EE569: Introduction to Digital Image Processing

ASSIGNMENT 4

Texture Analysis
Image Matching

Suchismita Sahu | 19th March 2019 | USCID: 7688176370

PROBLEM 1: TEXTURE ANALYSIS

TEXTURE CLASSIFICATION

This task is to generate 25 5*5 Laws filters and to create filtered images from which texture features are computed and eventually classify the input texture image into 4 clusters using K-means clustering (unsupervised learning) and visual inspection (supervised learning).

ABSTRACT AND MOTIVATION

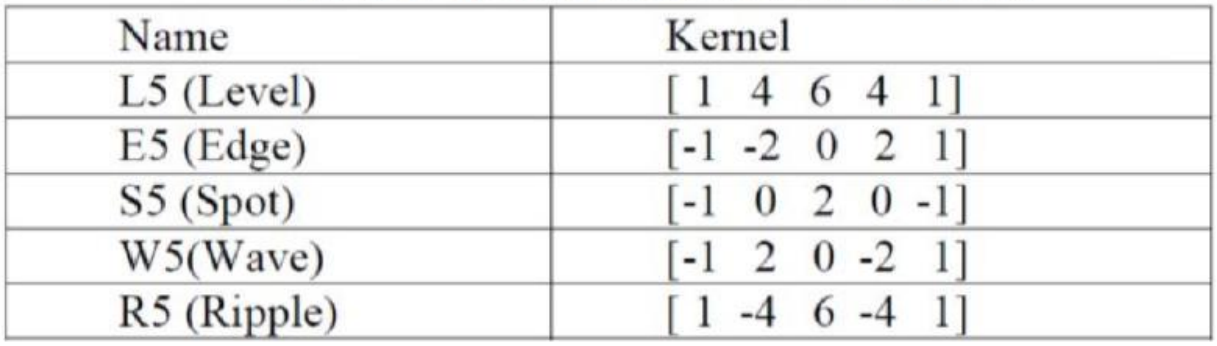
Textures in images can be considered as patterns that are repeated periodically that can be segmented and used for classification eventually. Thus, Image texture features are important in image segmentation and image classification tasks. Textures provide the spatial arrangement of various intensity values in an image.

APPROACH AND PROCEDURE

Texture classification involves feature extraction and clustering by generating a codebook and extracting features from it.

Firstly, we construct 25 Laws Filters using the kernels for Wave, Edge, Spot, Ripple and Level as illustrated below:

Table 1: 1D Kernel for 5x5 Laws Filters



Name	Kernel
L5 (Level)	[1 4 6 4 1]
E5 (Edge)	[-1 -2 0 2 1]
S5 (Spot)	[-1 0 2 0 -1]
W5(Wave)	[-1 2 0 -2 1]
R5 (Ripple)	[1 -4 6 -4 1]

The input texture images are first filtered using the above Laws Filters and the local energy from their output images are computed: **Multichannel Feature Extraction.**

Laws Filters are calculated using the tensor product of 2 kernels. Hence, for the above 5 1-dimensional kernels, we get 25 2D Laws filters:

E5E5	**E5S5**	**E5W5**	**E5L5**	**E5R5**
S5E5	**S5S5**	**S5W5**	**S5L5**	**S5R5**
W5E5	**W5S5**	**W5W5**	**W5L5**	**W5R5**
L5L5	**L5E5**	**L5S5**	**L5W5**	**L5R5**
R5L5	**R5E5**	**R5S5**	**R5W5**	**R5R5**

Tensor Product can be illustrated as:

$$L5^T L5 = \begin{bmatrix} 1 \\ 4 \\ 6 \\ 4 \\ 1 \end{bmatrix} \begin{bmatrix} 1 & 4 & 6 & 4 & 1 \end{bmatrix}$$

$$L5^T L5 = \begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$$

An example: E5L5

$$\begin{bmatrix} -1 \\ -2 \\ 0 \\ 2 \\ 1 \end{bmatrix} \times \begin{bmatrix} 1 & 4 & 6 & 4 & 1 \end{bmatrix} = \begin{bmatrix} -1 & -4 & -6 & -4 & -1 \\ -2 & -8 & -12 & -8 & -1 \\ 0 & 0 & 0 & 0 & 0 \\ 2 & 8 & 12 & 8 & 2 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$$

Figure 1 Tensor Product

Overall STEPS:

1. **Preprocessing:**

DC components of the image are subtracted from the image so that we eliminate any high feature value that might hinder feature extraction using the equation:

$$Img_{DC_{removed}} = Input_{Img} - (Mean\ of\ all\ pixels\ in\ each\ of\ the\ input\ images)$$

2. **Feature Extraction:**

For each input image, we find 25 features of the image using the above calculated Laws filters.

The Laws filters are then clubbed to create a filter bank. Each filter from the filter bank is applied to the image and the average energy of the entire image is calculated which is the feature required. This process is then repeated for all the input images so that we have 25D feature vector for each training image.

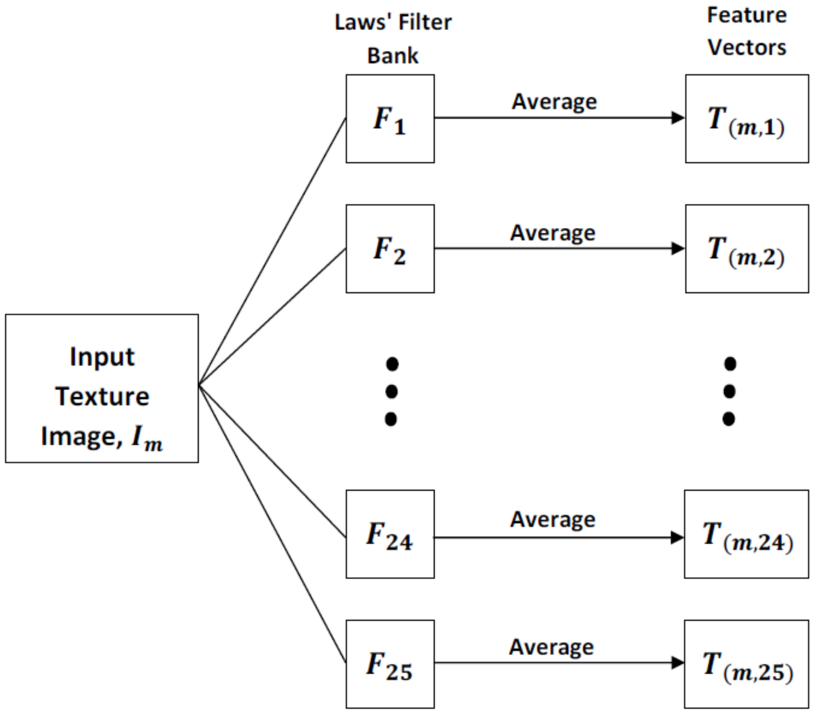


Figure 2 Laws Filter Bank

I_m is the input image here. And F1 to F25 are the Laws Filter created using the kernels and T are the respective feature vectors calculated by the following equation:

$$T_{(m,n)} = \frac{1}{KL} \sum_{i=0}^{K} \sum_{j=0}^{L} F_n(i,j)$$

The final feature vector obtained for each image is of the dimension M*N where M is the number of input images and N is the number of Laws filters used for texture feature extraction.

The above 12*25 output is then normalized to find the energy of the feature vector as per:

$$Feature_{Vector_{normalized}} = \frac{Feature_{Vector} - (Mean\ of\ all\ the\ data\ in\ the\ feature\ vector)}{Standard\ deviation\ of\ the\ feature\ vector}$$

3. **PCA For Dimensionality Reduction:**

After calculating the feature vectors using the above equation, we need to implement PCA to reduce the feature dimension as not all 25 feature vectors may be important.

Number of dimensions increases time and space complexity. PCA reduces dimensionality by calculating the Singular Value Decomposition i.e. Eigen Values and Eigen Vectors of the features and then reducing the feature space to n dimensions whose basis are the eigen vectors corresponding to the n top most eigen values.

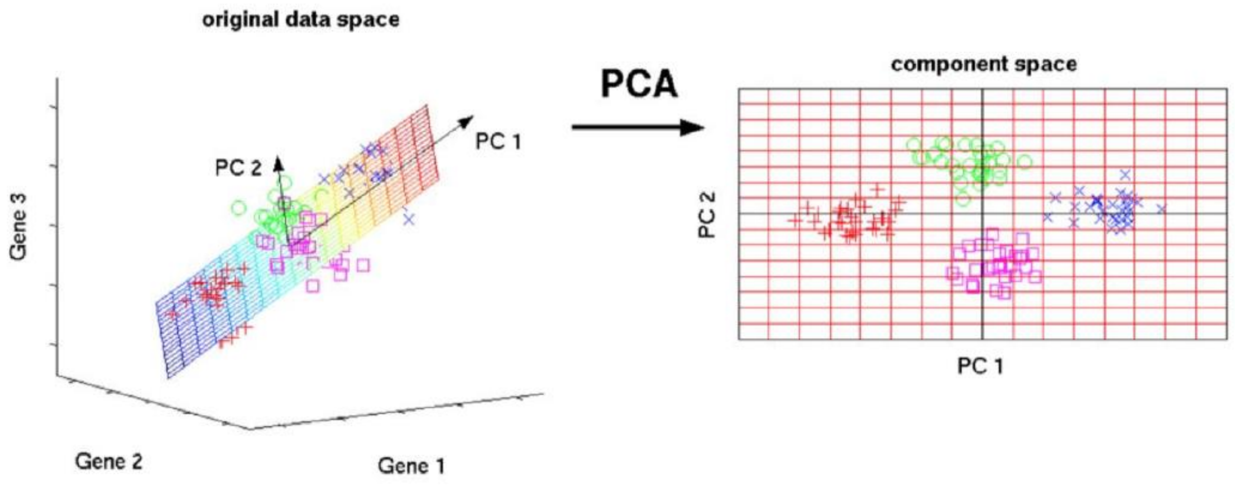
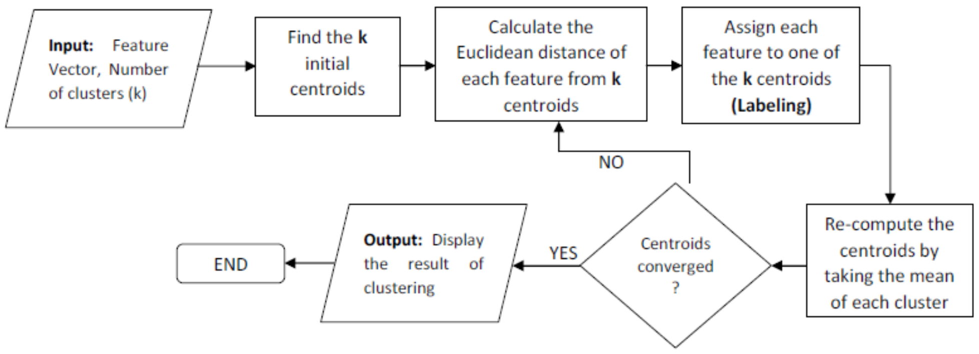


Figure 3 PCA

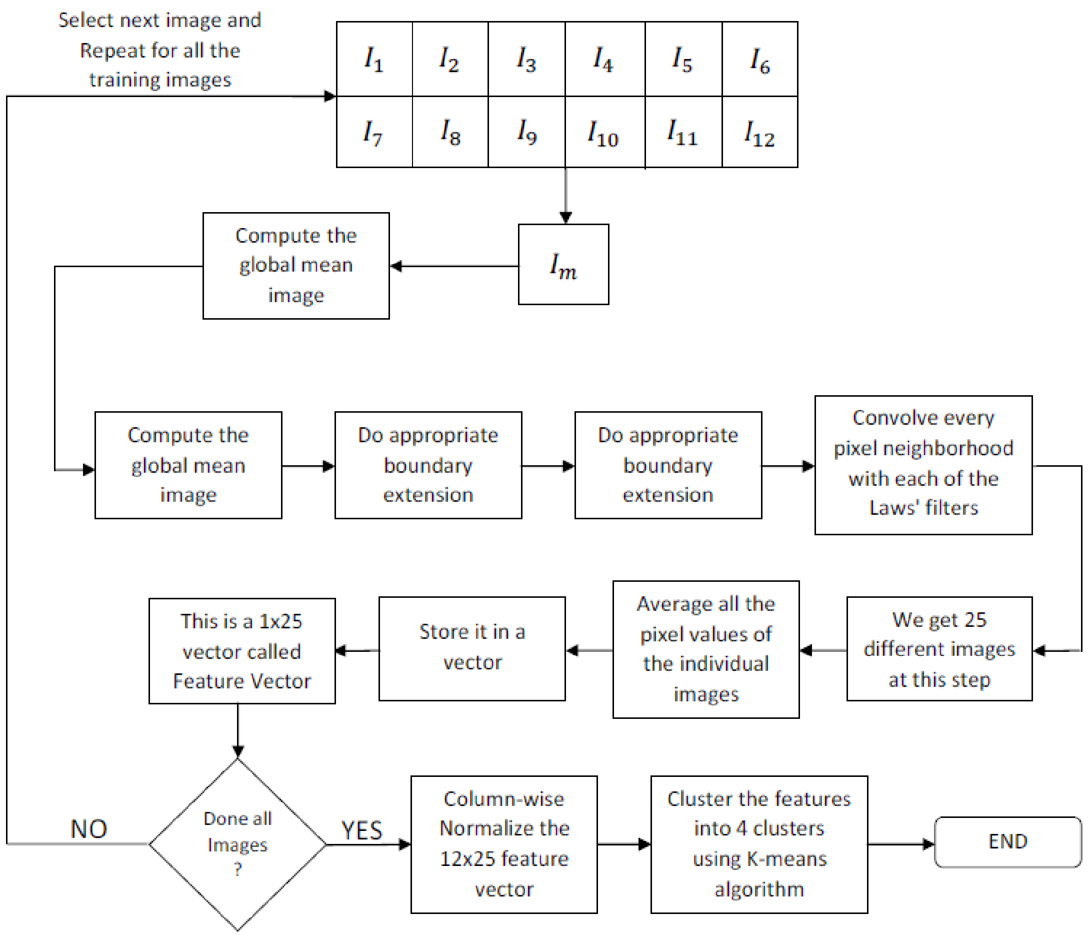
4. Clustering using K-Means:

The known textures are first clustered into n clusters with centroids. This is then passed to k-means clustering algorithm:



Input: Feature Vector, Number of clusters (k)
Find the k initial centroids
Calculate the Euclidean distance of each feature from k centroids
Assign each feature to one of the k centroids (Labeling)
Re-compute the centroids by taking the mean of each cluster
Centroids converged ?
NO
YES
Output: Display the result of clustering
END

Figure 4 K-means Clustering Flow-Chart



Select next image and Repeat for all the training images
I_1 I_2 I_3 I_4 I_5 I_6
I_7 I_8 I_9 I_{10} I_{11} I_{12}
I_m
Compute the global mean image
Compute the global mean image
Do appropriate boundary extension
Do appropriate boundary extension
Convolve every pixel neighborhood with each of the Laws' filters
We get 25 different images at this step
Average all the pixel values of the individual images
Store it in a vector
This is a 1x25 vector called Feature Vector
Done all Images ?
NO
YES
Column-wise Normalize the 12x25 feature vector
Cluster the features into 4 clusters using K-means algorithm
END

Figure 5 Texture Classification Flow Chart

Overall approach for Texture Classification is as follows:

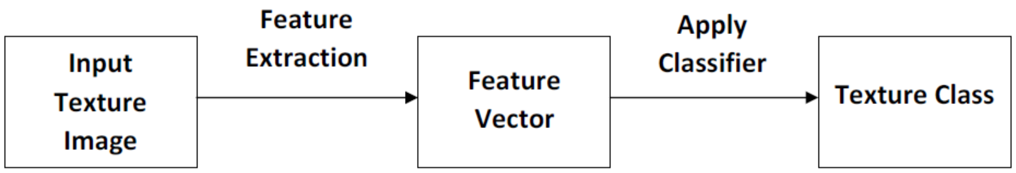


Figure 6 Overview of Texture Classification

Algorithm Used for Problem 1.(a) :TEXTURE FEATURE EXTRACTION

1. For each texture image input:
    a. Subtract the Image mean to reduce the illumination effect
    b. Apply 25 5*5 Laws filters to obtain the 25 texture filtered images
    c. Average the energy to form a 25 Dimensional Feature vector
    d. Define 15 Dimensional feature vector by replacing each pair with average as follows:

L5L5	L5E5/E5L5	E5S5/S5E5
E5E5	L5S5/S5L5	E5W5/W5E5
S5S5	L5W5/W5L5	E5R5/R5E5
W5W5	L5R5/R5L5	S5W5/W5S5
R5R5	W5R5/R5W5	S5R5/R5S5

    e. Thus, feature Set = n samples, m feature dimension
2. For each dimension, do feature normalization.
3. Perform K-Means Clustering:
    a. Chose random K initial centroids
    b. Assign each texture to one of the centroids by calculating Euclidean distance between the feature and the centroid
    c. Update the centroids using:

$$C_K = \frac{1}{\sum_{i\ j} I(Label(i,j) = K)} \sum_{i}\sum_{j} I(Label(i,j) = K)\ T(i,j)$$

    Where, Label is the indicator function and K is the cluster
    d. Repeat process till algorithm converges iteratively when there's no updating of centroid.
4. Visually inspect the clusters created of the input texture images to verify classification of clusters and check misclassification.

EXPERIMENTAL RESULTS

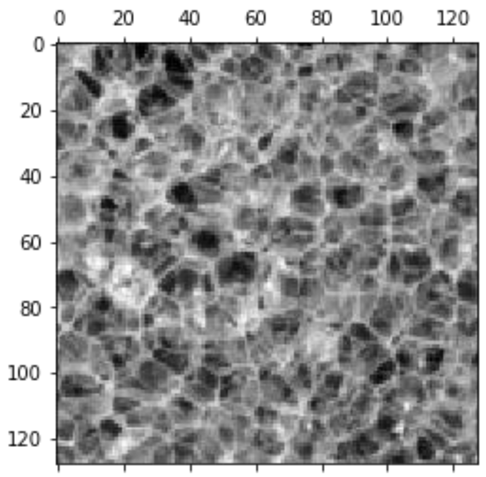


Figure 7 Texture 1

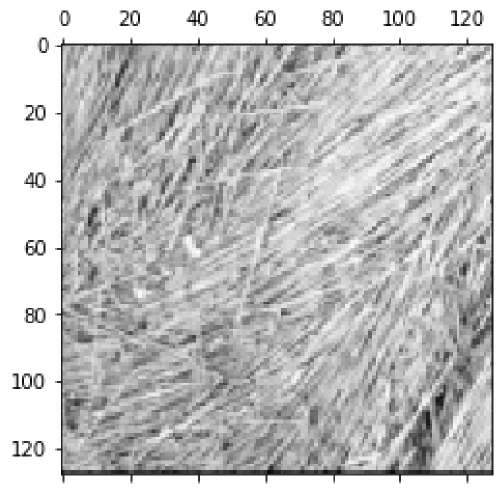


Figure 8 Texture 2

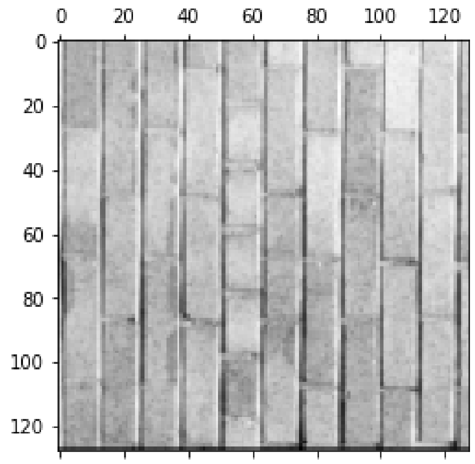


Figure 9 Texture 3

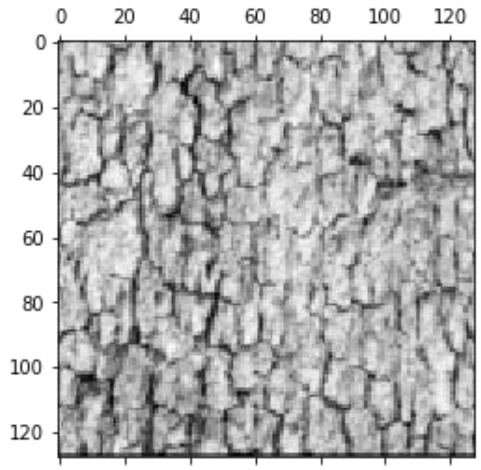


Figure 10 Texture 4

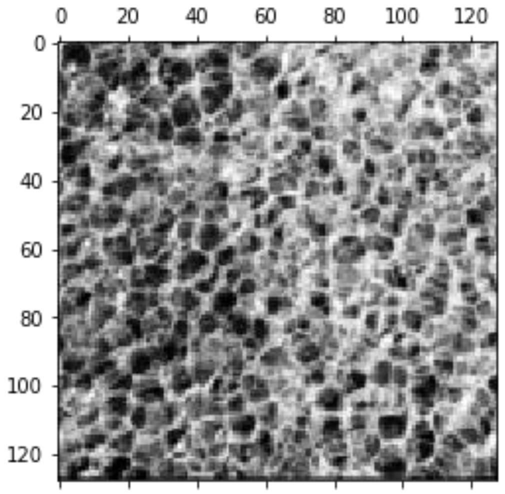


Figure 11 Texture 5

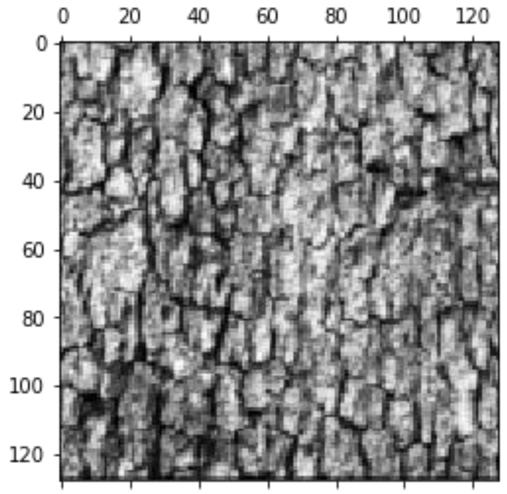


Figure 12 Texture 6

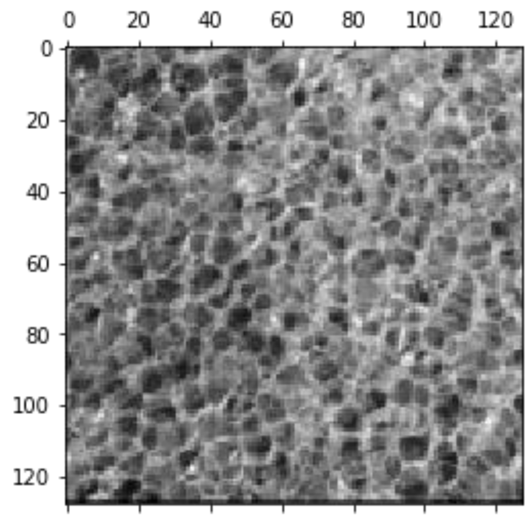


Figure 13 Texture 7

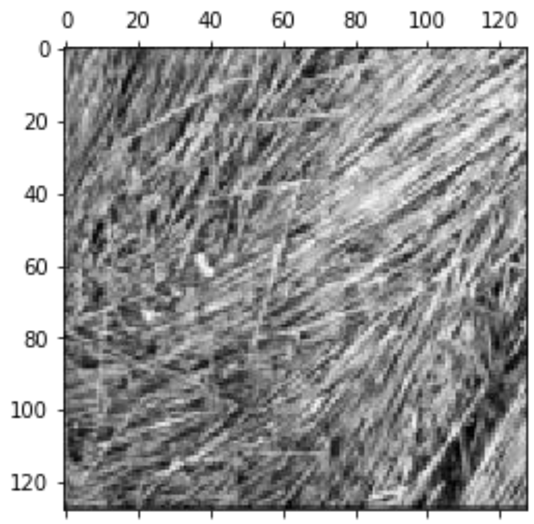


Figure 14 Texture 8

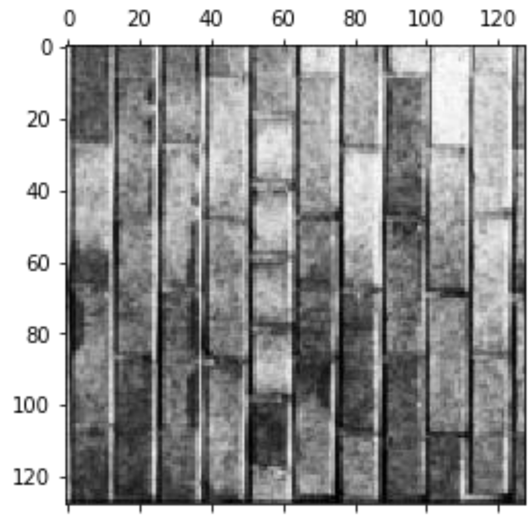


Figure 15 Texture 9

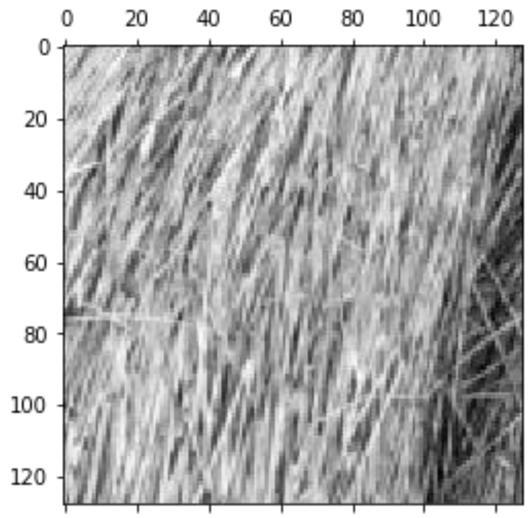


Figure 16 Texture 10

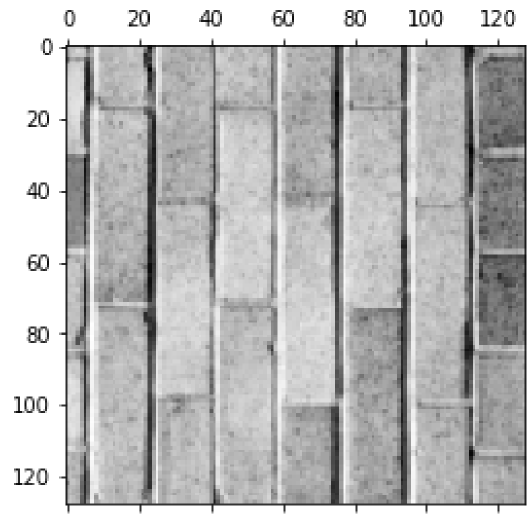


Figure 17 Texture 11

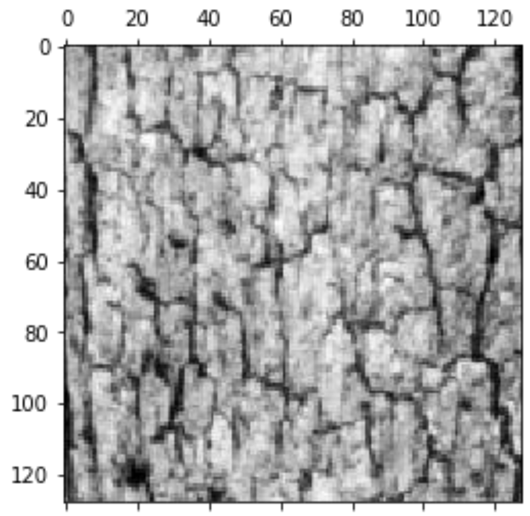


Figure 18 Texture 12

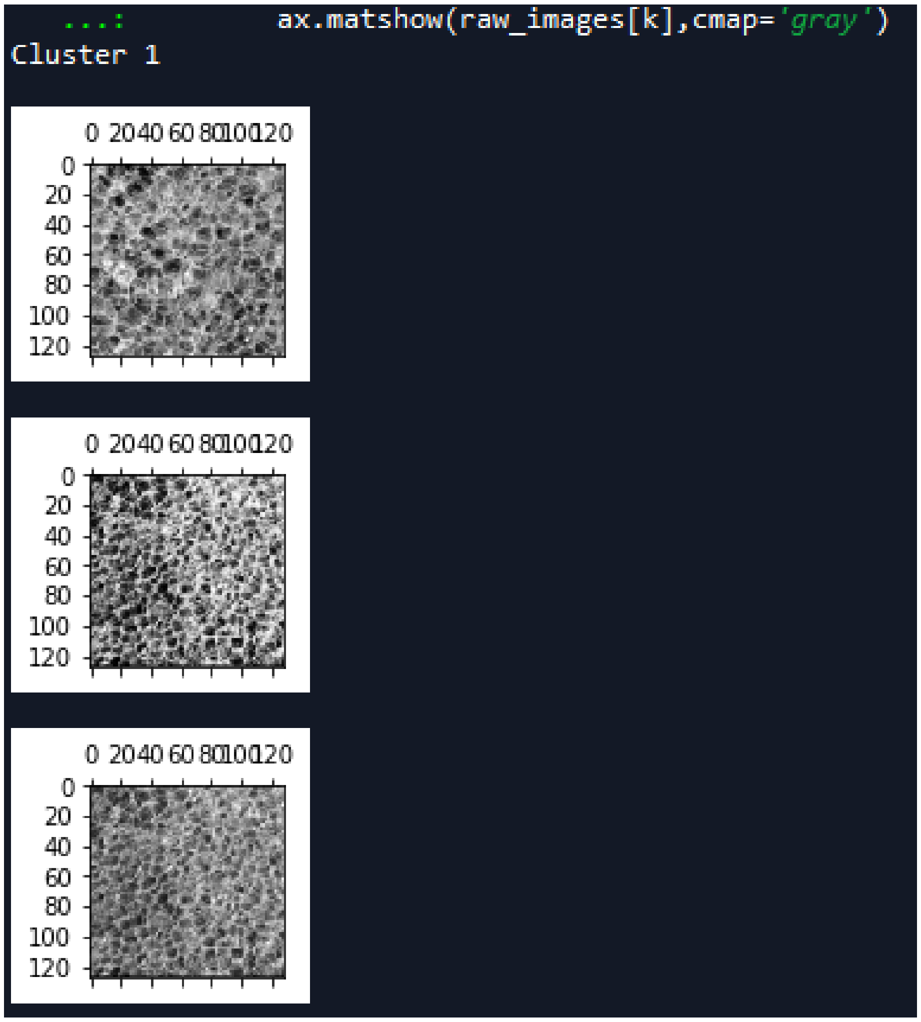
As we can see from the input images visually, we should have 4 clusters: Brick, Grass, Bark, Blobs.

Brick: Texture 3, Texture 9, Texture 11

Grass: Texture 2, Texture 8, Texture 10

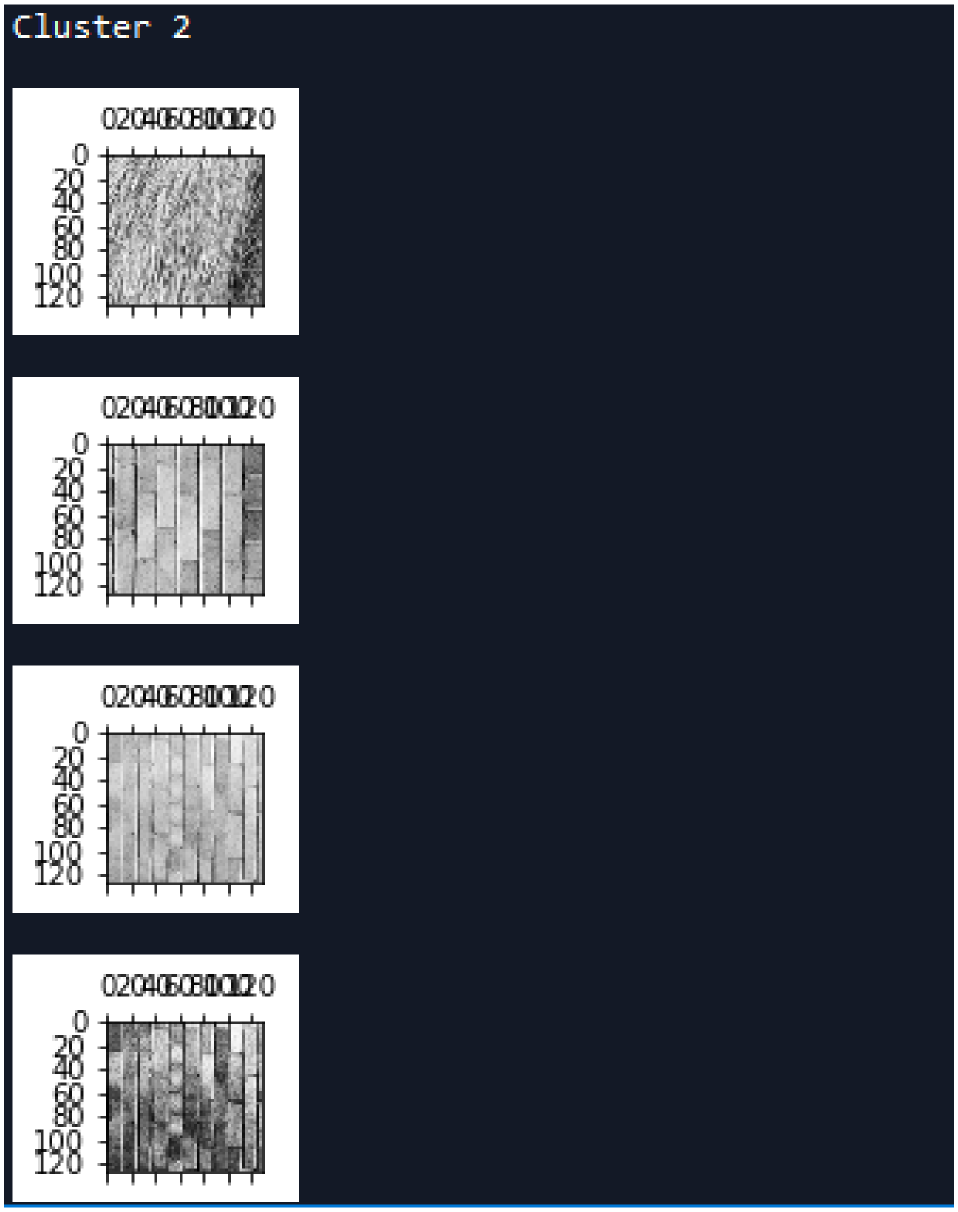
Bark: Texture 4, Texture 6, Texture 12

Blob: Texture 1, Texture 5, Texture 7



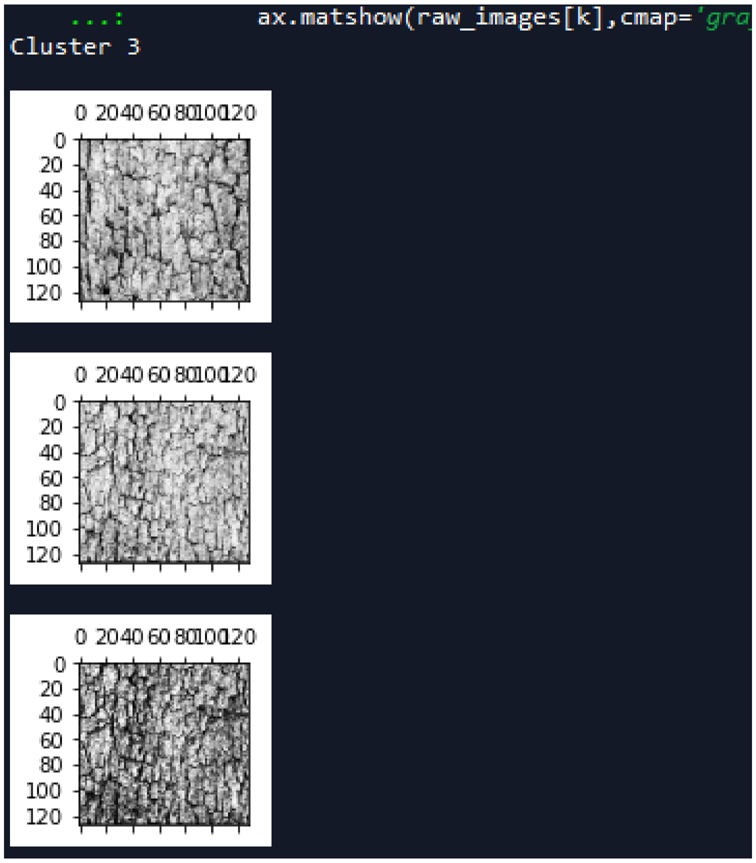
```
   ...:          ax.matshow(raw_images[k],cmap='gray')
Cluster 1
```

Figure 19 Blob Cluster Output



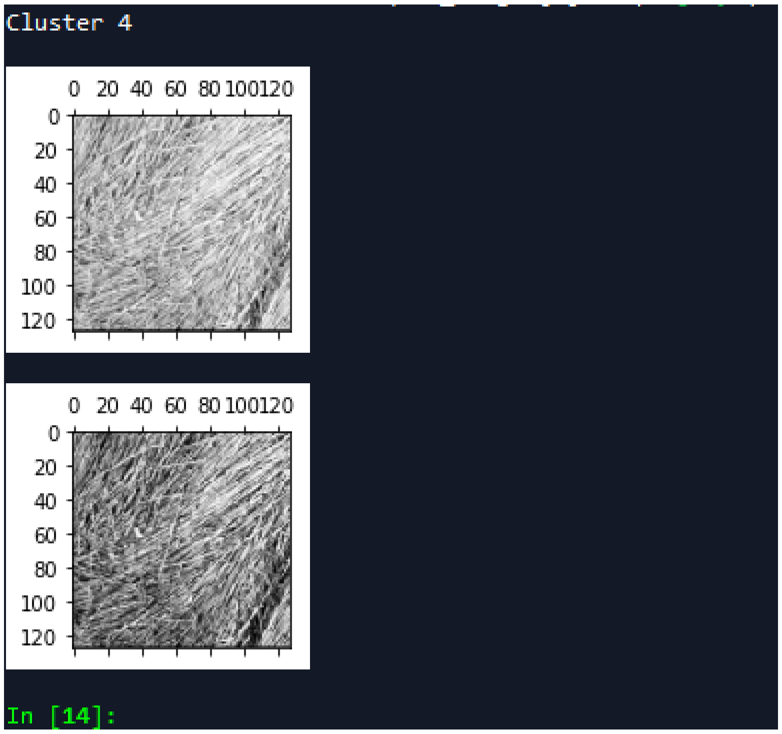
```
Cluster 2
```

Figure 20 Brick Cluster Output



```
    ...:          ax.matshow(raw_images[k],cmap='gra
Cluster 3
```

Figure 21 Bark Cluster Output



```
Cluster 4

In [14]:
```

Figure 22 Grass Cluster Output

As observed visually from the above results, there is one misclassification. One Grass texture is classified into Brick Texture.

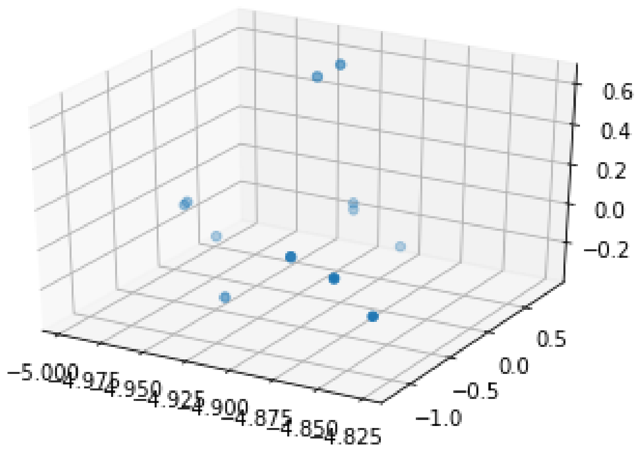


Figure 23 PCA of features in 3D

For Feature Discriminant Analysis, I picked the 1st row form Brick and Blob Textures and calculated the variance of each of the 25 filters applied to the images, i.e. my 12*25 data frame. As tabulated below: Sum((data-mean(data)).pow 2)

Filter Bank Type	**Variance**
L5L5	1.76
E5L5	1.09
S5L5	0.78
W5L5	0.40
R5L5	0.09
L5E5	1.53
E5E5	0.73
S5E5	0.48
W5E5	0.17
R5E5	0.12
L5S5	1.18
E5S5	0.57
S5S5	0.04
W5S5	0.10
R5S5	0.14
L5W5	1.14
E5W5	0.57

S5W5	0.04
W5W5	0.11
R5W5	0.10
L5R5	0.77
E5R5	0.35
S5R5	0.18
W5R5	0.11
R5R5	0.45

DISCUSSION

1. Implemented Boundary Extension while applying Laws Filters on the input texture images.
2. Discriminative power of the features extracted can be found by studying the number of overlapping pixels after filtering. Features having many overlapping points have weak discriminative power and cannot help in accurate classification. Lesser the number of overlapping data, the better the classification and strength of discriminative power.
3. Calculating the absolute difference between two feature vectors formed by the same Laws filter for two different texture images also gives us an idea about the strength of the Laws filter.
4. Feature reduction did not have a significant change in the performance of the clustering algorithm, but the feature convergence was much faster.
5. Filter **$S5^TS5$ ($S5^TW5$)** has the strongest discriminant power whereas **$L5^TL5$** has lowest discriminant power because the variance for **$S5^TS5$ ($S5^TW5$)** is minimum and it is maximum for **$L5^TL5$**.
6. This is because, none of the images have Wave so, the variance for S5W5 seems to be low. Also, because L5L5 is the only filter with non-zero mean, it is not an important feature extractor and hence has high variance and low discriminant power.
7. Grass Texture is wrongly classified into Class Bricks.
8. Apart from that, accuracy is 100% for the clustering.

IMAGE TEXTURE SEGMENTATION

The task is to implement texture image segmentation on the combination image.

APPROACH AND PROCEDURE

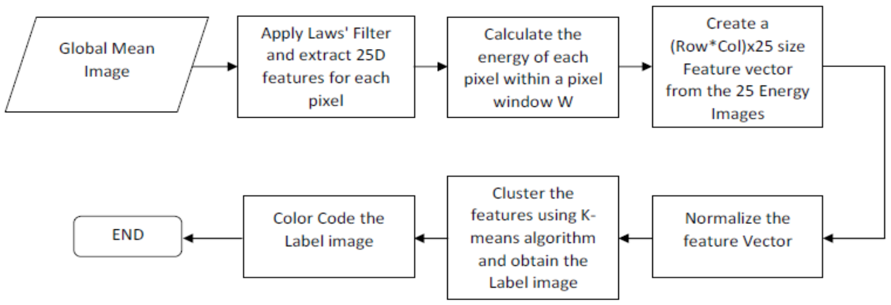

Global Mean Image
Apply Laws' Filter and extract 25D features for each pixel
Calculate the energy of each pixel within a pixel window W
Create a (Row*Col)x25 size Feature vector from the 25 Energy Images
Normalize the feature Vector
Cluster the features using K-means algorithm and obtain the Label image
Color Code the Label image
END

Figure 24 Flow Chart for Image Texture Segmentation

Overall Steps:

1. **Laws Feature extraction:**
   Apply the laws filters to get feature vectors
2. **Energy feature computation and normalization:**
   Implement windowed approach to compute the energy of each pixel using various window sizes: 13*13, 15*15 etc.

$$Energy = \frac{1}{(WindowSize * WindowSize)} \times \sum_{i=0,j=0}^{i=Height,j=Width} (Img(i,j) * Img(i,j))$$

   All the filters have zero-mean expect for laws filter L5TL5 thus feature extracted by L5TL5 is not useful and we use this energy to normalize the features.
3. **Segmentation:**
   K-means clustering to perform segmentation.

Algorithm Used for Problem 1.(b) :

1. For each texture image input:
   a. Subtract the Image mean to reduce the illumination effect
   b. Apply 25 5*5 Laws filters to obtain the 25 texture filtered images

   c. Average the energy using 15*15 window to form a 25 Dimensional Feature vector
   d. Thus, feature Set = n samples, m feature dimension
2. For each dimension, do feature normalization by L5TL5 energy.
3. Perform K-Means Clustering:
   a. Chose random K initial centroids
   b. Assign each texture to one of the centroids by calculating Euclidean distance between the feature and the centroid
   c. Update the centroids using:

$$C_K = \frac{1}{\sum_i \sum_j I(Label(i,j) = K)} \sum_i \sum_j I(Label(i,j) = K)\ T(i,j)$$

   Where, Label is the indicator function and K is the cluster
   d. Repeat process till algorithm converges iteratively when there's no updating of centroid.
4. Segmented output is displayed after clustering using grayscale color combination.

EXPERIMENTAL RESULTS

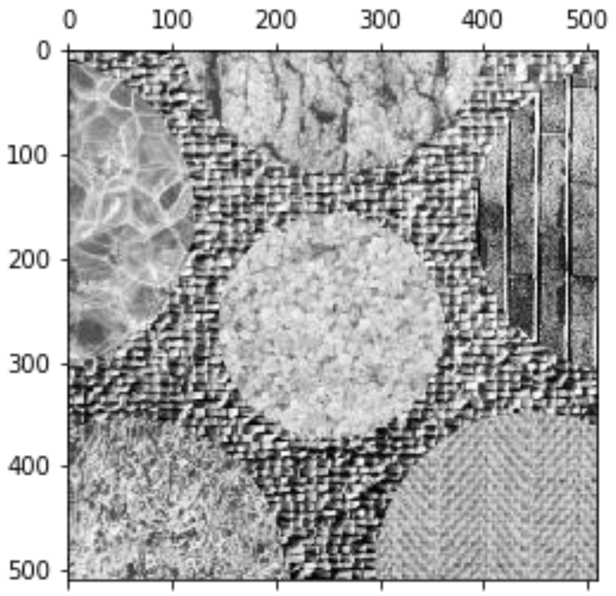


Figure 25 Texture Combination Image

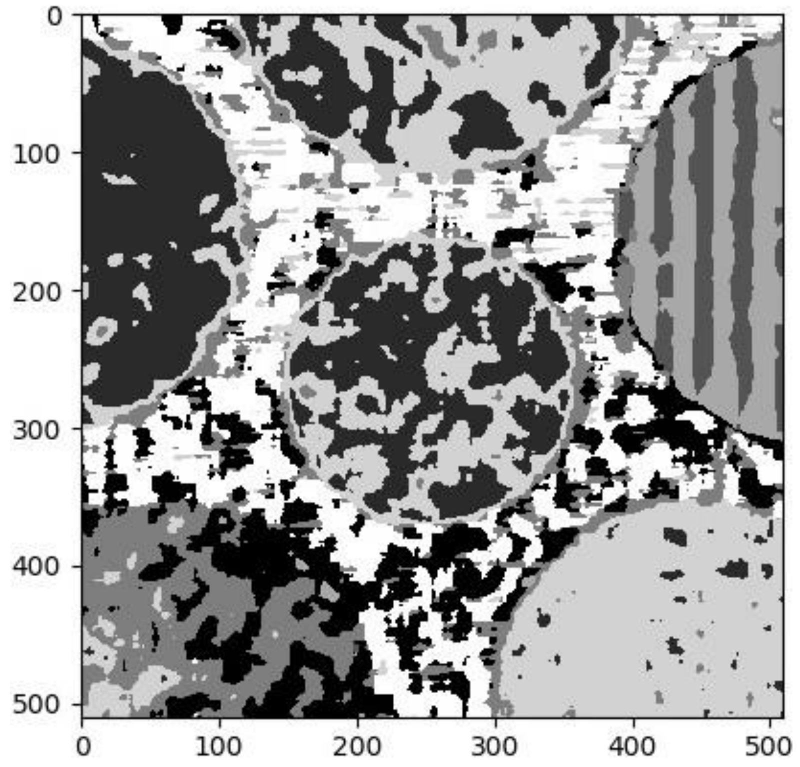


Figure 26 Output Segmented Image with 13*13 Window Size

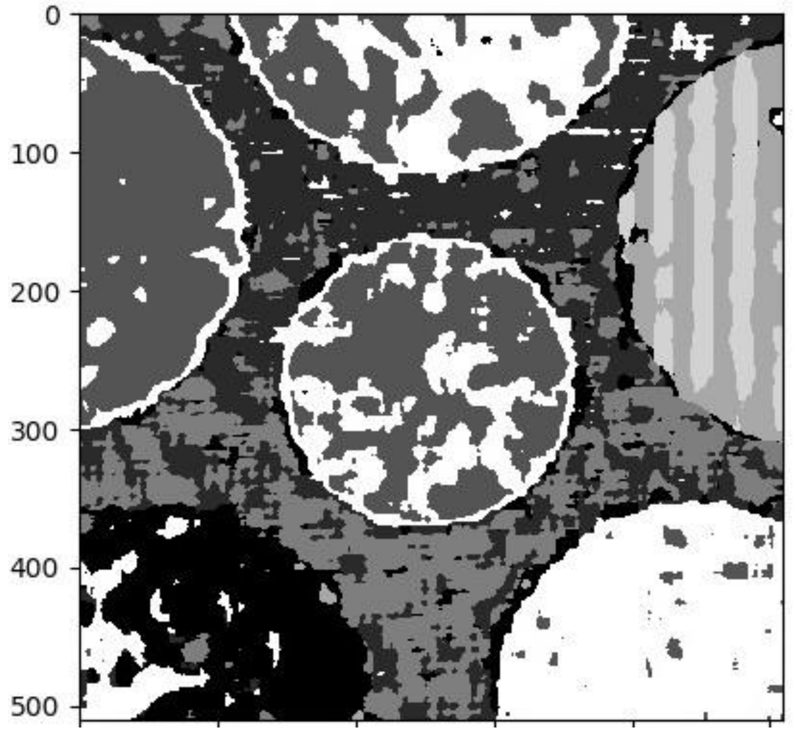


Figure 27 Output Segmented Image with 15*15 Window Size

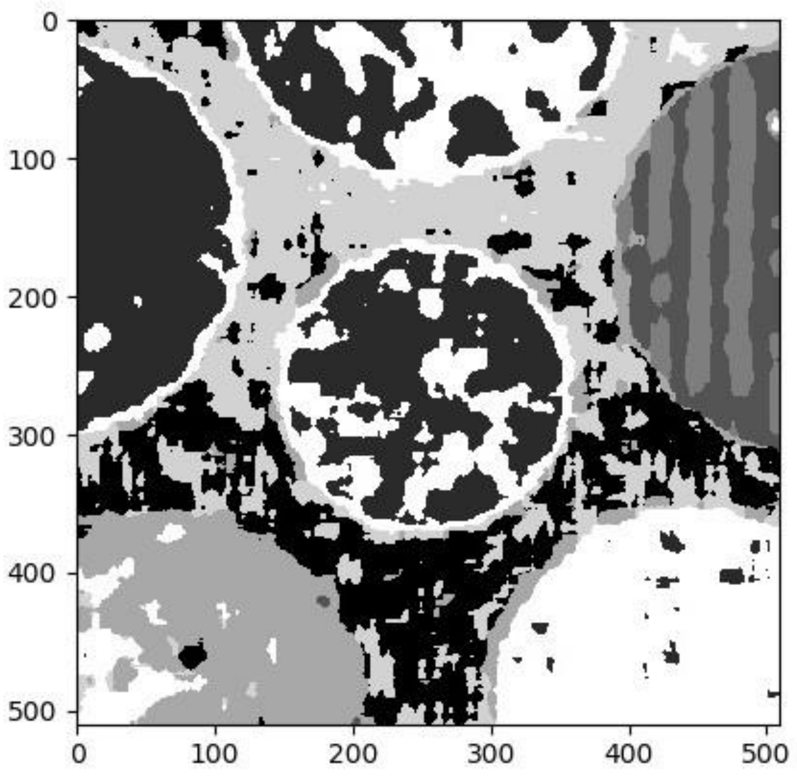


Figure 28 Output Segmented Image for Window Size 17*17

DISCUSSION

1. The best output was obtained for window size 15*15.
2. Segmented Image lacks clarity which depends on the initial random clusters chosen by the k-means algorithm.
3. Segmentation results get better as we increase the window size but until a certain threshold point after which the regions become bulgy and blocky.
4. Trade-off between smoothness of regions and accuracy of segmentation is present. i.e. accuracy decreases with window size but smoothness is better.

FURTHER IMPROVEMENT

Approaches:

1. I implemented 25*25 Laws Filters followed by PCA.

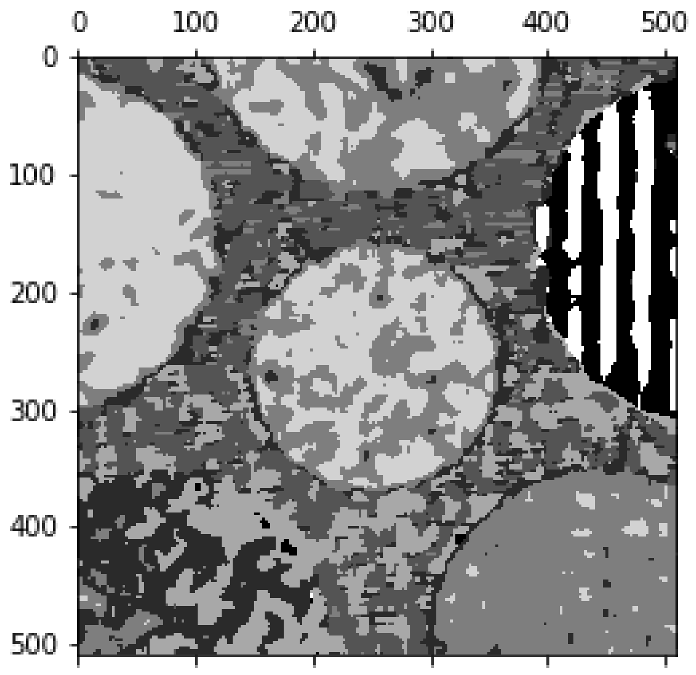


Figure 29 Improved PCA Segmentation output

2. Tried the method of two spirals where first spiral is meant to give coarse output and second finer output, but the segmentation did not improve on implementing this method.
    a. For coarse segmentation, K=3 and window size 15*15 and then performed Fine segmentation on the coarse segmentation output using K=5 and window size 15*15. Output was worse:

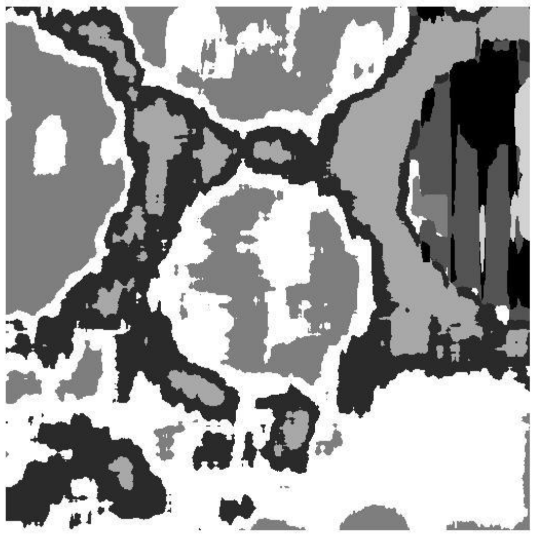


Figure 30 Two Spiral Method Output

3. Implemented Hole-filling algorithm e.g. dilation after segmentation as a post-processing method after implementing PCA using MATLAB.

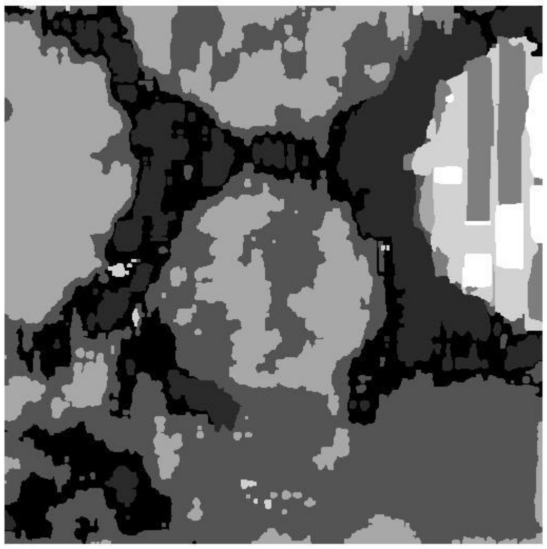


Figure 31 PCA and Post Processing using Hole-filling algorithms

PROBLEM 2: IMAGE FEATURE EXTRACTOR

ABSTRACT AND MOTIVATION

Image features are distinguishing primitive characteristics or attributes of an image. They can either be natural, i.e. defined by visual appearance of an image or artificial, that result from specific manipulations of an image. Natural features include luminance of a region of pixels and gray scale regions of texture.

Image features are important in image segmentation and image classification tasks.

SIFT: SCALE-INVARIANT FEATURE TRANSFORM

Scale-invariant Feature Transform is an algorithm to detect and describe local features in images. The applications are in robotic mapping, object detection, video tracking and Image matching, among others.

The SIFT key features of image objects are extracted from reference images and then stored in database. For object recognition then, every new image is detected by individually comparing each feature from the new image to the database and then features are matched based on Euclidean distance of their feature vectors.

APPROACH AND PROCEDURE

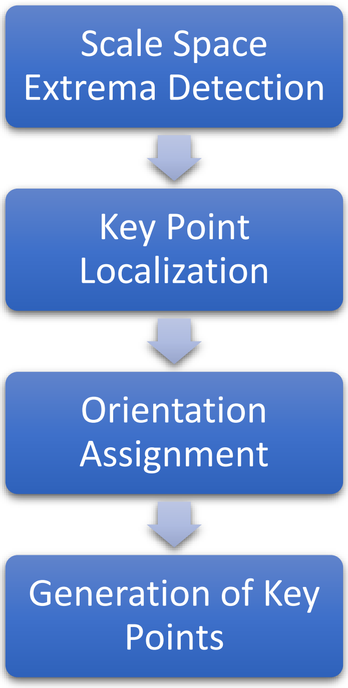


Figure 32 Overview of SIFT algorithm

Basically SIFT includes scale space calculation using Gaussian kernel construction and then computing DoG (Difference of Gaussian). This is followed by locating extrema of DoG and then taylor Series expansion is used to localize sub-pixels. Then we filter out low contrast points using space values and edge responses using Hessian Matrix. This is followed by assignment of key point orientations and creation of histogram of the directions of local gradients. Then we build the key point descriptors with location, orientation and scale features.

Scale Space Extrema Detection:

This is the first step of the algorithm that ensures that the key points are scale invariant by considering images represented on different scales. Then each octave in the scale space is down-sampled by 2 every step by applying Gaussian Filter with different sigma values. Difference of Gaussian (DoG) is then taken after smoothing the images which approximates LoG.

Given a Gaussian blurred image:

$$L(x,y,\sigma) = G(x,y,\sigma) \times I(x,y)$$

Where, $G(x,y,\sigma) = \frac{1}{2\pi\sigma^2} exp^{\frac{-(x^2+y^2)}{\sigma^2}}$

LoG is computationally expensive and hence slower. DoG is calculated using the following equations:

$$D(x,y,\sigma) = L(x,y,k\sigma) - L(x,y,\sigma)$$

$$\frac{\partial G}{\partial \sigma} = \sigma \Delta^2 G \qquad \text{Heat Equation}$$

$$\sigma \Delta^2 G = \frac{\partial G}{\partial \sigma} = \frac{G(x,y,k\sigma) - G(x,y,\sigma)}{k\sigma - \sigma}$$

$$G(x,y,k\sigma) - G(x,y,\sigma) \approx (k-1)\sigma^2 \Delta^2 G$$

$$\text{Typical values}: \sigma = 1.6;\ k = \sqrt{2}$$

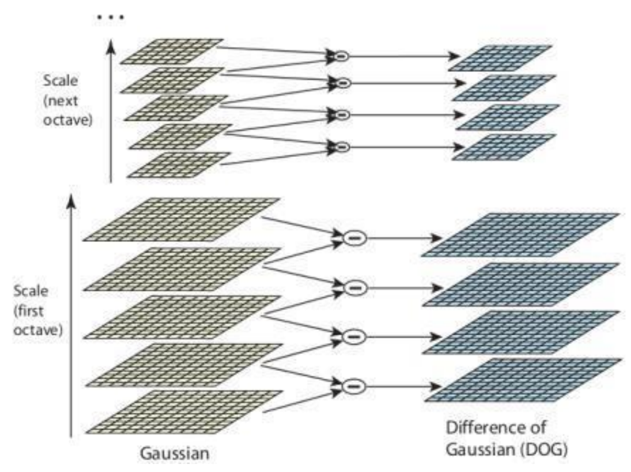


Figure 33 DoG and different scales of the same image

Key Point Localization:

Key points are found using fast approximations which can cause error if the points of interest are weak or edge features. Thus, Taylor series expansion is used to localize the sub-points. First the extrema of DoG are located by scanning each image then the maxima and minima are identified.

Orientation Assignment:

A value is then assigned to each key point found above to obtain the rotational invariance by taking the window of the neighborhood pixels. Neighborhood size is based upon the scale of the point and the orientation of each pixels in the window is weighted by its magnitude. Also, gaussian is weighted circular window is weighted by 1.5 times scaling factor. Maximum orientation is found using window where 80% of the maximum value points are considered to improve the robustness.

Key Point Descriptors:

From the previous descriptor key points found, we find blurred image of nearest scale and sample the points around. Then, the gradients and coordinates are rotated using the orientation previously found. Each region is then divided into several sub-regions with multiple bins. Then the feature descriptor is calculated as a set of histogram of image gradient orientations with a 4*4 pixel window neighborhood. The bins created are taken as a feature vector for a key point. The vector is then normalized so that the illumination effect is taken care of.

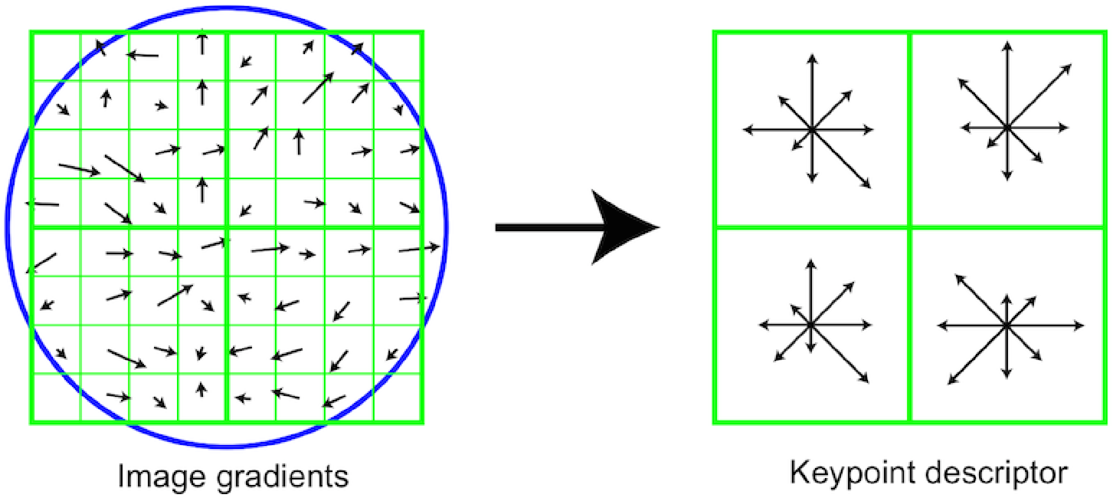


Figure 34 Gradient Orientation and Normalization (SIFT)

ANSWERS:

1. From the paper abstract, the SIFT is robust to what geometric modifications?
   Ans- SIFT is robust to:
   - Scaling
   - Translation
   - Rotation
2. How does SIFT achieves its robustness to each of them?
   Ans-
   Scaling: Locations in the image that are scale invariant is accomplished by searching for stable features in the image across all possible scales implementing scale space which is a continuous function of scale. Also, since DoG is a close approximation to the scale-normalized LoG, normalization of the Laplacian with the factor (sigma)^2 achieves true scale invariance.
   Hence, Scale invariant because of Multi-scale Filtering

   Rotation: Invariance to image rotation is achieved by assigning a consistent orientation to each key point of the image based on the local image properties. The key point descriptors can be represented relative to this consistent orientation and thus are not affected. It is helped by the extrema calculation.

   Translation: The normalization of the key point descriptors that allows for significant shift in gradient positions by creating orientation

histograms over 4*4 sample regions helps the features become translation invariant.
Hence, Rotation and translation invariant through locating the key-points as the extrema of DoG Function and each key-point is assigned a canonical orientation.

3. How does SIFT enhances its robustness to illumination change?
Ans:
The features are partially robust to illumination change because:
   a. Firstly, the vector is normalized to unit length
   b. Change in image contrast which is just each pixel multiplied by a constant multiplies the gradients by the same constant. So, this is canceled by vector normalization.
   c. Change in brightness which is just addition of a constant to each image pixel doesn't affect gradient values because they are computed from the pixel differences.
   d. Non-linear Illumination changes cause a large change in relative magnitudes for gradients but less likely to affect gradient orientations. This is handled by reducing the influence of large gradient magnitudes by thresholding the values in the unit feature vector to be no larger than 0.2, followed by re-normalizing it to unit length.
   e. Thus, overall partial robustness is achieved by thresholding the gradient magnitudes at a value 0.1 times the maximum possible gradient value.

4. What are the advantages that SIFT uses difference of Gaussians (DoG) instead of Laplacian of Gaussians (LoG)?
Ans- Advantages of using DoG are:
   - Speed
   - Computational Complexity is less for DoG since it is an approximation
   - DoG helps in feature enhancement than LoG which helps in edge detection

5. What is the SIFT's output vector size in its original paper?
And: Output Vector Dimension is = 160

ALGORITHM USED FOR SIFT FEATURE EXTRACTION:
1. Read the image and detect Key points using SIFT detector in OpenCV
2. For Detection, define vectors for the key points of each image input

3. Implement the detector class of OpenCV to detect the key points in each image
4. Use drawKeypoints() to show the keypoints of the images

DISCUSSION

The key-points obtained are of the same size, irrespective of their weight or their importance.

The properties of last parameter can be checked using drawKeyPoints() function which also shows the orientation of the key points. The size of the circle of the features denotes the importance of features and their prominence.

SIFT uses DoG as the scale space convolved with Gaussian Filter, Non-Maximum Suppression is used to remove outliers from Hessian matrix. The descriptors are calculated for bins of 128D histogram and a window of 16*16 is used.

IMAGE MATCHING

The task is to find relevant feature matches between two images. Flann based matcher can be used for this purpose of matching key points from the SIFT features calculated. Other than that, Brute Force is a simple and basic matcher. Brute Force takes the descriptor of one feature in the first set and then match with all other features using distance calculation and returns the closest one.

FLANN: Fast Library for Approximate Nearest Neighbors, contains collection of algorithms that are optimized for fast nearest neighbor search using large datasets for high dimensional features. FLANN works faster than Brute Force Matcher for large datasets.

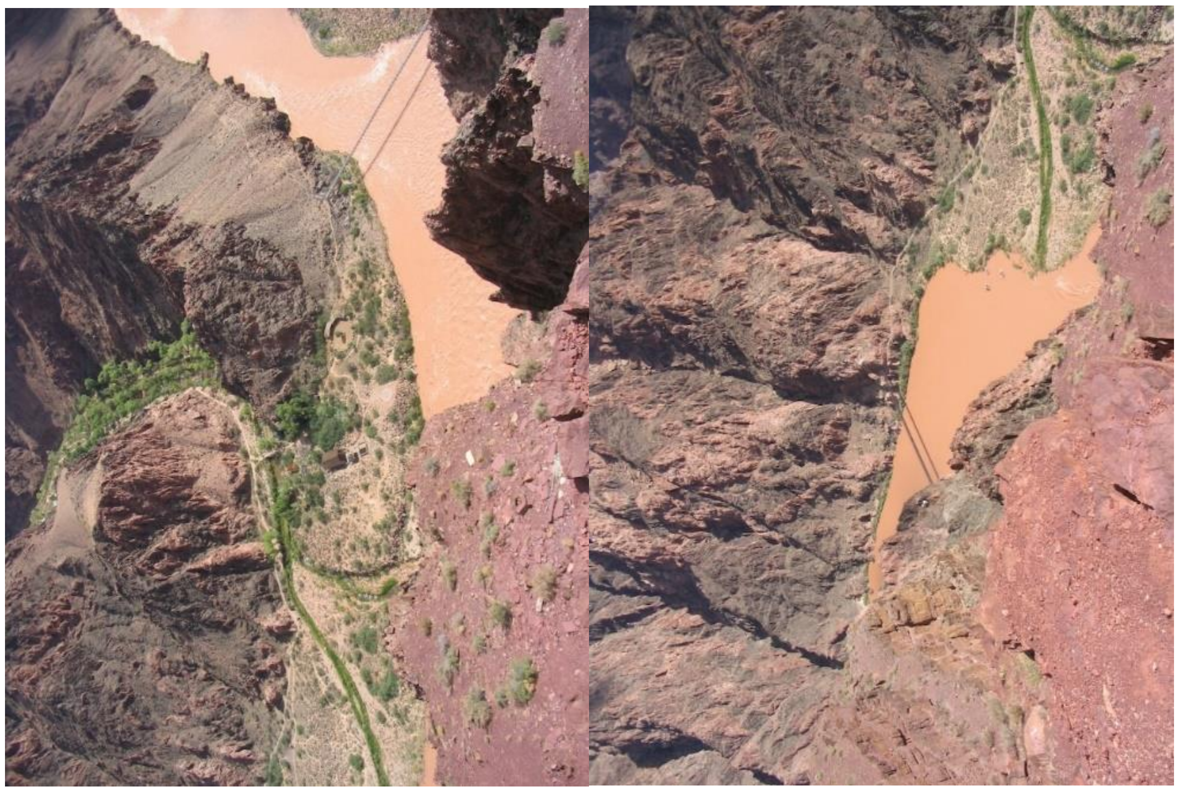
ALGORITHM FOR IMAGE MATCHING

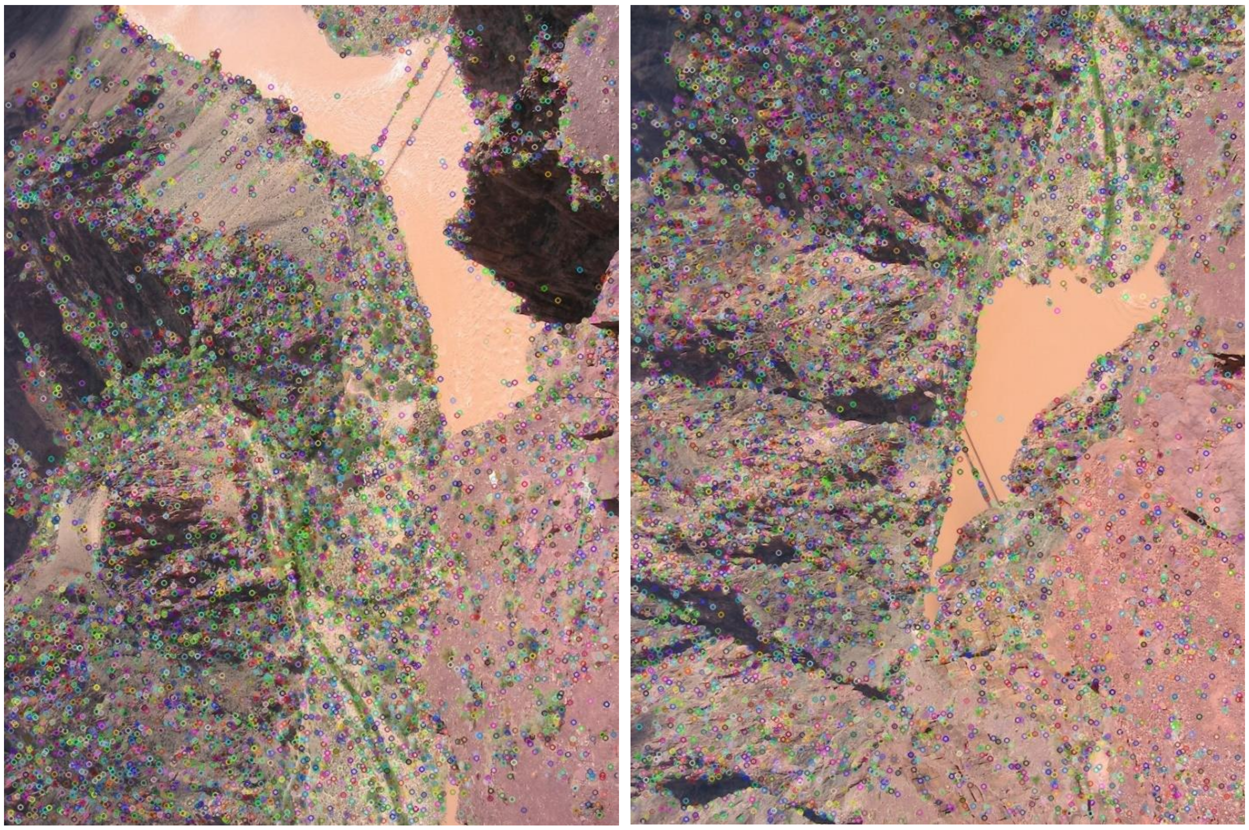
Using SIFT feature extraction using OpenCV followed by using FLANN based feature matcher.

1. Read the two input images.
2. Detect and compute the key points and descriptors using SIFT feature detector and suitable Hessian Parameter.
3. Implement FLANN based matcher/ Brute Force Matcher to match the image key-points
4. Draw the corresponding matching line using drawMatches().
5. Select the best edge i.e. the edge having the maximum Euclidean distance
6. Display the images using key points and matching lines.

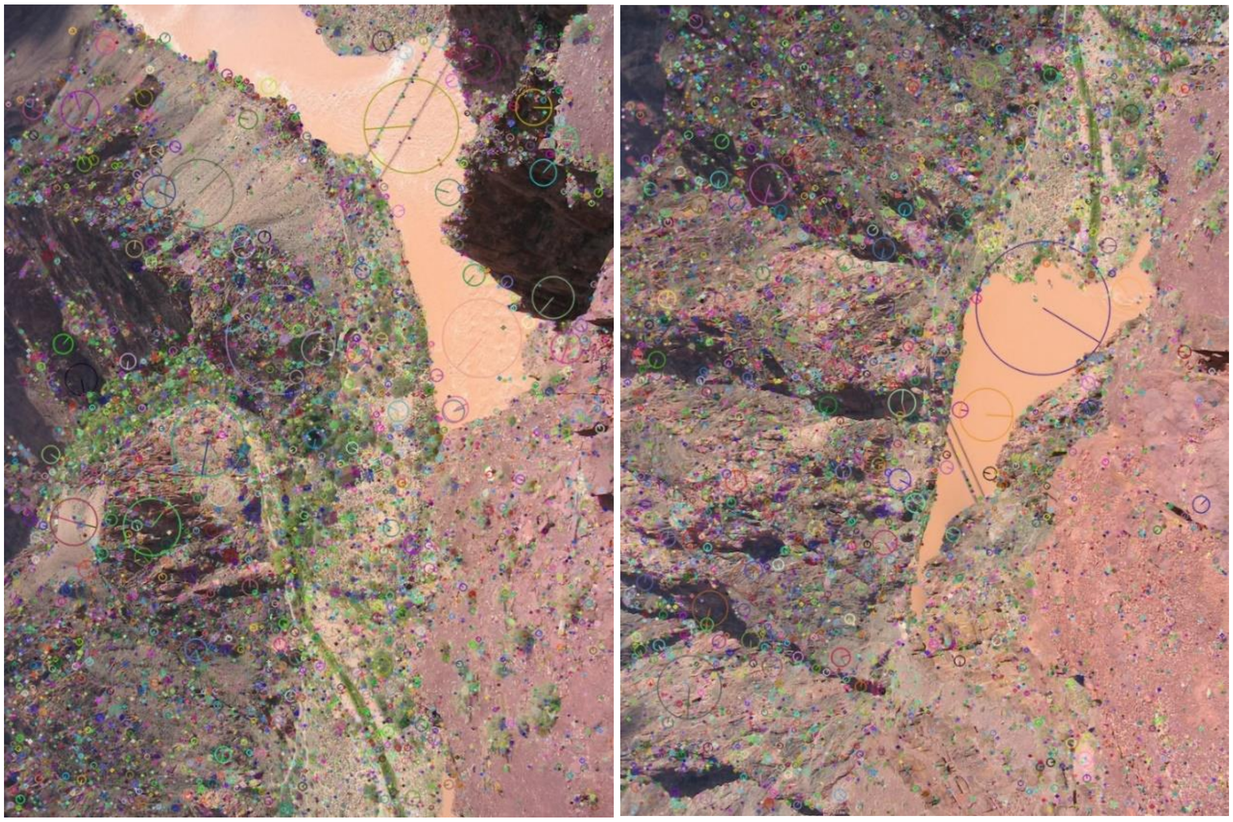
EXPERIMENTAL RESULTS

River 1 and River 2 Input Images





River 1 and River 2 SIFT keypoints



River 1 and River 2 SIFT keypoints with orientations

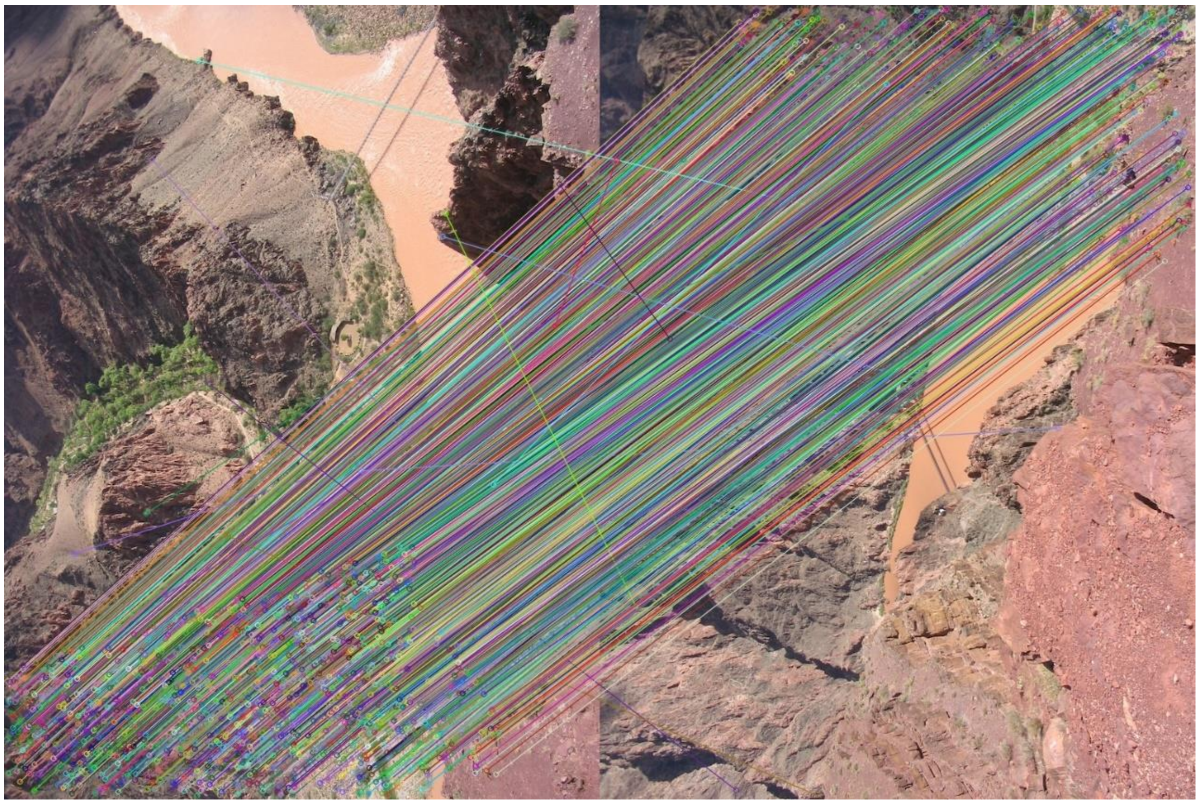


Figure 35 Image Matching using Brute Force Matching

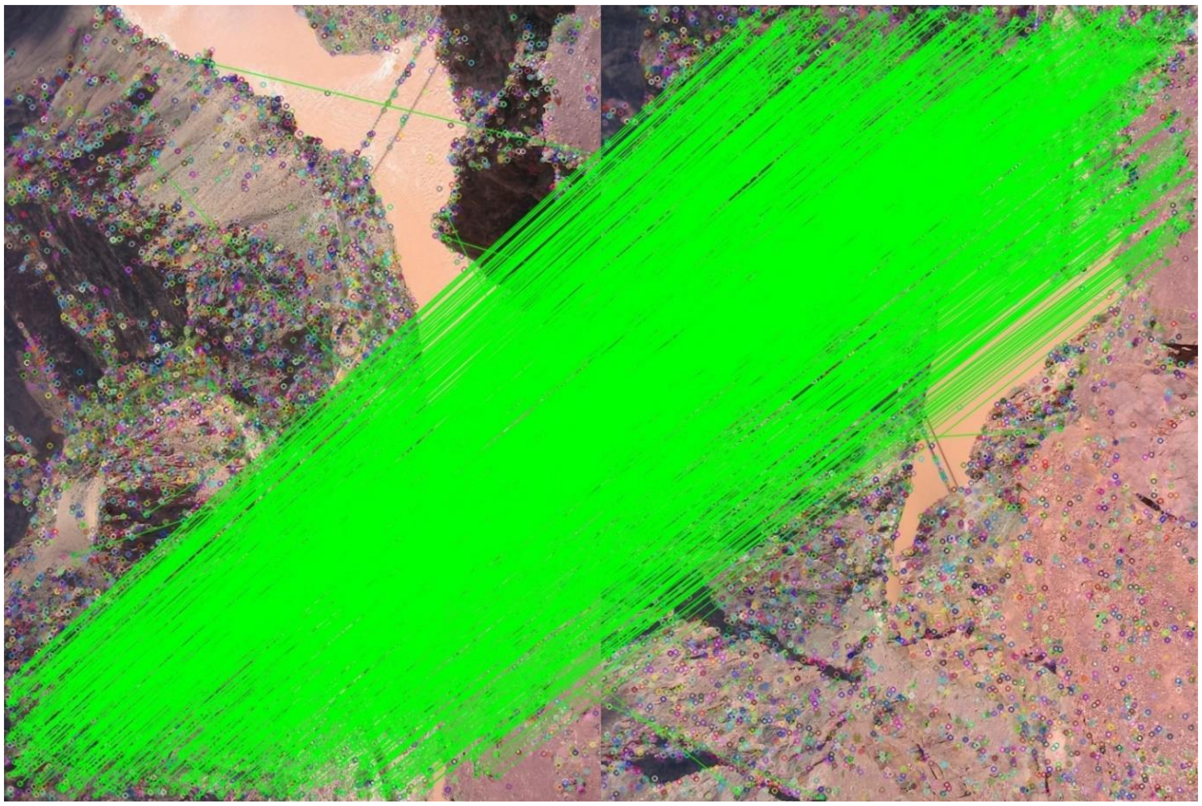


Figure 36 Image Matching using FLANN Matching

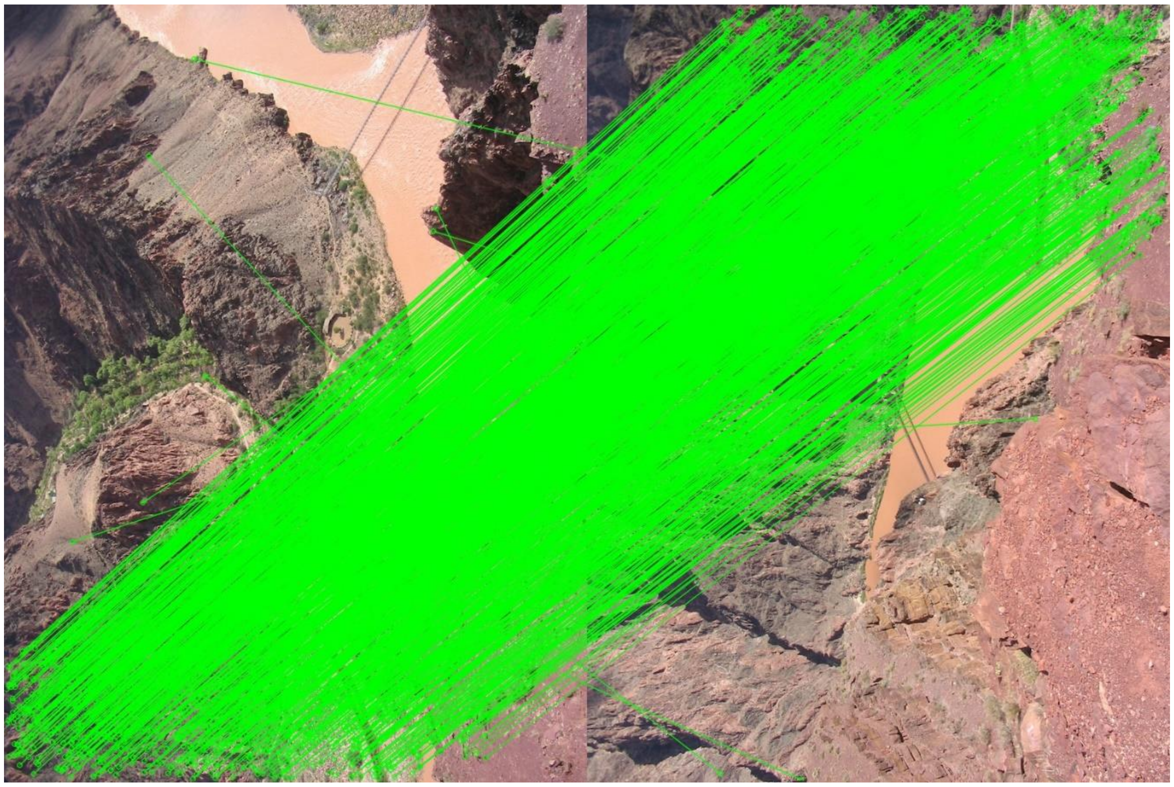


Figure 37 FLANN Matching without Keypoints

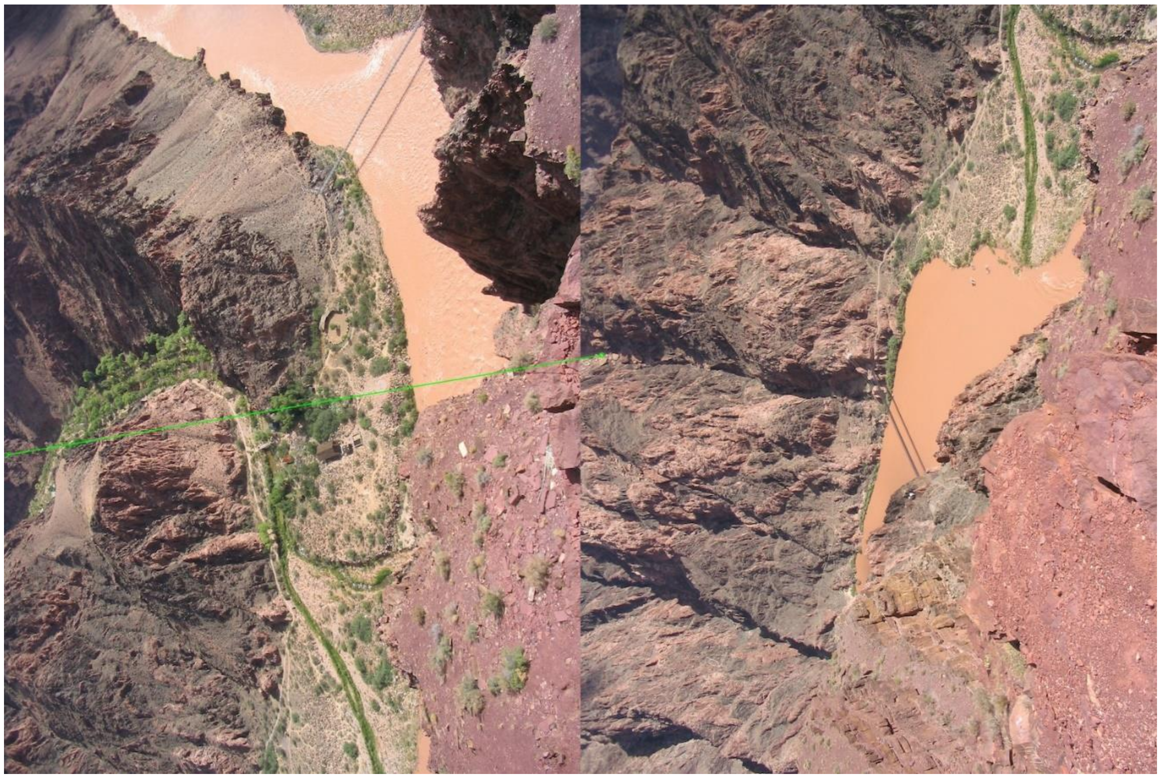


Figure 38 Largest Match Key point using FLANN Matching

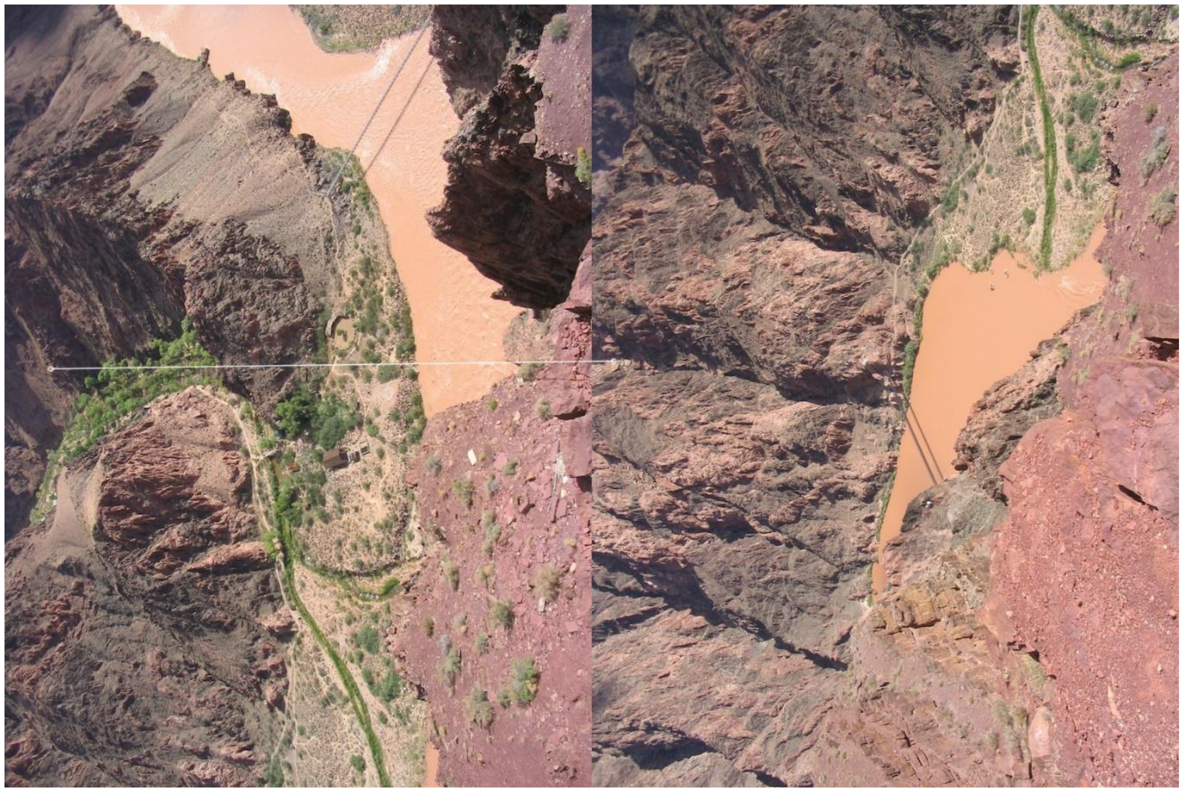


Figure 39 Largest Match Key point using brute Force Matching

DISCUSSION

The output images have the correspondence of top features having ratio greater than 0.75 i.e. if 3 features match, only then the two image points match.

The key points give us an idea of the scale invariant, and rotation invariant points of the image that we can use for image matching and object recognition purposes.

The size of the circle formed by the feature descriptors key points denotes the importance of the features and it’s importance and prominence i.e. the magnitude of the gradient of the key point descriptor.

Also, the orientation of the key points i.e. descriptor angles gives the normalized direction of the gradients. Largest keypoint matching is basically matching between two key points having the largest magnitude and same orientation of the descriptor.

As we can see, the key point features are overall mostly matched correctly for both Brute Force and FLANN matching.

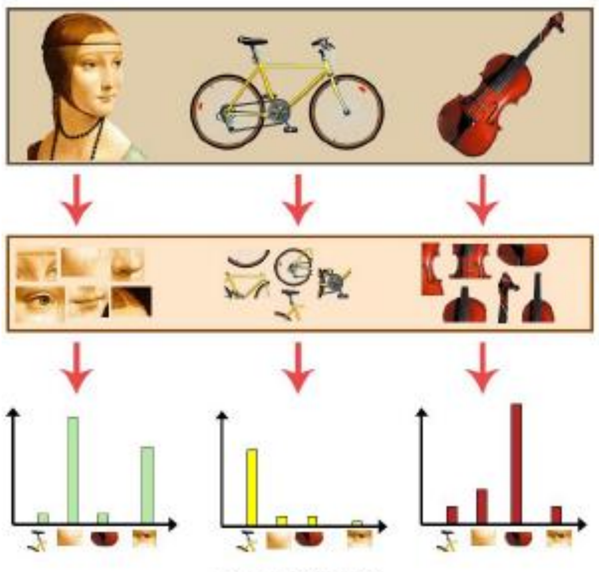
Parameters that define Image matching success or failure:

1. The key points calculated might be far apart because of the distance in dimensions of object that might cause incorrect matching. This can be solved by using bilinear interpolation. Resizing the image will help because feature extraction is invariant to scaling.
2. The orientation of the image and the photography angle might result in improper matching.
3. Since Brute Force matching computes match for every point, it can filter noisy match points too.
4. FLANN works faster than Brute Force Matcher for large datasets.
5. Also, since FLANN uses nearest neighbor's technique it overcomes the issue of irrelevant matches that might result in Brute Force as it sets up the minimum distance value.

BAG OF WORDS (BAG OF VISUAL WORDS)

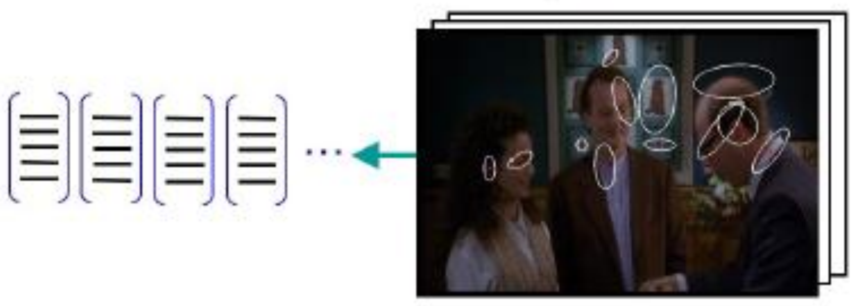
The task is to use the given sample of zeros and ones from MNIST dataset, use these images as training dataset to form a codebook with bin size of 2. Then extract the SIFT feature vector for the image of eight and show histogram of the BoW. Bag of words is the vector of frequency and count of words expressed as a histogram of each word.

Idea of bag of words is to be able to represent an image into a set of features that consist of key points and descriptors. Key-points are the rotation and scale invariant points in an image. Descriptor is basically the description of the stand-out key points. The key-points and descriptors are used to create vocabularies and eventually represent image as a frequency histogram of image features.

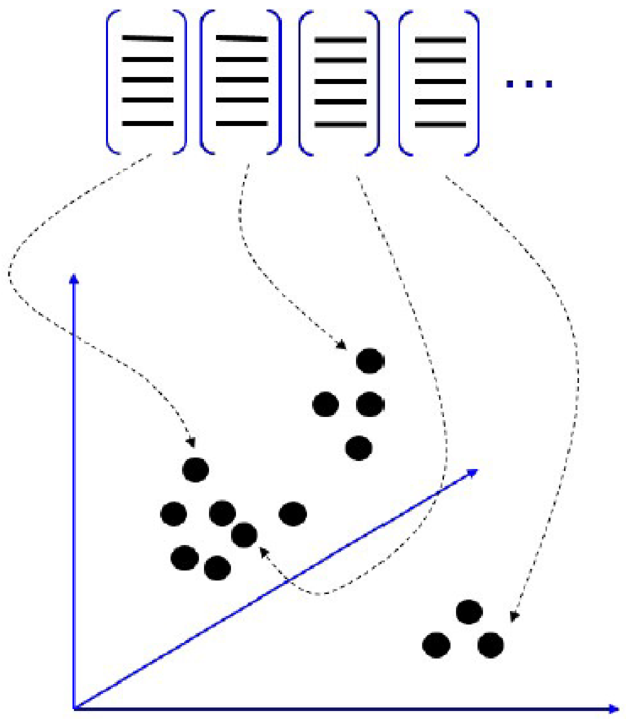

Histogram of visual words

Building Bag of Visual Words:

1. Detect features and extract the descriptors from each image in the dataset using feature extractor algorithms e.g. SIFT, KAZE, etc.


Detecting features and extracting descriptor

2. Make clusters from the descriptors using Clustering algorithms e.g. K-means. Then the center of each cluster is a part of the vocabulary of the visual dictionary.



3. Frequency histogram is then made for each image from the vocabularies and frequency of vocabularies. These vocabularies are bag of visual words (BOVW).

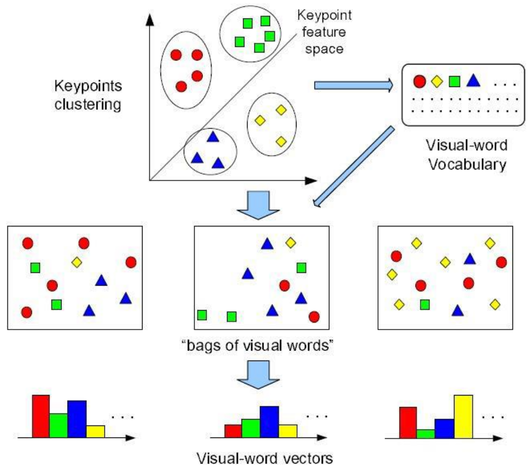


Figure 40 Bag of words Algorithm

ALGORITHM FOR BAG OF WORDS:

1. Extract local features of all the input zeros and ones images using detect and Compute SIFT feature detector and suitable Hessian parameter.

2. Create a BoW trainer for 2 bins
3. Add the descriptors of the zeros and ones images to the Bag of Words to create vocabulary or codebook
4. Apply the K-means clustering method on the codebook to cluster it into 2 clusters or Codebook having a final vocabulary 2*128 and descriptors will be <hessian parameter>*128
5. Find the Euclidean distance of each of the hessian parameter points for each descriptor from the 2 vocabularies and store the index of the minimum value, incrementing the respective data index by 1.
6. Save the data and plot histogram of each descriptors
7. To verify what does Eight image match to, we have to find codebook for the descriptors of the eight images, plot the histogram and check which does it match more to, zeros or ones.
8. To check matching, I do error calculation by subtracting individual bin values of zeros and ones from eight.
9. Less error means higher matching.
10. Further, I did Histogram Intersection calculation by calculating ratio of features of zeros and ones in the vocabulary.

EXPERIMENTAL RESULTS

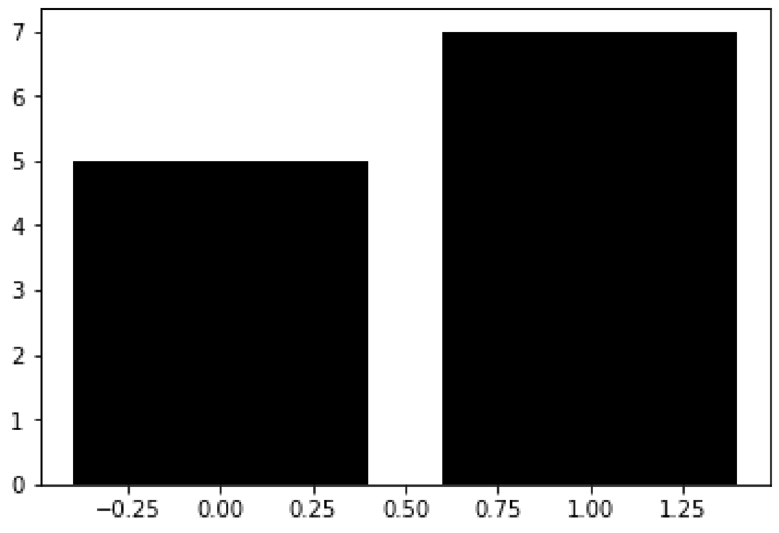


Figure 41 Histogram for Descriptors of Eight

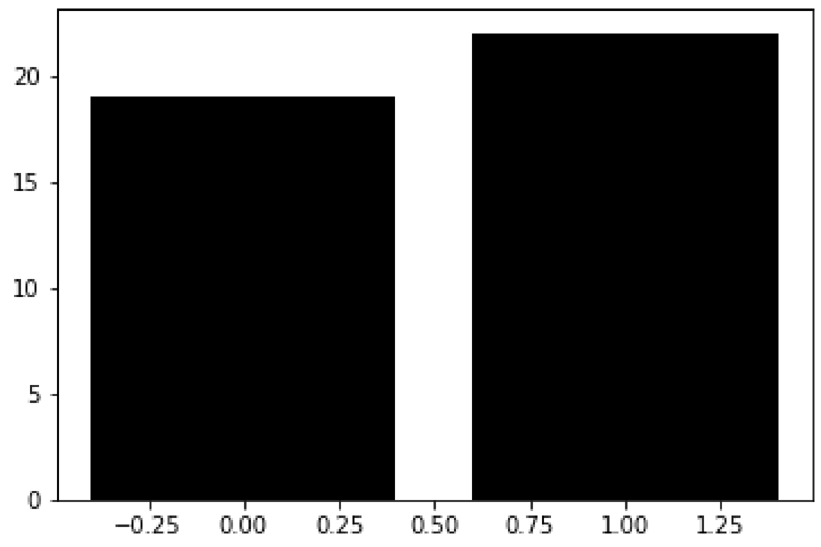


Figure 42 Histogram for Descriptors of Zeros

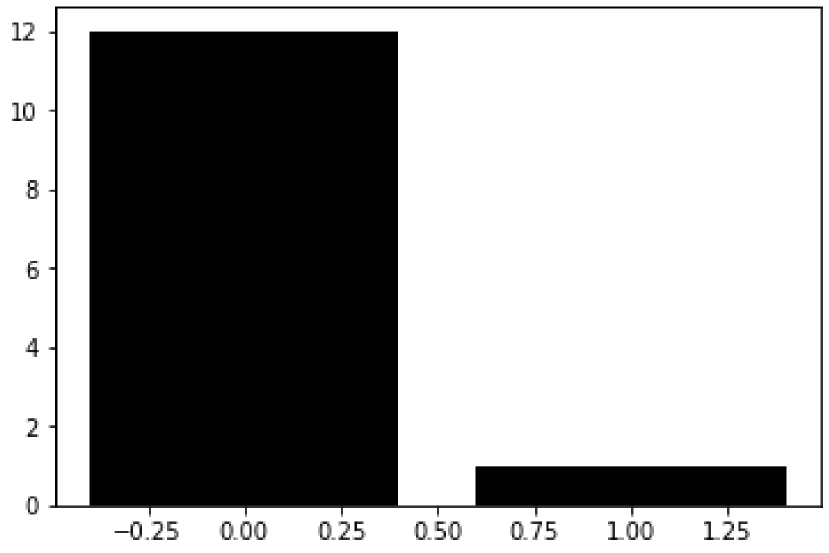


Figure 43 Histogram for Descriptors of Ones

DISCUSSION

1. Trained the codebook and used it to predict matching of eight.
2. With different iterations of K-means, the histogram of ones and zeros varies and sometimes Eight is more like Zeros and sometimes it is more like Ones. This occurs because SIFT features are invariant to geometric modifications, the features extracted may come from different orientations.

3. Firstly, I chose to drop the image which gave zero descriptors. This gave the same results as above.
4. As we can observe from the histograms, eight resembles Zero more than it does Ones which can be visually verified.
5. Also, the histogram intersection of zeros histogram with eight is more than for ones, hence Zeros is more like Eight.
6. This must be because zeros have more curves than ones that resemble eight more i.e. the descriptors of zero have more similar features with eight's descriptors than one's descriptors.
7. Secondly, I rescaled the image till it gave non-zero descriptors. Then eight seems more like ones than zeros. This happens because SIFT features extracted may have come originally from scaling artifacts.
8. There isn't enough training data for proper clustering classification of eight.
9. Dimension, orientation, and the quality of image plays important role in this matching.
10. Quality of image is important in matching. When saving .raw as .jpg at lower quality, the eight was getting classified as 1, but saving at higher quality classified it as 0 which is more true.

REFERENCES:

1. https://en.wikipedia.org/wiki/Scale-invariant_feature_transform
2. W. K. Pratt, "Geometrical Image Modification," , 2002. [Online]. Available: http://onlinelibrary.wiley.com/doi/10.1002/0471221325.ch13/summary. [Accessed 1 3 2019].
3. Class Notes and assignments
4. https://www.cs.ubc.ca/~lowe/papers/iccv99.pdf
5. https://docs.opencv.org/3.3.0/dc/dc3/tutorial_py_matcher.html
6. https://en.wikipedia.org/wiki/Image_segmentation

7. *Lindeberg, Tony (2012). "Scale invariant feature transform". Scholarpedia.* ***7*** *(5): 10491. doi:10.4249/scholarpedia.10491.*
8. Digital Image Processing, Second Edition, Rafael Gonzalez, Richard E Woods, Pearson Education
9. Digital Image Processing, Fourth Edition, William K. Pratt, Wiley-Interscience Publication
10. OpenCV documentation from www.opencv.org for OpenCV installation and reference source codes
11. Color code for implementation in segmentation problem from http://www.rapidtables.com/web/color/RGB_Color.htm
12. https://gist.github.com/betheazdavida
13. https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_matcher/py_matcher.html