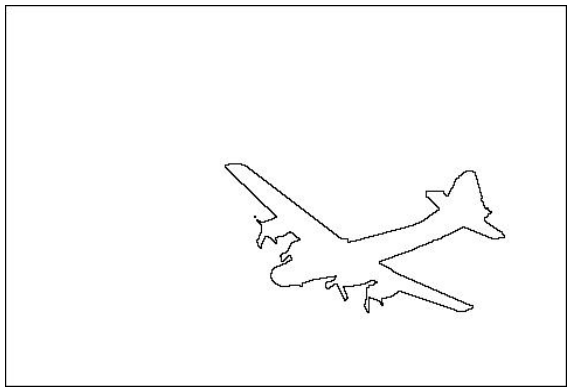
- src – The source 8-bit, 3-channel image. In our approach, we give the image after applying median filtering.
- sp – The spatial window radius. Here, we use the value 8.
- sr – The color window radius. Here, we use the value 16.
- maxLevel – Maximum level of the pyramid for the segmentation. Here, we use the value 1.
- termcrit – Termination criteria, that is, when to stop mean shift iterations. Here, we use

(cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 5,1) where 5 is the number of iterations and 1 is the required accuracy. When either of these is met, it terminates.

OUTPUT

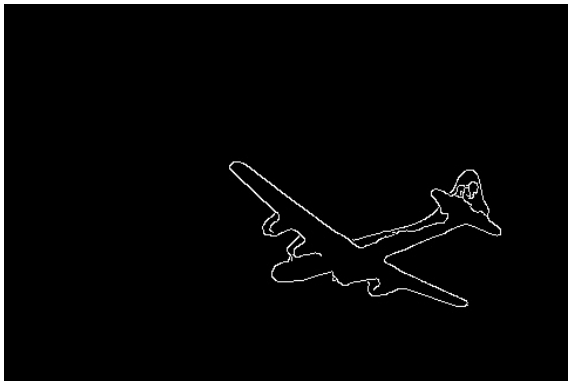
1)

Original Image



Ground Truth

After Mean shift

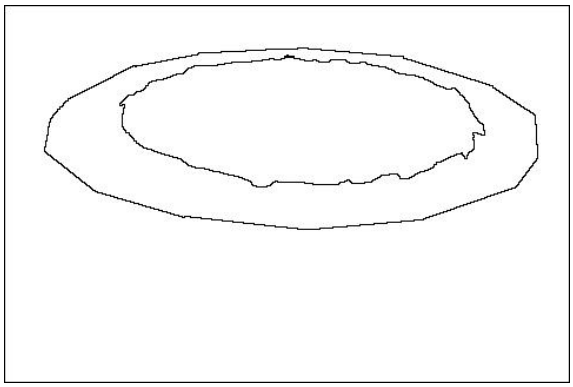


Output of Canny

Final Output

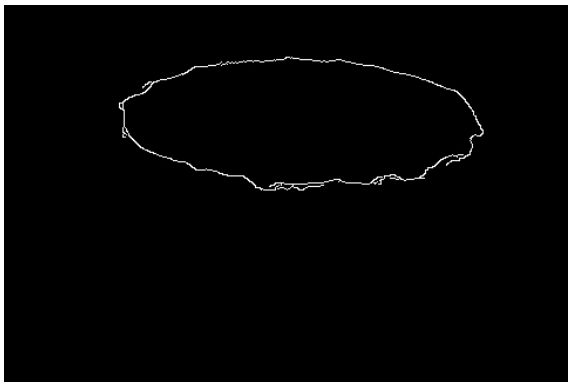
2)

Original Image



Ground Truth

After Mean shift

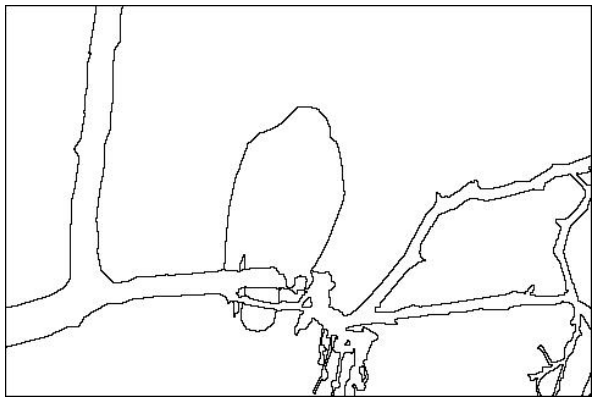


Output of Canny

Final Output

3)

Original Image



Ground Truth

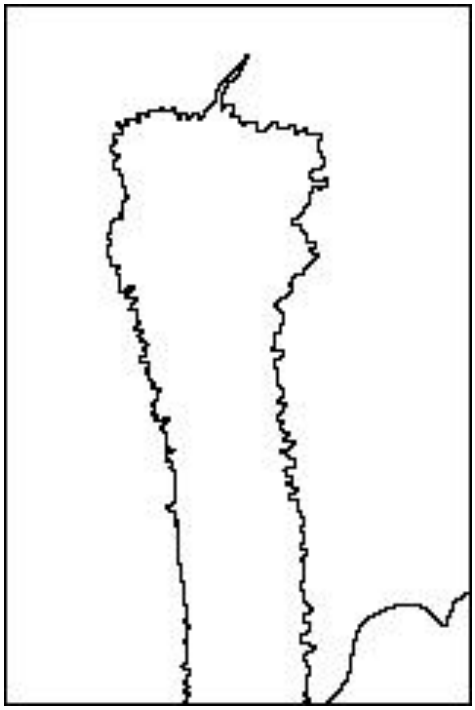
After Mean shift

Output of Canny

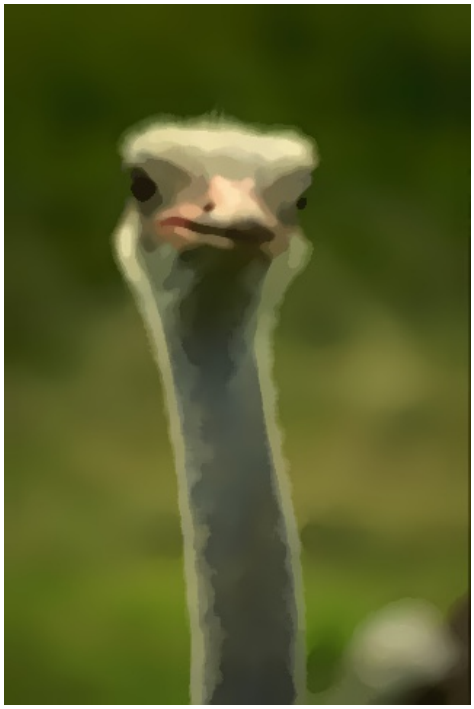
Final Output

4)

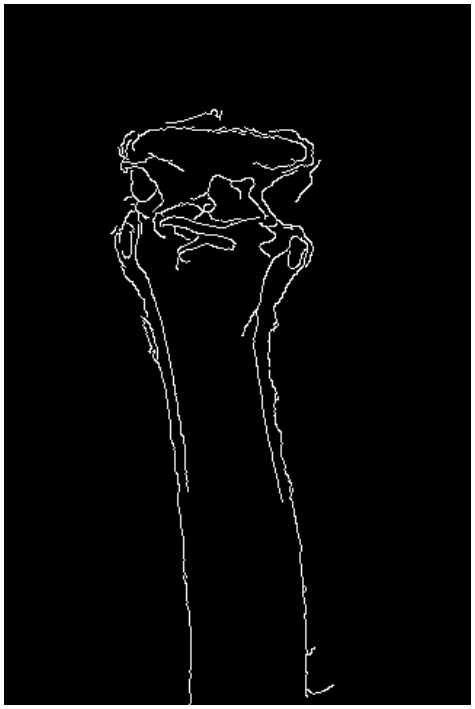
Original Image



Ground Truth



After Mean shift

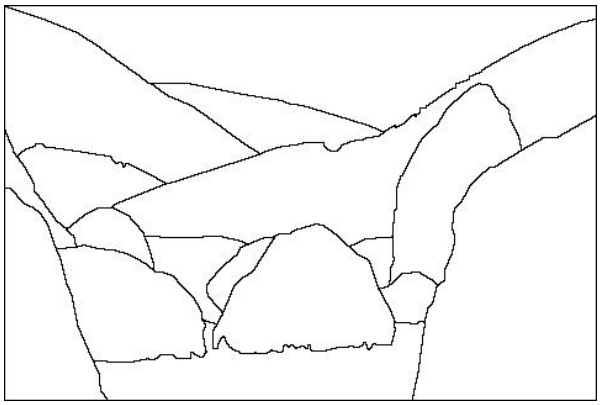


Output of Canny

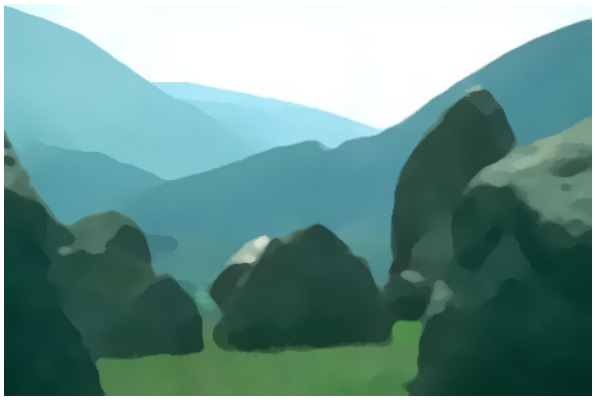
Final Output

5)

Original Image



Ground Truth



After Mean shift

Output of Canny

Final Output

QUESTION 2B

- Play with Panorama
- Use this code
- http://ramsrigoutham.com/tag/ransac/
- and stitch together a panaroma
- Explain how SURF is different from SIFT (10 sentences)
- Briefly explain the main principles of FLANN matching (5 sentences)

ANSWER

PANORAMA

In our implementation, we use SURF to find the keypoints and the descriptors of the input images. OpenCV-Python lets us create a SURF object using the syntax 'cv2.xfeatures2d.SURF_create(<hessianThreshold>)' where hessianThreshold is the threshold for hessian keypoint detector used in SURF. The keypoints and descriptors of the two images to be merged are found using the SURF object and stored.

We use a FLANN matcher with kd tree as the algorithm with 5 randomized trees and 50 as the value of check (number of times trees in the index are recursively traversed). Then, we find good matches which satisfy the Lowe's ratio test. We then find corresponding keypoints for the good descriptors and create the Homography matrix using the OpenCV-Python function 'findHomography()'. Using this Homography matrix, we warp the two images together and append the warped portion to the original image to get the Panorama.

OUTPUT

1)

Picture 1

Picture 2

Panorama

2)

Picture 1

Picture 2

Panorama

Comparison between SIFT (Scale-Invariant Feature Transform) and SURF (Speeded Up Robust Features)

1. SIFT is a scale-invariant algorithm to detect and describe local features in an image. SURF, an algorithm to detect and describe local features in an image, is inspired from SIFT with lesser number of computations and hence, higher speed. When speed is not critical, SIFT outperforms SURF.
2. The pure image descriptor in SIFT is significantly better than the pure image descriptor in SURF but the pure interest point detector in SURF is better than that in SIFT.
3. For faster computation, SURF uses integral images whereas SIFT doesn't use integral images. An integral image is an image whose entry at a location ***x*** represents the sum of all pixels in the input image within a rectangular region formed by the origin and ***x***.
4. SIFT uses Difference of Gaussian(DoG), which approximates Laplacian of Gaussian (LoG), to detect the keypoints whereas SURF uses determinant of approximation of Hessian of the filtered image, which is a better method.
5. Since SURF uses integral images, its performance is independent of the kernel size of the filter. SIFT doesn't have this property.
6. For scale-invariance in SIFT, we initially apply Gaussian blur to get a set of images of different scales. In SURF, for reducing the computation we do not directly apply Gaussian blur, but instead we approximate the effect of Gaussian blur using box linear filter.
7. In SIFT, we iteratively filter the input image with a Gaussian of fixed width until the scale of the next octave (octave corresponds to doubling of scale factor) is reached whereas in SURF, as there is no runtime penalty for applying Gaussian filters of increased size due to integral images, we can compute directly the image filtered at each scale without using previous scaled results. So, this step of SURF can be parallelized.
8. In SURF, there is a lowest level of scale of $\sigma = 1.2$ which corresponds to box filter size of 9x9 whereas SIFT does not have any such lower bounds on scale factor.
9. The SURF descriptor describes intensity distribution based on first order Haar wavelet responses in x and y directions, which leads to a descriptor of 64 dimensions (there are versions with other dimensions also, but they are less efficient when it comes to matching). In SIFT, we use gradient orientation for describing intensity distribution, leading to 128 dimensional features.

10. In SURF, for fast indexing during the matching stage, the sign of the Laplacian (trace of the Hessian matrix) of the keypoints is used. This feature is available at no extra computational cost as it was already computed during the detection phase. SIFT does not have such a fast indexing.

Main Principles of FLANN matching

FLANN (Fast Library for Approximate Nearest Neighbors) is a library that contains collection of algorithms optimized for fast nearest neighbor search in large datasets and for high dimensional features. Finding the best match in local image features is done by finding the nearest neighbors.

For high-dimensional spaces, there are often no known algorithms for nearest neighbor search that are more efficient than simple linear search. As linear search is too costly, this has generated an interest in algorithms that perform approximate nearest neighbor search, in which optimal neighbors are sometimes returned. Such approximate algorithms can be orders of magnitude faster than exact search, while still providing near-optimal accuracy. Performance of these algorithms varies widely based on the properties of dataset such as dimensionality, correlations, clustering characteristics and size.

In FLANN, there is an automatic algorithm selection and configuration, which allows the best algorithm and parameter settings to be determined automatically for any given dataset.

There are two main algorithms – randomized kd-tree algorithm and hierarchical k-means tree algorithm – used in FLANN. Randomized kd-tree algorithm is an improved version of kd-tree algorithm in which randomized trees are built by choosing split dimension randomly from the first D dimensions (usually D=5) on which data has greatest variance. For searching the trees, we maintain a single priority queue across all randomized trees so that search can be ordered by increasing distance to each bin boundary. There is a search precision parameter which specifies the maximum number of leaf nodes examined before search is terminated.

Hierarchical k-means tree algorithm is constructed by splitting the data points at each level into k distinct regions using a k-means clustering and then applying same method recursively. There is a priority queue which keeps track

of unexplored branches along the tree traversal. Branch that has the closest center to the query point is extracted from the priority queue and the search is carried out. There are two parameters – branching factor (how deep we can go in a branch) and check parameter (number of leaf nodes examined before termination).

FLANN matcher takes a subset of given data and finds out the optimum algorithm and its parameters by solving it as an optimization problem which minimizes a cost function (a combination of search time, tree build time and tree memory overhead) in the parameter space.

QUESTION 2C

- Implement Bike vs Horse Classification
- Dataset is available on LMS
- Use Bag-of-visual words approach (SIFT/SURF + K-means + Logistic Regression/KNN)
- Explain the procedure and your approach and observations

ANSWER

The concept of Bag of Words is taken from text analysis. The idea is to represent a document as a 'bag' of important keywords, without ordering of the words. For images, the idea is similar. We represent an object as a bag of visual words - patches that are described by a certain descriptor.

We can use the bag of visual words model for object categorization by constructing a large vocabulary of many visual words and representing each image as a histogram of the frequency of words that are in the image.

First, we need to build a visual dictionary. We do that by taking a large set of object images and extracting descriptors from them. Next, we cluster the set of descriptors to k clusters. The cluster centers act as our dictionary's visual words.

Given a new image, we represent it using the model in the following manner: first, extract descriptors from the image. Next, for each descriptor extracted compute its nearest neighbor in the dictionary. Finally, build a histogram of length k where the i^{th} value is the frequency of the i^{th} dictionary word. This model can be used in conjunction with a kNN model for object classification.

In our approach, the first task was to create a Bag of Visual words which uses Kmeans algorithm to train visual vocabulary. Python OpenCV allows this using the syntax 'cv2.BOWKMeansTrainer(<dictionarySize>)' where dictionarySize is the size of the dictionary.

Then, we extract the descriptors of the training images. We did this using the SIFT(Scale-Invariant Feature Transform) approach. Python OpenCV allows us to create a SIFT object using the syntax 'cv2.xfeatures2d.SIFT_create()'. This SIFT object was used to detect the keypoints and descriptors using the function 'detectAndCompute(<image_name>,None)'. The descriptors of each of the

images are added to the bag of words. When all the descriptors are obtained in the bag, we cluster the descriptors into ‘k’ (here value used is 12) clusters using the function cluster().

Now, we initialize a BOWImgDescriptorExtractor object. The syntax is ‘cv2.BOWImgDescriptorExtractor(sift2, cv2.BFMatcher(cv2.NORM_L2))’ where ‘sift2’ is a SIFT object used to compute descriptors for an input image and its keypoints. Here, Brute-Force Matcher is used as the descriptor matcher that is used to find the nearest word of the trained vocabulary for each keypoint descriptor of the image. BOWImgDescriptorExtractor object computes descriptors for a given image and its keypoints set. Then, it finds the nearest visual words from the vocabulary for each keypoint descriptor and computes the bag-of-words image descriptor as a normalized histogram of vocabulary words encountered in the image. The i^{th} bin of the histogram is a frequency of i^{th} word of the vocabulary in the given image. The setVocabulary() function is used to set a visual vocabulary. The compute() function of BOWImgDescriptorExtractor object computes an image descriptor using the set visual vocabulary. Using this compute() function, we create the features set required to create our kNN model. The labels are obtained from the file path.

Using the set of features and labels we have, we build our kNN model. We use this kNN model to predict the class of our test images by extracting the descriptors of the image.

In our approach, we used a training set of 86 images of bikes and horses. The test set of images contained 32 images. With this training and test set, we could achieve a classification accuracy of 81.25%.

OUTPUT

Bike

Bike

Bike

Bike

Horse


MotorcycleWorld.com
AUTOMOTIVE DIVISION
CITY OF LONG BEACH
243
CHESTNUT AVENUE
ENTER
www.honda.com

Horse

Horse

Horse

Bike

```
Created Bag of Words
Created KNN model
Actual Class    -       Predicted Class
Bike            -       Horse
Bike            -       Bike
Bike            -       Bike
Bike            -       Horse
Bike            -       Bike
Bike            -       Bike
Bike            -       Bike
Bike            -       Bike
Bike            -       Bike
Bike            -       Horse
Bike            -       Bike
Bike            -       Horse
Bike            -       Bike
Bike            -       Bike
Bike            -       Bike
Bike            -       Bike
Bike            -       Bike
Horse           -       Horse
Horse           -       Horse
Horse           -       Horse
Horse           -       Horse
Horse           -       Horse
Horse           -       Horse
Horse           -       Horse
Horse           -       Horse
Horse           -       Bike
Horse           -       Horse
Horse           -       Horse
Horse           -       Horse
Horse           -       Bike
Horse           -       Horse
Horse           -       Horse

Accuracy -  81.25 %
```