EE 569: Digital Image Processing
Homework #4

By
Brinal Chimanlal Bheda
USC ID - 8470655102
bbheda@usc.edu
Spring 2019

Issue date: 03/04/2019
Submission Date: 03/19/2019

Problem 1: Texture Analysis

(a) Texture Classification

I. Abstract and Motivation

Textures are important part of the images as they carry information for helping us differentiate between various objects present in the given picture. In texture analysis, we try to understand the textures. Texture is one of the very early day image processing problem. The reason for this is because in digital processing, pictures were taken from the airplane and see the surface of the earth looking at different kinds of textures say ocean, land, city, rural areas, forests, desert.

Kenneth designed a set of filters that work in parallel called filter banks and got very good results for the texture segmentation. Input is scanned by 3x3 or 5x5 filters present in the filter bank and we get the response map corresponding to each filter. Thus, we get multiple response maps and operate on them to get the response. Similar to neural nets in these days, however they are multi-layer. The filters which we use above is called Laws filters. The common issue with these filters is data independent. But now-a-days, in neural network, filters are driven depending on the data, hence, they are data driven. Also, this is a single stage filtering, not multi-stage like neural nets. Now, this data driven multi-stage filters are used for general image processing applications.

Texture is quasi-periodic patterns. It is not totally smooth or random like noise. Given the texture and check which texture class it belongs to is the texture classification problem. Given the texture and have to find the boundary is the texture segmentation problem. This is different kind of texture processing to do called as texture analysis.

Three applications:

a. Texture Classification – Have n texture types corresponding to n class types and given texture X, check to see in what class this X texture belongs.
b. Texture Segmentation – Have one image which contains 2 textures, segmentation is to determine the boundary. Once get the results, use different grayscales to represent the region, however, we may not get smooth boundary. This is done to display the segmented results.
c. Texture Synthesis – Say there is a texture class A, having samples #1, #2, #3... and so on. Now synthesize a texture from these samples similar in the class. This is called as the generative model. Generate new samples but belonging to the same class. New versions of neural network models have also come up.

One such model is called VAE: Variational Auto Encoder, other one is called GAN: Generative Adversary Network. These are the state of art image generation tools.

One of the issues these days is texture representation. The mathematical tool used before was Wavelets. The major issue is that wavelet is linear representation and it is only single scale. There are two transforms – Saak (Subspace Approximation using Augmented Kernels) and Saab (Subspace Approximation using Adjusted Bias). It is multi-layer (multi-stage filter) and data driven/dependent. These are very new tools and inspired by the neural network model.

Laws filter designed the filter bank, the advantage of this filter bank is explained below. Say we have just one pixel, it is hard to determine the texture from it, so we increase the window size and say take 3x3 window, it is still impossible to tell what the texture it is. This is because it is too small. A reasonable size of the picture is needed to determine the texture. The size of the image chosen carries a lot of information. It is important to see how large the image is, this is called receptive field.

Image understanding – A suitable window of the image. The size and location of the receptive field is important. Hence, define a flexible size such that the receptive field can be larger or smaller. This can be reached by using multi-layer, which neural network captures. The location of the receptive field is important for the object detection. Laws filter is not so powerful as it considers only 3x3 or 5x5 window size.

In Laws filter, he considered 3 kinds of 1D filter.
L3: $\frac{1}{6}$ (1,2,1) → L corresponds to the level or brightness. This is a filter with the weight (1,2,1). This is a 1D template. Multiply and add the weights and generate response to the middle one. L denotes local average. 3 stands for 3 co-efficients.
E3: $\frac{1}{2}$ (-1,0,1) → E denotes edge detector. This is a 1D template.
S3: $\frac{1}{2}$ (-1,2,-1) → S means spot, when detect a spot.
In order to get 2D, do the tensor product.
The above 1D filter are applied to the horizontal direction.

$$(L3)^T: \frac{1}{6}\begin{bmatrix}1\\2\\1\end{bmatrix}$$

$$(E3)^T: \frac{1}{2}\begin{bmatrix}1\\0\\-1\end{bmatrix}$$

$(S3)^T: \frac{1}{2}\begin{bmatrix}-1\\2\\-1\end{bmatrix}$

The above 1D filter are applied to the vertical direction.

Tensor product gives 2D result combining the above 3 choices in horizontal and 3 choices in vertical direction, so get the nine 2D filters. If texture is different locally, then this method works well, otherwise the size of window is too small. In frequency domain response, the L3 gives low pass filter, the E3 gives band pass filter and the S3 gives high pass filter. So we decompose the 2D frequency regions into 9 frequency band channels.

Another way of looking at Laws filter is shown below. One is we consider the frequency decomposition and frequency band. Another viewpoint to Laws filter decomposition is called Subspace representation. The 3x3 image patch in spatial domain is represented by flattened 9D vector. We have 9-unit vectors, each representing a pixel in the 3x3 patch. The 3x3 patch is a linear combination of these 9 vectors, wherein the pixel values are weights multiplied each with the unit vectors. This is spatial domain representation.

Now we change the representation, we use Laws filters as the representation. We have Laws filter #1, #2, …. and so on. Represent into 9 1D filter. In this 1D filter, each unit is a Laws filter. For smooth image, the low frequency bands have more energy than others. But if there are some textures, those filters have more energy than other filters. So the 9 filters do not play the same role. Some filters are more important than others and this leads to Principal Component Analysis (PCA). So we change the space to find several dimension which has more stronger energy.

Principal Component Analysis (PCA)

In the signal space, there are different kind of representation bases. But the projection of the bases is more random or uniform. But some places give better representation. Say if have smooth signal, do Fourier transform to eliminate high frequency. As we use cosine transform for image compression, to keep the low frequency band. The pixel with highest energy which is more important becomes the first principal component, and the second highest energy becomes the second principal component. Order the energies in different sub bands. We use this method to approximate certain texture, because energy distribution in the 9 bands is non-uniform. However, doing this in spatial domain is not possible. Thus, we look at Laws filter from two angles, one is look at the frequency domain decomposition and another domain is looking at the change of space representation.

Texture Classification

Collect feature vector for each texture type. Suppose have Class A so get few training samples. Then apply 3x3 or 5x5 Laws filters to all patches scanning the whole image. Collect the Laws filter response. Each patch gives one response. Write the responses in 9D or 25D vector for the center pixel. Collect them and calculate the energy for each response. Compute average energy of all patches. The response can be positive or negative. If there is strong positive response i.e. in phase or strong negative response i.e. out of phase, then there is strong correlation with the filter. For 9 or 25 Laws filter, we have 9 or 25 averaged energies. This is the feature vector for texture class A, similarly repeat for texture B, C… and so on. Suppose there is texture X, we find the feature vector for texture class X and find the Euclidean distance between A and X in 9D or 25D space and normalize the distance according to the variance of the energy samples. Choose the one with the smallest distance and that is classification result. Also, can use the Mahalanobis distance to calculate.

Say, for texture classification, there are two texture class A and B. Energy of a channel (filter) is not deterministic but random. They are all from same texture class. Consider the sample mean and sample variance. Then compute the corresponding average. Use the nearest neighbor rule. It also depends on the kind of texture, if the texture is tricky, similar resolution, similar granularity then the results comes poor. Number of textures class types considered for comparison also defines the problem complexity. Mostly there are large number of class of texture types chosen for computation because of the large dataset chosen. Texture classification problem is a supervised learning as the labels for each texture class are known. The problem provides the images and their class labels.

For texture segmentation, it is more complicated as one image itself consists of the different types of textures. It is challenging as at the corners or edges of different texture types it is difficult to determine the texture class being different. And thus, we use unsupervised learning. This is related to the clustering as we have in 9D or 25D space different textures types and they need to make different clusters.

(Source: A comparative study of efficient initialization methods for the k-means clustering algorithm by M. Emre Celebi, Hassan A. Kingravi, Patricio A. Vela)

(Source: Data clustering: 50 years beyond K-means by Anil K. Jain, Michigan State University)

k-means clustering

Try different values of k and use the one which gives the best solution. k is the target cluster number. Try multiple k and choose the one which is more reasonable. Now, given the k, how to do k-means clustering? For initialization, choose k random seeds. For iteration, we do generalized Lloyd iteration. This consists of 2 steps: 1. Grouping based on seed and the sample distance. Partition the cluster into k domain. 2. For each cluster, find the new centroid and use it as the new seed. Repeat the above process until it converges. There is no unique result as it depends what initial we choose and start from. The final clustering result is highly dependent on the initial seed distribution. Now we use k-means++ software package, it works better when k is larger.

(Source: Question PDF – HW #4)

In this problem, I will implement texture analysis and segmentation algorithms based on the 5x5 Laws filters constructed by the tensor product of the five 1D kernels shown below,

Tabel 1.1. 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]

(Source: Discussion lecture dated 03/05/2019)

Texture classification/segmentation

- Feature Extraction (Law's filters)
- Feature Dimension Reduction (PCA)
- Classifier (K-means)

Part A) Laws filter is used to do the texture classification. Classify the 12 different textures into 4 types of textures given: bark, straw, brick and bubbles.

Part B) This is quite same as part A, but segment one input image given into 7 different textures.

- Texture Classification/Segmentation

What/Where are the texture?

This is a very important topic because there a lot of applications like object classification and object segmentation is dependent on such local spatial variance of intensity or color. That is why texture analysis is very important here.

(Source: Question PDF – HW #4)

Twelve texture images from texture1.raw to texture12.raw (size 128x128) are provided for the texture classification task in this part. Samples of these images are shown below:

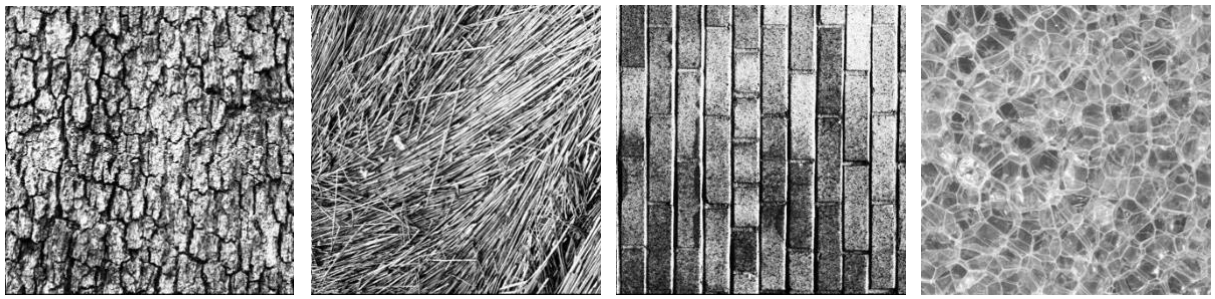


Figure shows four types of textures: bark, straw, brick, and bubbles

(Source: http://sipi.usc.edu/database/database.php?volume=textures&image=19#top)

II. Approach and Procedure

(Source: Discussion lecture dated 03/05/2019)

Signal-processing-based algorithms:

- The Laws Algorithm
- Use texture filters applied to the image to create filtered images from which texture features are computed

We use filters and generate some response and analyze the response. So, the filter used here is Laws filter. The whole procedure is shown in the diagram above. The overall process is we input the texture images and apply the filter to get the response and then we try to analyze all these responses which we call features and then we can get the results.

The Laws Algorithm

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]

5 different kernels for 5x5 Laws filters are used: Level – to detect level or brightness and average the pixel values, Edge – to detect edge, Spot – to detect spot, Wave – to detect wave, Ripple – to detect ripple. Based on these 5 basic kernels, generate 25 different filters. This is basically a vector product that is column vector times the row vector.

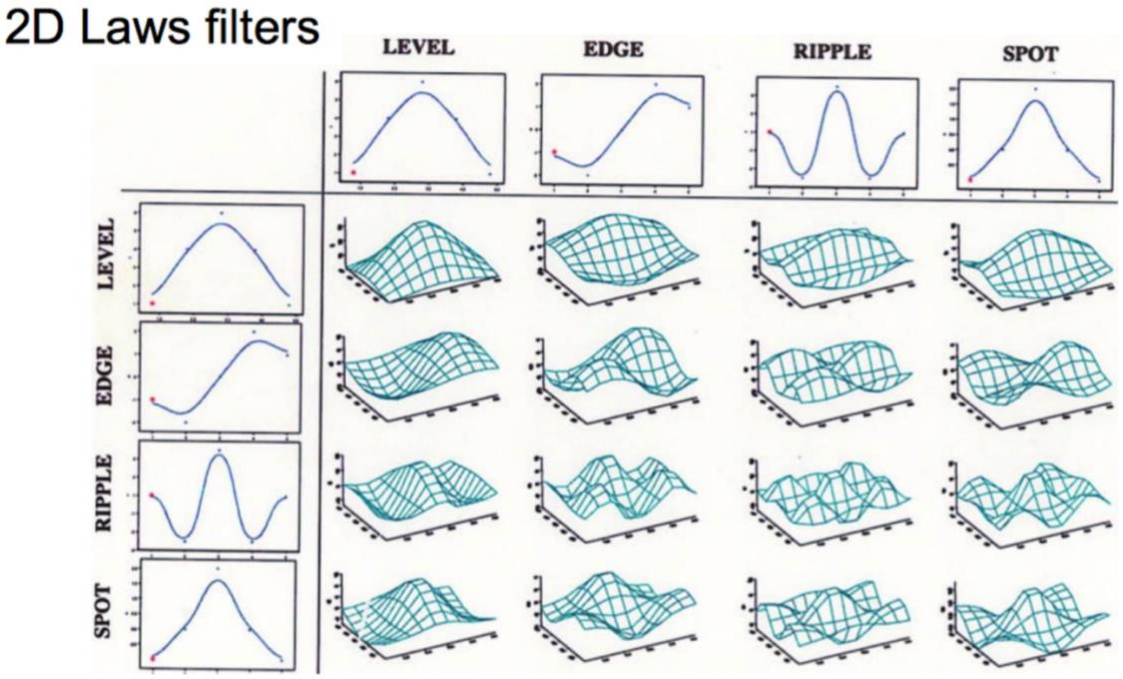
2D Laws filters

- An example: E5L5

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

Each filter is convolved with another to generate the 25 tensor products. This is the way how we create 25 different filters. Before performing this operation, we need to do boundary extension by replicating the pixels being used.

Below shown is how the filters look like. Say we have 4 kernels and 16 filters.



For level, along the first row, every filter is focused on the level performance. We can extract feature correspondingly by these different filters.

Since the input is 12 different images, so for each image do the following steps:

Subtract the image mean from the input image because there are some cases like may be both are straw images but say one is darker and other is lighter so after we extract the feature, in the feature space these two vectors may disport away far from each other. We have to avoid such a case by reducing the illumination effects. Find the average value of all the pixel values in the input image and subtract the average found from the original input image pixel values. Also, by doing this, it helps to remove the DC component in the image and along with it the redundant information from the image is also removed. This also prevents the feature vector from being dominated by high energy value.

Then apply the 25 filters to get the 25 filtered images and obtain the local energies. Example is shown below:

Procedures (Prob1a)

For each texture image:

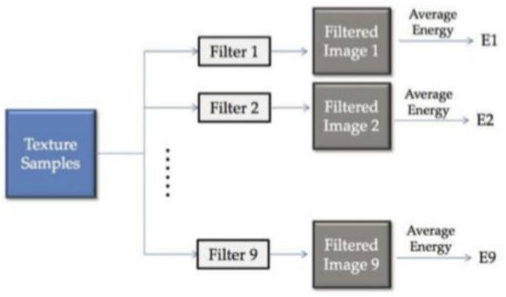
- Subtract image mean (reduce illumination effects)
- Apply 25 5x5 Laws filters to get 25 filtered images
- Average energy to form a 25-D feature vector
- (Define 15-D feature vector as follows (replace each pair with average))

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

→Feature Set (n samples x m feature dimensions)

For each dim

- Feature normalization



The blue box is the input image. There are 25 filters. Applying these filters, we get the response image and based on each response image, calculate the average energy to form a 25-D vector for that certain texture image. This means we use one vector to represent each texture image. There are a lot of filters which are symmetric like L5E5/E5L5. They are quite similar so if we want to reduce the dimension, we sum up the two dimensions together to form a new one so that we can reduce the dimension from 25 to 15.

The calculation for the energy (feature) of the image is shown by the formula below:

$$\text{Energy} = \sum_{x=1}^{Row} \sum_{y=1}^{Col} (\text{abs}(f(x,y)))\text{^}2$$

Where, f(x,y) = Pixel Intensity,
Row = Number of Rows,
Col = Number of Columns

Each image has a vector. At the end, we have a set of vectors which is called as the feature matrix. For that matrix, for each dimension we have to do feature normalization. Usually we subtract the mean and divide it by the standard deviation. Because may be sometimes one dimension is so dominant that it's value is like several hundred but the other dimension is very small like 0.9 or something. Hence, we need to reduce this imbalance. Thus, this part was the feature extraction.

Now, we need to do feature dimension reduction. The purpose to do that is that we have high dimensional space i.e. 25-D space. Data points would be sparse in this high dimensional space and so the calculation is not reliable for the modelling and the computational expense also would be very high. Based on these reasons, we want to reduce the dimension for our features.

Curse of dimensionality

- Data points sparse with respect to dimensionality
- Unreliable modeling
- Computationally expensive

(Source: https://medium.com/greyatom/why-how-and-when-to-scale-your-features-4b30ab09db5e)

(Source: https://stats.stackexchange.com/questions/29781/when-conducting-multiple-regression-when-should-you-center-your-predictor-varia)

Feature Vector Normalization

1- Min-max scaling retains the original distribution of scores except for a scaling factor and transforms all the scores into a common range [0, 1]. It is useful when dealing with features with hard boundaries.

2- Standardization, which is calculated using the arithmetic mean and standard deviation of the given data, changes data distribution to zero-mean, unit-variance standard Gaussian. Standardization can be used for algorithms that assumes zero centric data.

(Source: https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html)

Several methods are available for this, but we use the Principal Component Analysis (PCA) in Linear Algebra.

Two common methods

- Feature selection
- Feature reduction -> Principal component analysis (PCA)

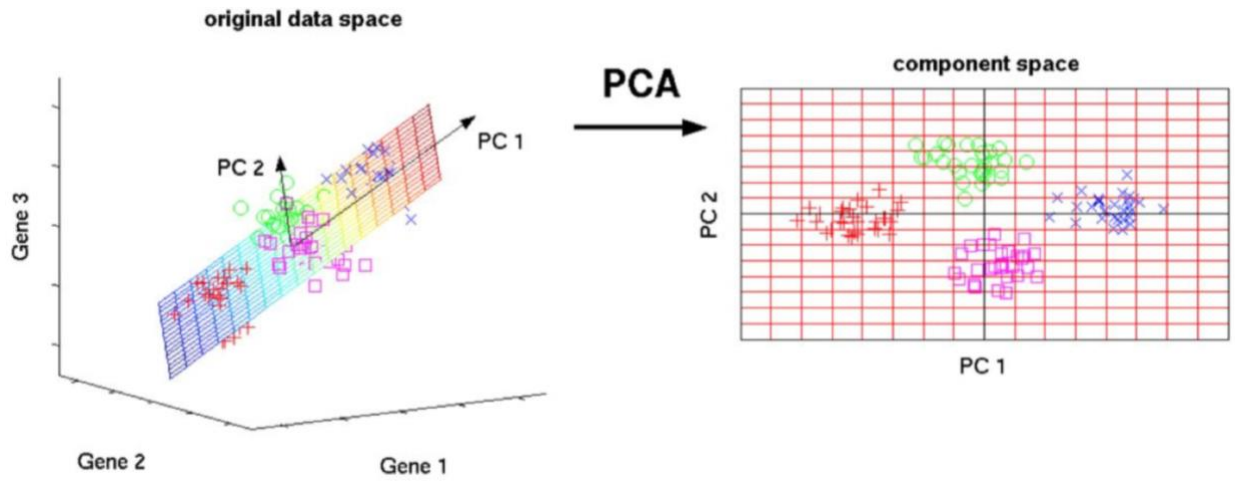
It is very important to do PCA on high dimensional data, by using PCA, better data representation is possible.

Retain significant eigenvalues (or singular values)

- ✓ Usually a high dimensional feature matrix has lot of redundancy
- ✓ As a result, its covariance matrix has many zeros (or close to zero) eigenvalues (or singular values)
- ✓ Such values can be discarded safely

Since orthogonal transform, resulting transformation gives uncorrelated features

- ✓ Why is uncorrelated features desirable?



I used build-in functions. The links are provided below:

You can use built-in PCA Object/function

OpenCV
https://docs.opencv.org/ref/2.4/d3/d8d/classcv_1_1PCA.html

Matlab
https://www.mathworks.com/help/stats/pca.html

The mathematical procedure for PCA using singular value decomposition is shown below:

Compute PCA via SVD

- Consider feature matrix X of size nxm
  - n is the number of features
  - m is the number of data points
  - Let Xi denotes the ith column of matrix X
- Compute Mean mX (vector of size n x 1)
  - $mX = (\Sigma i \in [1,m]\ Xi\ )/\ m$
- Compute zero mean feature matrix ZX of size nxm
  - $\forall\ i \in [1,m]$, $ZXi=Xi-mX$
  - ZX is also of size n x m

Compute PCA via SVD

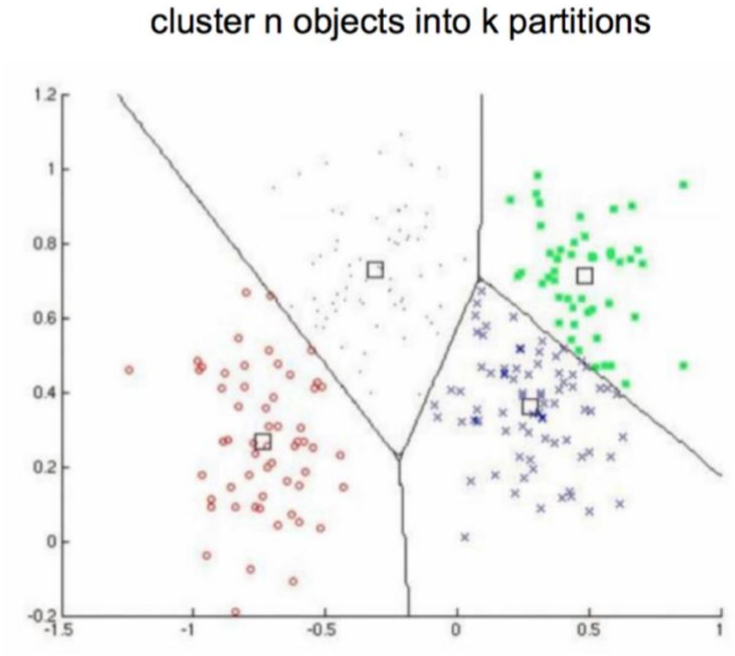
- Compute SVD of ZX
  - $ZX = U \cdot S \cdot VT$
  - U is of size nxn, S is of size nxm and V is of size mxm
- Dimensionality Reduction
  - Equivalent to discarding small eigen/singular values
  - Assume S has singular values in descending order of magnitude
  - Then, if we want to reduce dimensionality from n to n'
  - We just retain first n' singular values. This corresponds to retaining first n' columns of U, S

Compute PCA via SVD

- Compute reduced transformation matrix UR
  - $\forall\ i \in [1,n'] \ni n' \leq n$, URi =Ui
  - UR is of size nxn'
- Finally, transformation step; let RX be dimensionality reduced matrix, then
  - $RX = URT \cdot ZX$
  - RX is of size n'xm
- Use RX instead of X

K-means

A type of unsupervised data analysis algorithm



For the classification part, I used K-means. This is a type of unsupervised data analysis algorithm, means we do not need any information about labels. We do implement the procedure to get the feature and then analyze the final results. For example below, the space is the feature space i.e. the 25-D high dimensional space. All the samples are spread out in this space. We divide them or cluster them into several parts or groups based on their similarity. The similarity here is defined by the distance. Usually, we do the Euclidean distance defined below or use the Mahalanobis distance or can use some other distance weighted by Gaussian weights.

Distance measure

- determine how the similarity of two elements
- influence the shape of the clusters

Euclidean distance

$$\mathrm{d}(\mathbf{p},\mathbf{q}) = \mathrm{d}(\mathbf{q},\mathbf{p}) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + \cdots + (q_n - p_n)^2}$$

$$= \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2}.$$

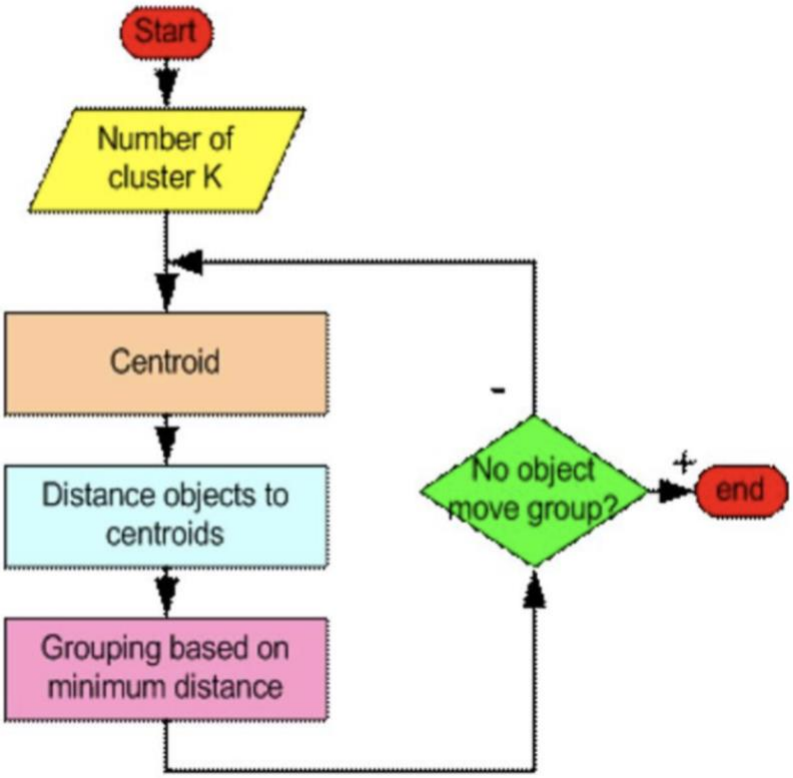
Mahalanobis distance

$$d(\vec{x}, \vec{y}) = \sqrt{(\vec{x} - \vec{y})^T S^{-1} (\vec{x} - \vec{y})}.$$

The algorithm I implemented for K means is shown below:
First, start to give the algorithm the number of clusters as wanted. Say for part (a), the cluster number should be 4. So tell the algorithm, I want 4 different groups. I randomly choose the centroids for each cluster which are the representative for that cluster or group. And then start to assign each image to a cluster by calculating the Euclidean distance of the image with K centroids. And then label each of the sample with the group number or cluster number.

The flowchart showing the algorithm I implemented for K-means is shown below:

Procedures

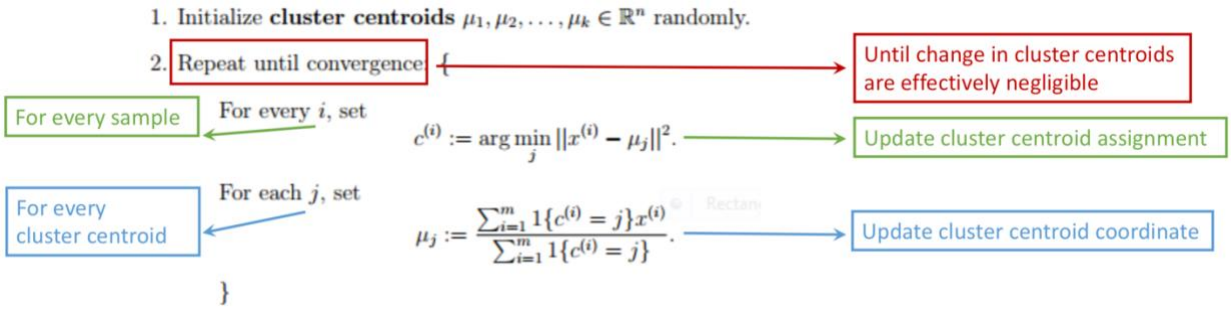


- Converge fast
- Depend on initialization very much
- Require cluster number

After all the images are assigned to corresponding or respective clusters, the centroids are recalculated by considering the images in the respective clusters. We use this procedure iteratively several times until it converges that means the centroid won't change too much. So during each iteration, center will move a little bit. So in the iteration what I do is that, for each sample, update it's label. After the assignment is updated, update the cluster centroid co-ordinate. Centroid is the representative of that cluster. Repeat it several times until the change is so small that it is already converged. Each time all the samples will be relabeled, that is they will be re-examined.

Procedures

In the clustering problem, we are given a training set $x^{(1)}, \ldots, x^{(m)}$, and want to group the data into a few cohesive "clusters." Here, we are given feature vectors for each data point $x^{(i)} \in \mathbb{R}^n$ as usual; but no labels $y^{(i)}$ (making this an unsupervised learning problem). Our goal is to predict k centroids **and** a label $c^{(i)}$ for each datapoint. The k-means clustering algorithm is as follows:

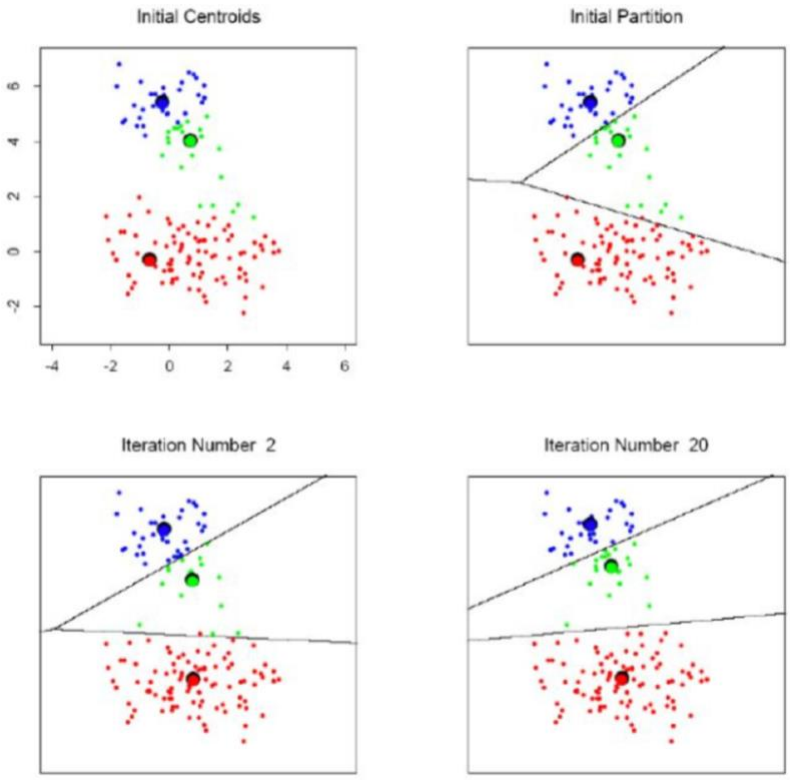


In the formula, there is this indicator function, that means in each cluster, the updated co-ordinate for the cluster is only based on the samples that are labelled by that cluster.

- Indicator function $1\{c^{(i)} = j\} = \begin{cases} 1 & c^{(i)} = j \\ 0 & c^{(i)} \neq j \end{cases}$

(Source: http://stanford.edu/~cpiech/cs221/handouts/kmeans.html)

Below in the image, there are initial centroids which are picked randomly. We give it blue, green and red dots. After the first iteration, it is shown below. And after several iterations, it becomes better as we see and compare in the image below. Every time the cluster centers will move.



(Source: Question PDF – HW #4)

Cluster them into four texture types with the following steps below to complete this problem.

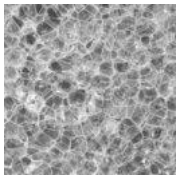
1. Feature Extraction: Use the twenty-five 5x5 Laws Filters to extract feature vectors from each pixel in the image (use appropriate boundary extensions).
2. Feature Averaging: Average the feature vectors of all image pixels, leading to a 25-D feature vector for each image.
3. Reduce the feature dimension from 25 to 3 using the principal component analysis (PCA). Plot the reduced 3-D feature vector in the feature space. (May use built-in C++/Matlab functions of PCA.)
4. Clustering: Use the K-means algorithm for image clustering based on the 25-D and 3-D obtained in Steps 2 and 3, respectively. Discussed the effectiveness of feature dimension reduction over K-means. Report the results and compared them with the reality (by eyes).

The texture classification algorithm is implemented as follows:

- Read the input image “texture1~12.raw” whose dimensions are height_size = 128, width_size = 128, BytesPerPixel = 1 using fread() function

- Subtract the mean of all pixel values or average of the input image from the original input image
- Perform the boundary extension by pixel replication
- Define the 5 1D masks for 5x5 Laws filter – L5, E5, S5, W5, R5
- Calculate the 5x5 masks of Laws filter by finding tensor products of the above 5 masks in different combinations
- Apply the 25 Laws filters on the input images
- Calculate the average energy features of the filtered image for the 25 filters for each texture and do feature normalization
- Each input image has 25 feature energy, repeat the above steps for all the 12 input images
- Reduce the 25D to 15D using the logic above (repeat each similar pair with average) and normalize using (x-mean)/standard deviation scaling formula
- Perform PCA to do the feature reduction from 25D to 3D
- Use k means clustering algorithm mentioned above for k = 4 on the 12 feature vectors obtained
- Label the input images based on the clustering into their respective class type
- Plot the graph showing the cluster formation and labels for each texture input image
- Display the table showing the input image texture number, showing the corresponding class label and class name it belongs to in the cluster

III. Experimental Results

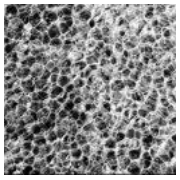


Input image – texture1.raw

Input image – texture2.raw

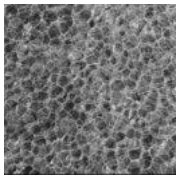
Input image – texture3.raw

Input image – texture4.raw



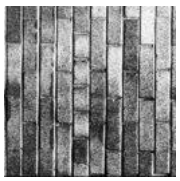
Input image – texture5.raw

Input image – texture6.raw



Input image – texture7.raw

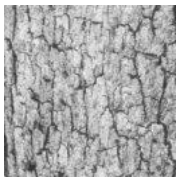
Input image – texture8.raw



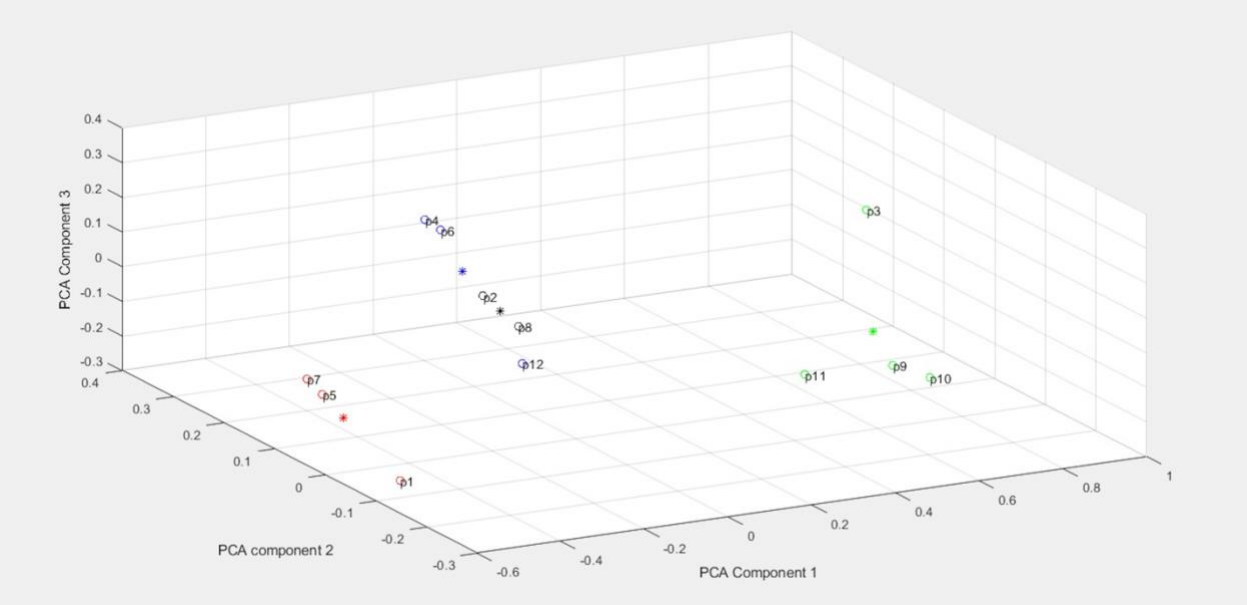
Input image – texture9.raw

Input image – texture10.raw

Input image – texture11.raw



Input image – texture12.raw



3D plot showing the centroids and the 4 clusters consisting of each corresponding texture image to the class label

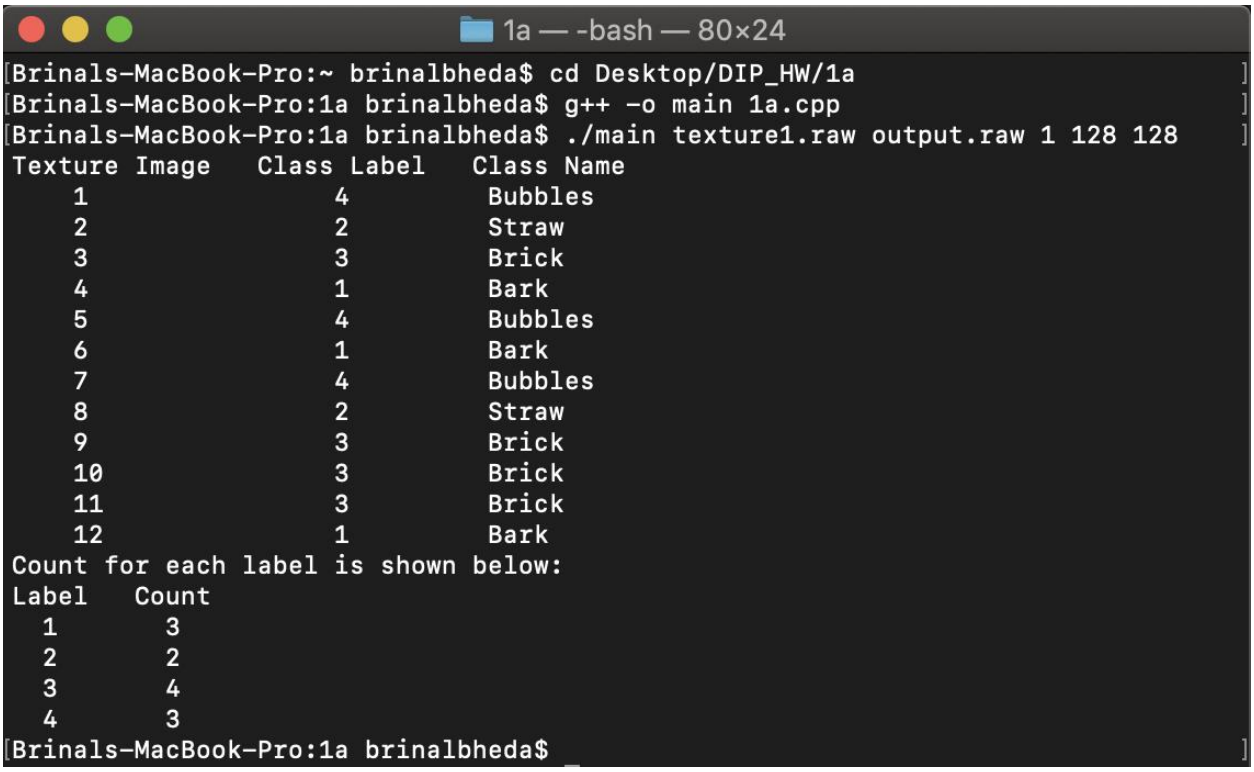


Table showing the texture classification according to the class label and texture name, also displaying the count for the respective cluster label

Cluster	Input image number
1	4, 6, 12
2	2, 8
3	3, 9, 10, 11
4	1, 5, 7

IV. Discussion

The twelve texture input images from texture1.raw to texture12.raw have been clustered into four texture types – cluster 1 - bark, cluster 2 – straw, cluster 3 – brick, cluster 4 – bubbles. This was done using 4 main steps – feature extraction using 5x5 Laws filter, feature averaging, PCA, K-means clustering algorithm.

I performed the boundary extension of the image by the pixel replication method.

On performing the feature averaging on the feature vector of all the image pixels, giving the 25D feature vector for each image. Discriminative power here means the ability that can help discriminate all the samples, i.e. cluster similar ones and

distinguish different classes. A feature which has good discriminative power is able to cluster the input images correctly and properly. There are 12 input images in this problem, and we have 25 features in total. The features which have large number of overlapping points have less discriminant power. The strongest discriminative power filter is the one which has more number of separate clusters formed which helps to differentiate between individual cluster class types. For the comparison of the discriminative power of different features, I plotted each feature on 1D plane and this is done for different class labels. I checked if the feature values overlap with the other cluster class type. Then I concluded that the features with many overlapping points do not give good accuracy for the texture image. Whereas, the features which have high discriminative power are the ones which help to classify the texture to their corresponding clusters perfectly and successfully. Thus, the feature dimension that has the strongest discriminative power is E5E5 or L5L5 and the one with the weakest is R5R5.

The feature dimension was reduced from 25 to 3 using the principal component analysis (PCA) and the reduced 3D feature vector plotted in the feature space is shown above. However, this did not cause a huge difference in the performance of clustering. It surely affects the speed, the k means clustering algorithm with 3D features converges faster than the k means clustering algorithm with 25D features, as there are lesser number of computations. The accuracy is improved.

The K-means clustering was done by choosing the value of k as 4 and we can see 4 texture clusters formed in the 3d plot above. Also, we see the table showing the classification results of texture images and corresponding class labels or cluster name it belongs to. Classification result may suffer from low sample number. 12 texture images may not be sufficient for a good classification and hence we see that texture 10 does not give proper classification.

The effectiveness of the feature dimension reduction over K-means is discussed here. The table for comparison of results obtained from K means clustering and from reality (by eyes) is compared and shown below. We can see the 12 texture input images visually and give the output determining in which cluster class do each one belongs by looking by our eyes.

Texture Input Image Number	Classification based on visual looking by eyes	Classification using the k means algorithm
1	Bubbles	Bubbles
2	Straw	Straw
3	Brick	Brick

4	Bark	Bark
5	Bubbles	Bubbles
6	Bark	Bark
7	Bubbles	Bubbles
8	Straw	Straw
9	Brick	Brick
10	Straw	Brick
11	Brick	Brick
12	Bark	Bark

We can see from the table above that the output for the classification based on visual looking by eyes and classification using the k means algorithm are all same except for the case of texture image 10 where the actual output should be straw however, the clustering labels it in the brick class as we can see from the 3D plot as well. It is classified to the wrong cluster class type. We can also see this from the output where the texture image number and corresponding class type and label is mentioned. We see that the count of labels under the brick shows 4 and under straw shows only 2. In the correct ideal case, each cluster class should have 3 texture images.

5 by 5 filters are larger than 3 by 3 Laws filters ones, making them able to differentiate more complex textures and hence, this leads to better results. However, for 1a, classification result may suffer from low sample number. Only 12 input texture images may not be sufficient for a good classification. Also, a single good feature is not enough to cluster the input images. A combination of different filters will help to gain a better result.

One more observation is that the texture feature extraction is a process which is sensitive to noise and this might also be one of the reasons of the inaccurate output. To improve the result, we should perform denoising on the texture image, make it noiseless and then check the clustering again. Here, we observe that the clustering got accurate results.

(b) Texture Segmentation

I. Abstract and Motivation

After applying the Laws filter to texture classification, we now use it for texture segmentation. Texture segmentation is difficult compared to texture classification. This is because there are various inside the same image which have to be segmented. This is mainly done to help in applications like object recognition, as it segments different objects in the image along the texture. In this process, we do not classify the individual images, we will be classifying the image pixels (different region pixels in the same input image). The main steps to perform in the texture segmentation operation are, feature extraction using the Laws filter and k means clustering algorithm, both of these are mentioned above and have been implemented in prob 1(a).

General Image Segmentation

Why Image Segmentation Is Difficult?

- What is a good number of segments?
  - What is the purpose of the segmentation?
    - Visual-saliency-based segmentation may be more meaningful
    - Object-recognition-based segmentation may be more relevant
- Human uses 3D information to segment (e.g. occlusion) while computers have only 2D image information
- Human uses semantics to group pixels while computers are very weak in semantic understanding

Basic Ideas

- Contour detection (contour serves as a separator)
  - Active contour
- Region growing
  - Watershed
- Graph-based
  - Pixels are nodes, their similarity is defined by an edge value
    - Very similar -> small edge value
    - Very different -> large edge value
    - How to define similarities? mostly related to color (could be others)

(Source: Discussion lecture dated 03/05/2019)

Signal-processing-based algorithms:

- The Laws Algorithm
- Use texture filters applied to the image to create filtered images from which texture features are computed

We use filters and generate some response and analyze the response. So, the filter used here is Laws filter. The whole procedure is shown in the diagram above. The overall process is we input the texture image and apply the filter to get the response and then we try to analyze all these responses which we call features and then we can get the results.

The Laws Algorithm

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]

5 different kernels for 5x5 Laws filters are used: Level – to detect level or brightness and average the pixel values, Edge – to detect edge, Spot – to detect spot, Wave – to detect wave, Ripple – to detect ripple. Based on these 5 basic kernels, generate 25 different filters. This is basically a vector product that is column vector times the row vector.

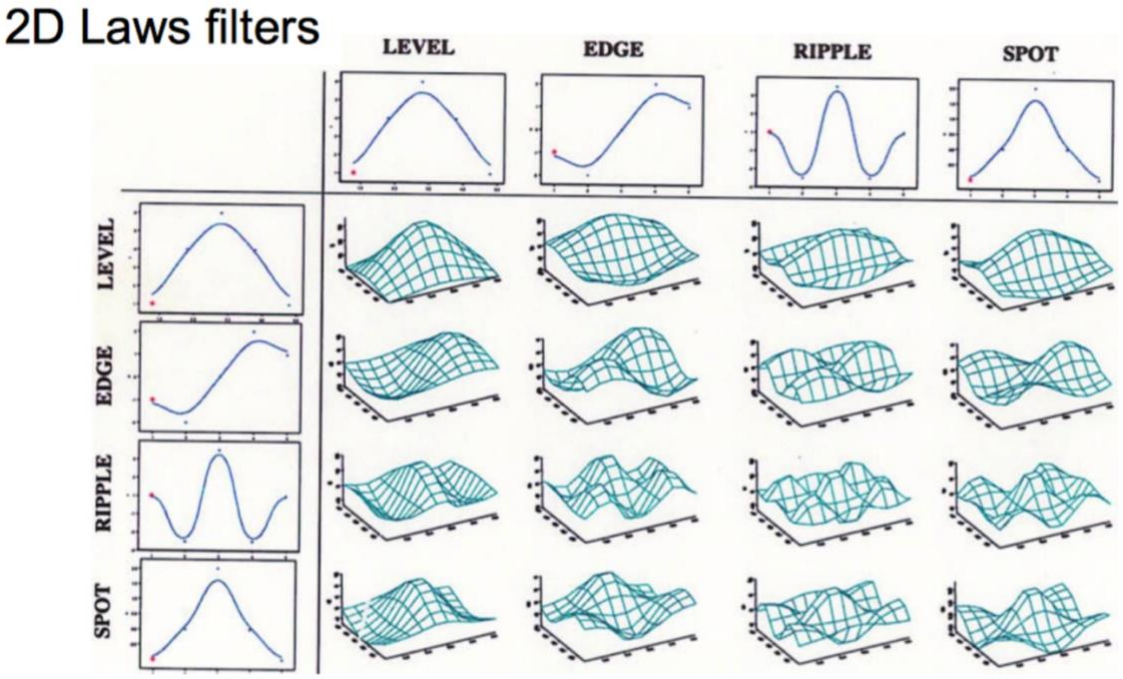
2D Laws filters

- An example: E5L5

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

Each filter is convolved with another to generate the 25 tensor products. This is the way how we create 25 different filters. Before performing this operation, we need to do boundary extension by replicating the pixels being used.

Below shown is how the filters look like. Say we have 4 kernels and 16 filters.



For level, along the first row, every filter is focused on the level performance. We can extract feature correspondingly by these different filters.

Since the input is one image having 7 different patches, so for each do the following steps:

Consider a window of an appropriate size and scan through the whole input image, doing the following operations on the center pixel. Subtract the mean of all pixels in each window of input image from the input image center pixel because there are some cases like may be both are similar regions but say one is darker and other is lighter so after we extract the feature, in the feature space these two vectors may disport away far from each other. We have to avoid such a case by reducing the illumination effects. Find the average value of all the pixel values in that window size in the input image and subtract the average found from the original input image center pixel values. Also, by doing this, it helps to remove the DC component in the image regions and along with it the redundant information from the image is also removed. This also prevents the feature vector from being dominated by high energy value.

There are 25 filters. Applying these filters, we get the response image and based on each response, calculate the average energy to form a 25-D vector for that certain texture region. This means we use one vector to represent each texture region. There are a lot of filters which are symmetric like L5E5/E5L5. They are quite similar so if we want to reduce the dimension, we sum up the two dimensions together to form a new one so that we can reduce the dimension from 25 to 15, however, this is not done in 1(b).

The calculation for the energy (feature) of the image is shown by the formula below:

$$\text{Energy} = \sum_{x=1}^{Row} \sum_{y=1}^{Col} (\text{abs}(f(x,y)))\text{^}2$$

Where, f(x,y) = Pixel Intensity,
Row = Number of Rows,
Col = Number of Columns

For that matrix, for each dimension we have to do feature normalization. Usually we subtract the mean and divide it by the standard deviation. Because may be sometimes one dimension is so dominant that it's value is like several hundred but the other dimension is very small like 0.9 or something. Hence, we need to reduce this imbalance. Thus, this part was the feature extraction.

Find the localized energy for each pixel for each of the 25 filtered outputs. To calculate the feature vectors, use the below formula to compute the energy. Thus, 25 energy vectors are formed.

$$\frac{1}{Window\ size * Window\ size} * [summation\ of\ all\ pixel\ values\ in\ each\ window\ for\ each\ filtered\ image]$$

Energy Feature Vector Normalization

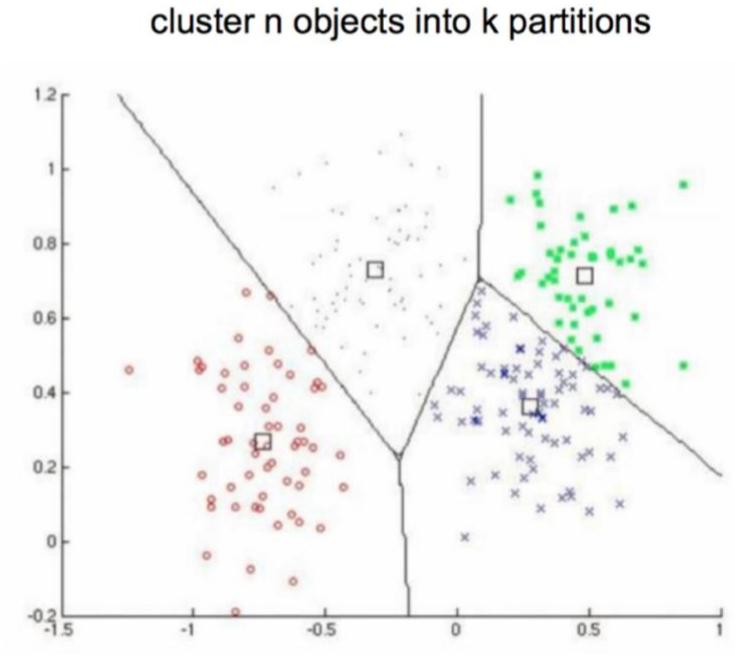
All kernels have a zero-mean except for $L5^T\ L5$. Actually, the feature extracted by the filter $L5^T\ L5$ is not a useful feature for texture classification and segmentation. Use its energy to normal all other features at each pixel.

1- Min-max scaling retains the original distribution of scores except for a scaling factor and transforms all the scores into a common range [0, 1]. It is useful when dealing with features with hard boundaries.

2- Standardization, which is calculated using the arithmetic mean and standard deviation of the given data, changes data distribution to zero-mean, unit-variance standard Gaussian. Standardization can be used for algorithms that assumes zero centric data.

K-means

A type of unsupervised data analysis algorithm



For the classification part, I used K-means. This is a type of unsupervised data analysis algorithm; means we do not need any information about labels. We do implement the procedure to get the feature and then analyze the final results. For example below, the space is the feature space i.e. the 25-D high dimensional space. All the samples are spread out in this space. We divide them or cluster them into several parts or groups based on their similarity. The similarity here is defined by the distance. Usually, we do the Euclidean distance defined below or use the Mahalanobis distance or can use some other distance weighted by Gaussian weights.

Distance measure

- determine how the similarity of two elements
- influence the shape of the clusters

Euclidean distance

$$\mathrm{d}(\mathbf{p},\mathbf{q}) = \mathrm{d}(\mathbf{q},\mathbf{p}) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + \cdots + (q_n - p_n)^2}$$

$$= \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2}.$$

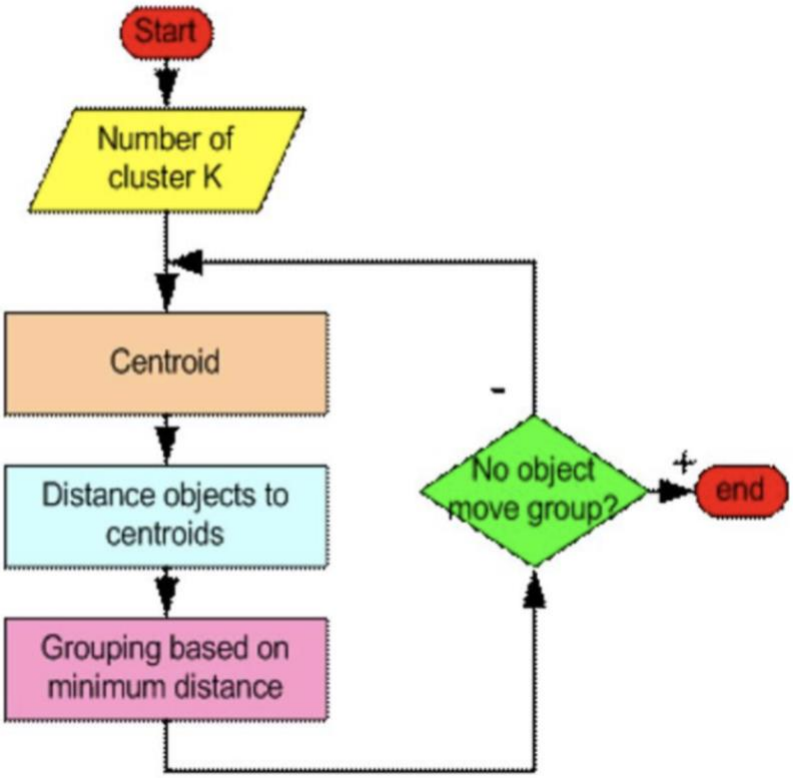
Mahalanobis distance

$$d(\vec{x}, \vec{y}) = \sqrt{(\vec{x} - \vec{y})^T S^{-1} (\vec{x} - \vec{y})}.$$

The algorithm I implemented for K means is shown below:
First, start to give the algorithm the number of clusters as wanted. Say for part (b), the cluster number should be 7. So tell the algorithm, I want 7 different groups. I randomly choose the centroids for each cluster which are the representative for that cluster or group. And then start to assign each image region to a cluster by calculating the Euclidean distance of the image region with K centroids. And then label each of the sample with the group number or cluster number.

The flowchart showing the algorithm I implemented for K-means is shown below:

Procedures

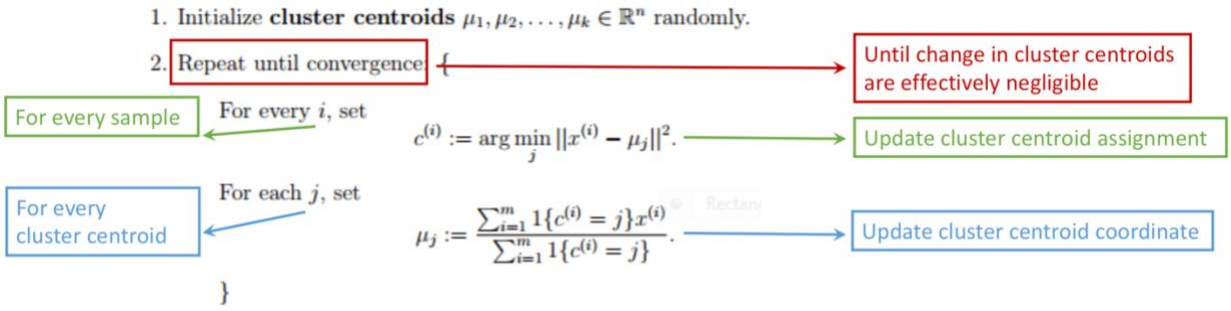


- Converge fast
- Depend on initialization very much
- Require cluster number

After all the image regions are assigned to corresponding or respective clusters, the centroids are recalculated by considering the image regions in the respective clusters. We use this procedure iteratively several times until it converges that means the centroid won't change too much. So during each iteration, center will move a little bit. So in the iteration what I do is that, for each sample, update it's label. After the assignment is updated, update the cluster centroid co-ordinate. Centroid is the representative of that cluster. Repeat it several times until the change is so small that it is already converged. Each time all the samples will be relabeled, that is they will be re-examined.

Procedures

In the clustering problem, we are given a training set $x^{(1)}, \ldots, x^{(m)}$, and want to group the data into a few cohesive "clusters." Here, we are given feature vectors for each data point $x^{(i)} \in \mathbb{R}^n$ as usual; but no labels $y^{(i)}$ (making this an unsupervised learning problem). Our goal is to predict k centroids **and** a label $c^{(i)}$ for each datapoint. The k-means clustering algorithm is as follows:



In the formula, there is this indicator function, that means in each cluster, the updated co-ordinate for the cluster is only based on the samples that are labelled by that cluster.

- Indicator function $1\{c^{(i)} = j\} = \begin{cases} 1 & c^{(i)} = j \\ 0 & c^{(i)} \neq j \end{cases}$

After the whole process of clustering is done, segmenting the image into textures. Now, each texture or label is given a gray value. We have 7 textures in the input image comb.raw and hence 7 gray levels (0, 42, 84, 126, 168, 210, 255) are used to recognize the seven segmented texture regions in the output image shown below.

II. Approach and Procedure

(Source: Discussion lecture dated 03/05/2019)

Since we have one large input image, we need to prepare small patches for classification. The classification is based on for those small patches. Say, window is 15x15, so for classification, label will be for that 15 by 15, for that pixel. So each pixel will be considered for that 15x15 patch and the result we get is only for that pixel. For that 15x15 patch, we perform feature extraction just by using the same procedure as in problem 1(a).

The result may look similar to the image shown below:

Texture Segmentation Example

Give different grayscales for different labels and show the results. Usually the random selected initialization points may not work well. So do some prior work to initialize the model. There is an algorithm called K-means++. The key idea is that the initialization is trying to find the samples which are most far apart from each other.

How to select initial cluster centers?

1. Randomly

2. Find initial sample points which are most far apart (K-means++)

How to select initial cluster centers? -> kmeans ++

1. Choose one center randomly (uniform distribution)
2. For remaining data point x, compute D(x) (the distance between x and the nearest center)
3. Choose one new data point at random as a new center, where a point x is chosen with probability proportional to D(x)2.
4. Repeat Steps 2 and 3 until k centers have been chosen.

The description for K-means algorithm is shown above. Start with one center centroid first, by choosing it uniformly and randomly. For the other ones, we generate the clusters one by one. For example, for the second one, calculate the

distance for the remaining data point to its nearest center and then choose the new centroid, based on the distance. So the distance is some kind of description for probability. Repeat it until select the number you want.

Creating windows is an approach to provide enough info for generating feature vector for each single pixel. Just like one sample is an image in 1a), in 1b) one sample is such kind of windows. In other words, texture class prediction is pixel-wise in 1b). This window approach is used for calculating the average energy for each pixel in a given filtered image. The total sample number is supposed to be the pixel number 510*510, as each window serves its center pixel for feature generation. That is, each window generates the 25-D feature vector for center pixel.

(Source: Question PDF – HW #4)

In this part, apply the twenty-five 5x5 Laws Filters to texture segmentation for the image shown below with the following steps.

1. Laws feature extraction: Apply all 25 Laws filters to the input image and get 25 gray–scale images.
2. Energy feature computation: Use a window approach to computer the energy measure for each pixel based on the results from Step 1. Try a couple of different window sizes. After this step, we obtain 25-D energy feature vector for each pixel.
3. Energy feature normalization: All kernels have a zero-mean except for $L5^T$ $L5$. Actually, the feature extracted by the filter $L5^T$ $L5$ is not a useful feature for texture classification and segmentation. Use its energy to normal all other features at each pixel.
4. Segmentation: Use the k–means algorithm to perform segmentation on the composite texture images based on the 25-D energy feature vectors.

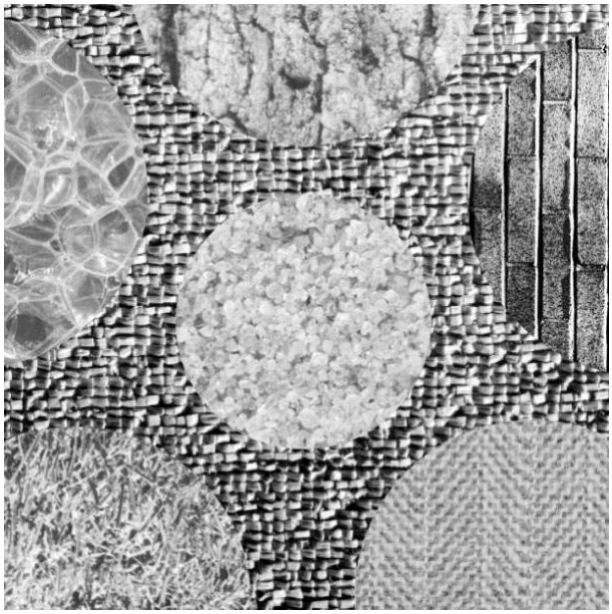


Figure showing composite texture image (comb.raw)

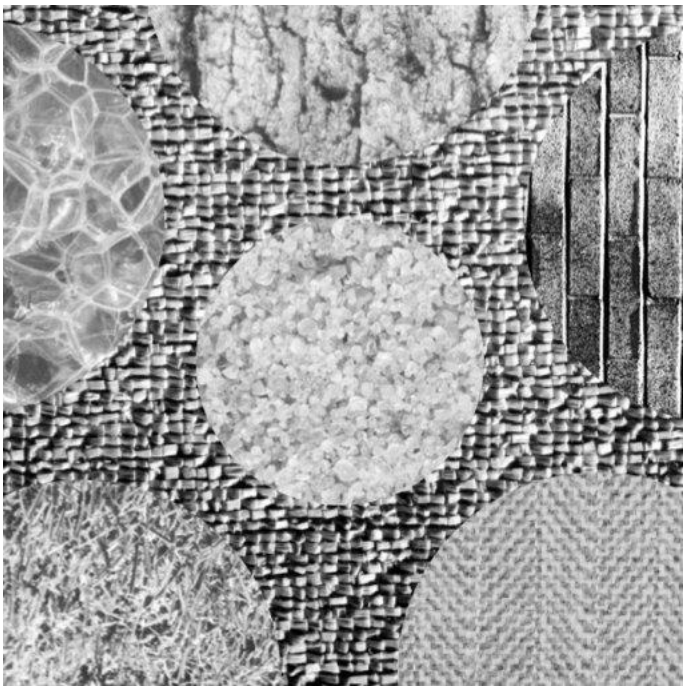
In order to denote segmented regions, if there are K textures in the image, the output image will be of K gray levels, with each level represents one type of texture. For example, there are 7 types of texture in the figure above so I can use 7 gray levels (0, 42, 84, 126, 168, 210, 255) to do denotation.

Algorithm implemented for texture segmentation:

- Read the input image "comb.raw" whose dimensions are height_size = 510, width_size = 510, BytesPerPixel = 1 using fread() function
- The window size variable is defined and the output is observed for different window sizes say 7, 15, 31
- Subtract the mean of all pixel values in the window or average of the pixel values in the window in the input image from the original input image center pixel
- Perform the boundary extension by pixel replication
- Define the 5 1D masks for 5x5 Laws filter – L5, E5, S5, W5, R5
- Calculate the 5x5 masks of Laws filter by finding tensor products of the above 5 masks in different combinations
- Apply the 25 Laws filters on the input images
- Calculate the average energy features of the filtered image for the 25 filters and do feature normalization
- The input image has 25 feature energy

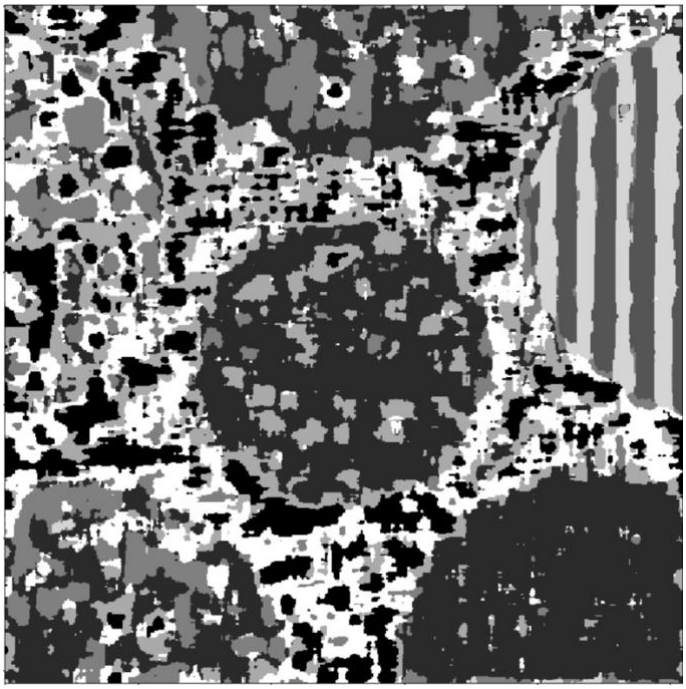
- Compute the localized energy for each pixel of the Laws filter output images
- Localize the energy of each pixel, this is done by dividing each pixel value by L5TL5 pixel energy value in the 25 energy arrays
- Initialize using the kmeans++ algorithm
- Use k means clustering algorithm mentioned above for k = 7 on the feature vectors
- Calculate the Euclidean distance of each pixel with 7 centroids and label that pixel
- Update the new centroid values by averaging the feature values in each cluster and iteratively repeat the steps until the centroid converges
- Assign the 7 labels these 7 gray levels (0, 42, 84, 126, 168, 210, 255) mentioned above, to each cluster after k means algorithm execution
- Display the output image for different window sizes

III. Experimental Results

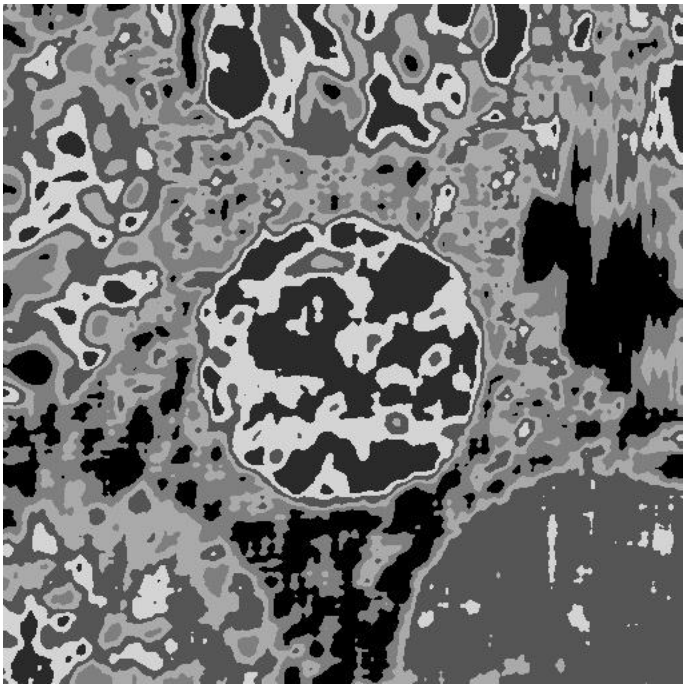


Input image – comb.raw

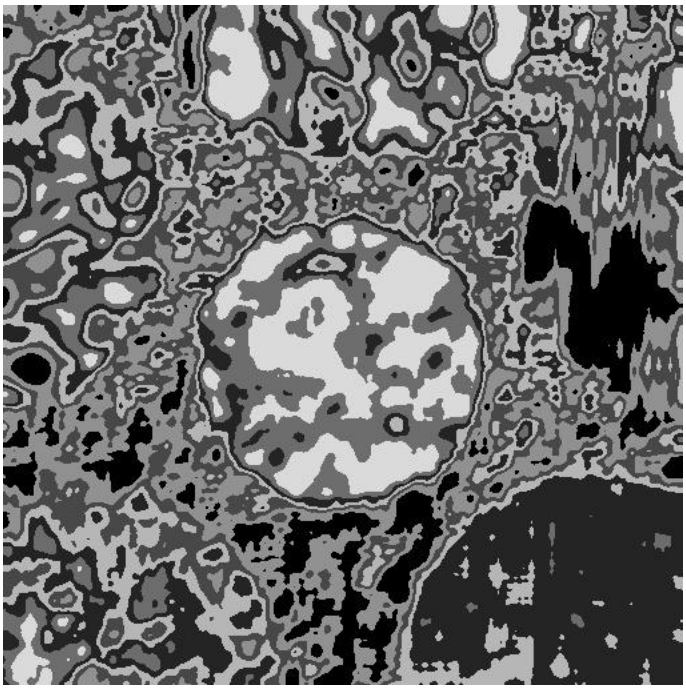
Output images showing texture segmentation for different window sizes, with normalization



Window size = 13

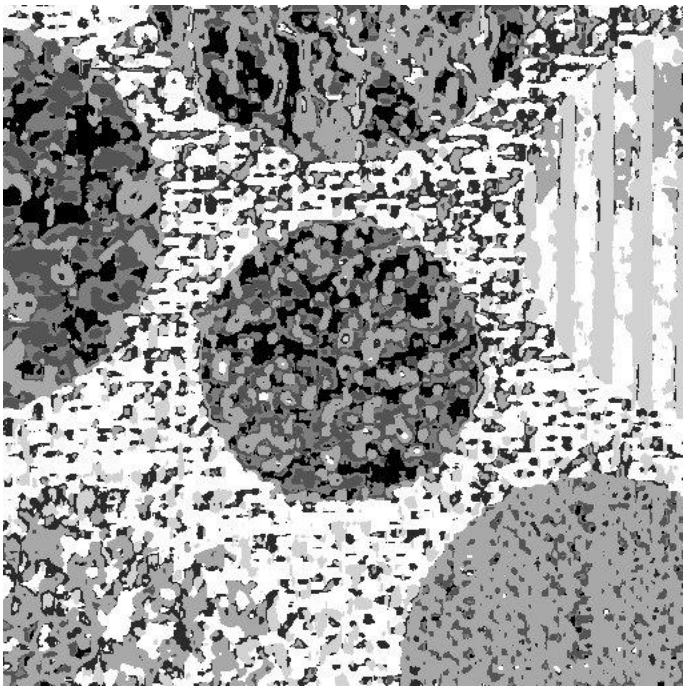


Window size = 17

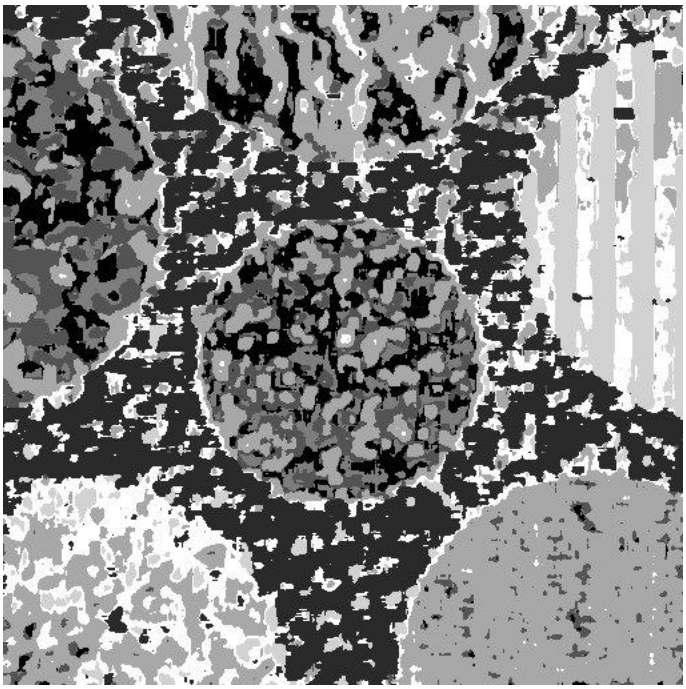


Window size = 19

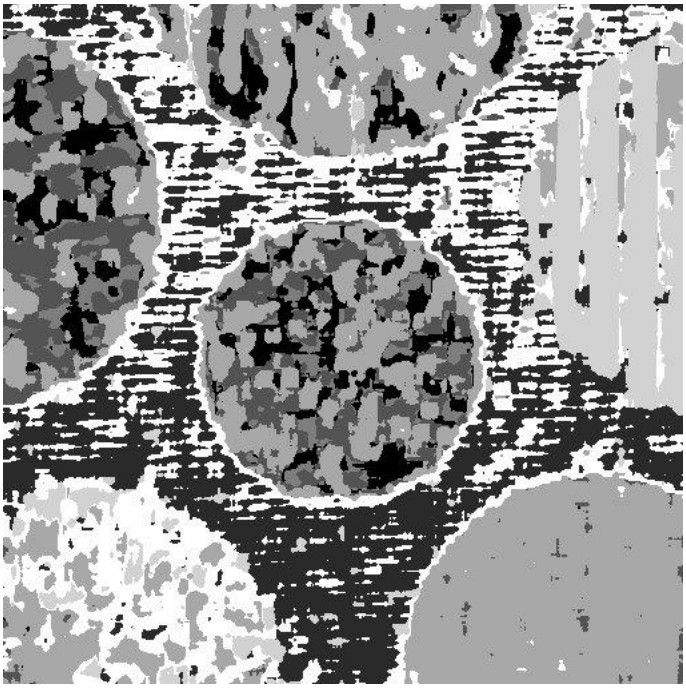
Output images showing texture segmentation for different window sizes, without normalization



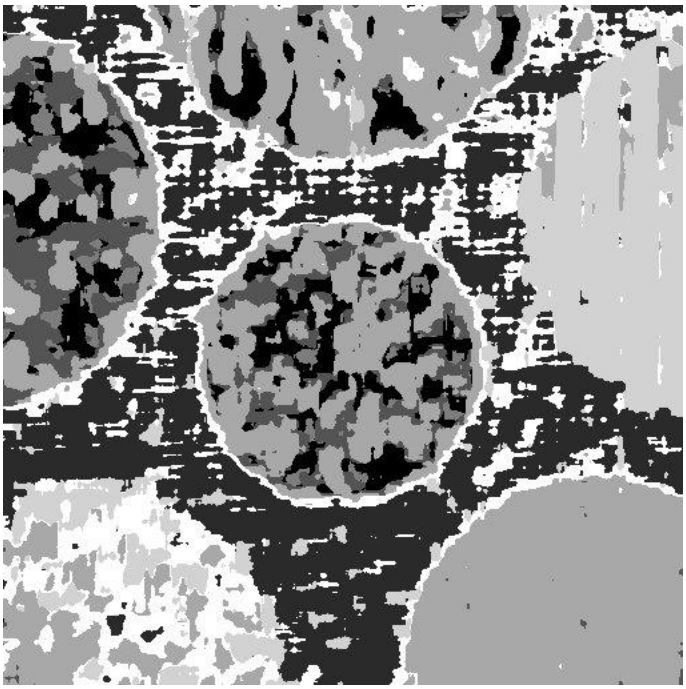
Window size = 7



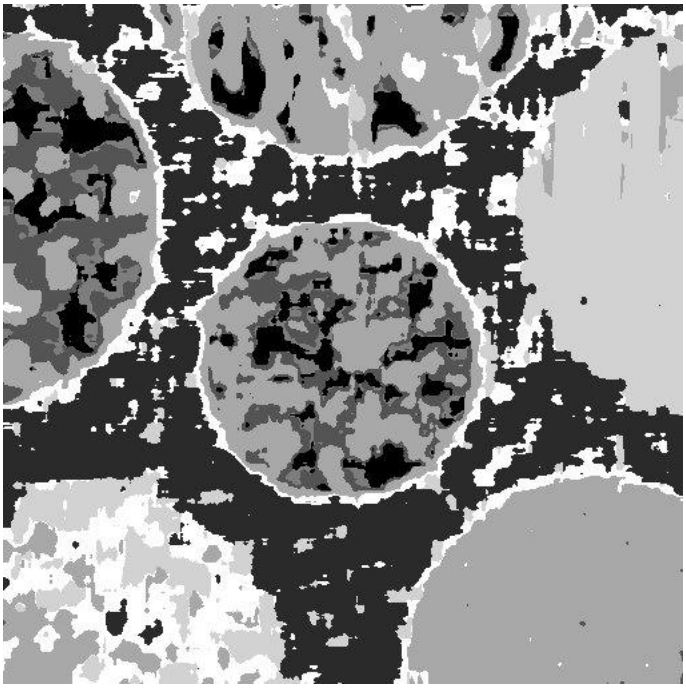
Window size = 9



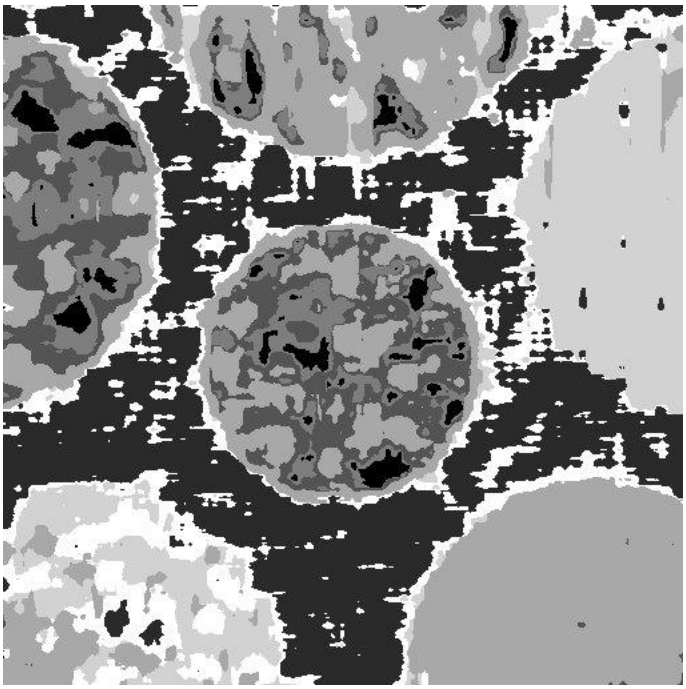
Window size = 13



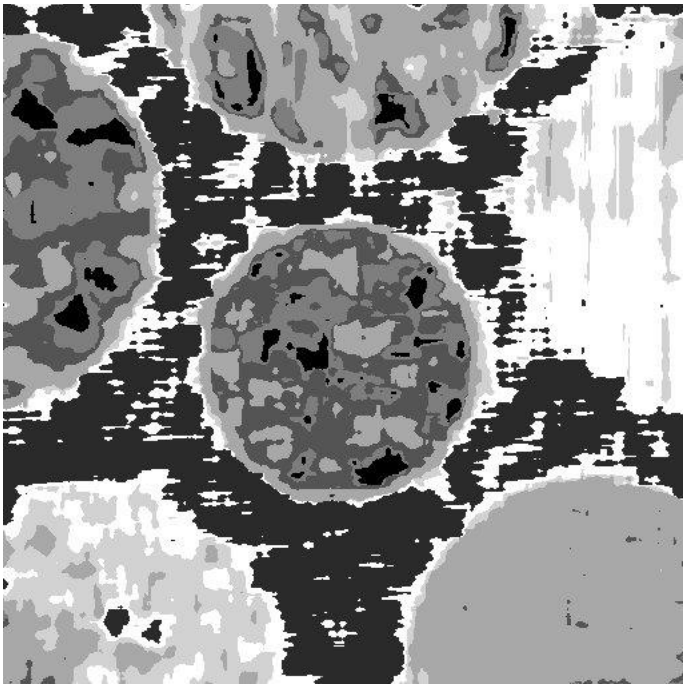
Window size = 15



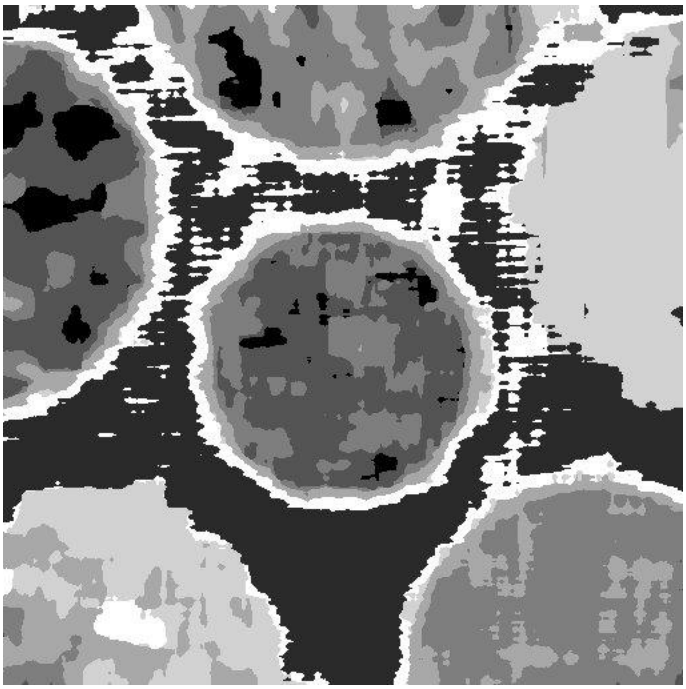
Window size = 17



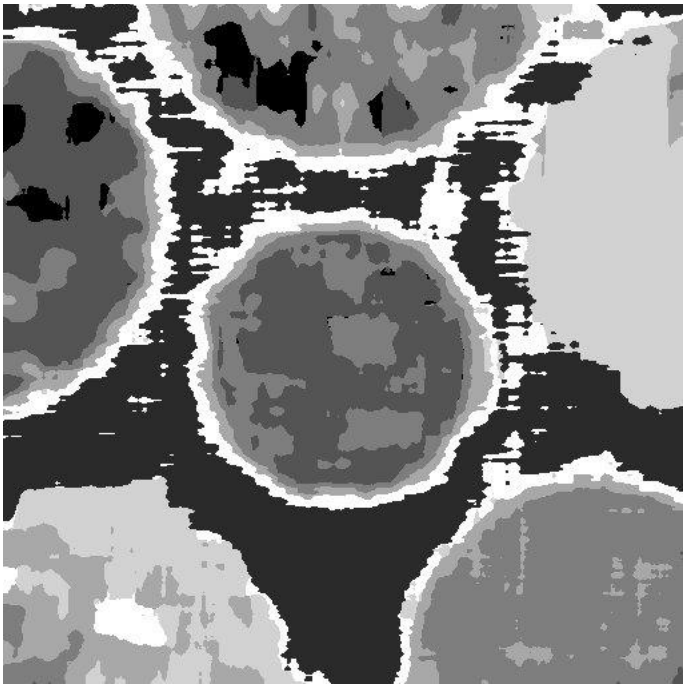
Window size = 19



Window size = 21



Window size = 30



Window size = 31

IV. Discussion

I performed the boundary extension of the image by the pixel replication method. 5 by 5 filters are larger than 3 by 3 Laws filters ones, making them able to differentiate more complex textures and hence, this leads to better results.

Image segmentation was done using the Laws filter. The feature normalization was performed using the L5TL5 filter as this filter has 0 mean and it is not used in texture classification for the features. However, normalization does not give quite good results. This may be because of the level brightness being affected and not being able to sort to the correct labels. They can still be improvised. The results show using with and without normalization. L5L5 was removed because it created outliers.

In the outputs displayed above for different window sizes, we can observe that there are 7 regions and hence the segmentation results look quite good by adopting the algorithm discussed above. We can see that the as the window size is increased from 7 to 31 gradually, the output results become better and the quality of segmentation is also improved. However, once a certain window size is reached, after changing furthermore we observe that the quality of segmentation does not vary much and it is not affected. We can see this observation in the last 2 cases where window sizes are 30 and 31, that the output does not show drastic segmentation changes.

The best quality for segmentation is obtained after the window size 15 to 21, however, even 31 gives quite improved segmented results, showing the 7 regions in forms of different gray levels helping to differentiate the boundary of the individual regions in the texture comb.raw image. The outputs with the lower window size have more distorted regions inside, and do not give a clean boundary of showing different texture regions segmented. However, the output image chosen for segmentation to determine how well the results show really depends on the application the image is being used for.

(c) Advanced Texture Segmentation Techniques

I. Abstract and Motivation

For enhancing the techniques used in 1(b) and getting better segmentation results, we adopt various techniques, like dimension reduction using PCA – principal component analysis and also develop post processing technique to merge small holes, especially inside the regions and obtain a clean segmented output showing 7 distinct regions.

Principal Component Analysis (PCA)
In the signal space, there are different kind of representation bases. But the projection of the bases is more random or uniform. But some places give better representation. Say if have smooth signal, do Fourier transform to eliminate high frequency. As we use cosine transform for image compression, to keep the low frequency band. The pixel with highest energy which is more important becomes the first principal component, and the second highest energy becomes the second principal component. Order the energies in different sub bands. We use this method to approximate certain texture, because energy distribution in the 9 bands is non-uniform. However, doing this in spatial domain is not possible. Thus, we look at Laws filter from two angles, one is look at the frequency domain decomposition and another domain is looking at the change of space representation.

(Source: https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html)

When performing texture analysis, we use Laws filter for feature extraction and all different kinds of features are extracted like low-level and high-level, important and less important. The crucial features containing a lot of information might be overcome at times by the features which are unimportant depending on the data image as well as the application we are using it for. Hence, it is necessary to reduce the dimension from high to low without the loss of information. PCA is used to reduce the higher dimensional data to the low dimensional data. This is done in such a manner that a higher variability in the data will be reproduced.

(Source: https://en.wikipedia.org/wiki/Principal_component_analysis)

In the PCA process, the Eigen values and Eigen vectors for the high dimensional data are changed to the low dimensional data by finding n largest Eigen values. And thus finding the corresponding Eigen vectors to them. This is done for dimensionality reduction and it uses singular value decomposition mentioned below.

Several methods are available for this, but we use the Principal Component Analysis (PCA) in Linear Algebra.

Two common methods

- Feature selection
- Feature reduction -> Principal component analysis (PCA)

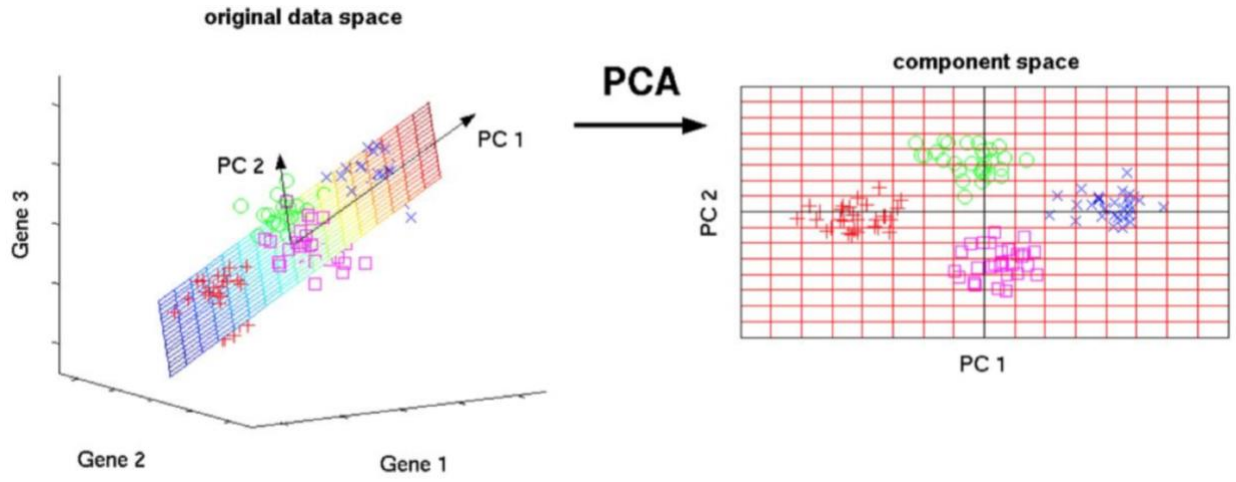
It is very important to do PCA on high dimensional data, by using PCA, better data representation is possible.

Retain significant eigenvalues (or singular values)

- ✓ Usually a high dimensional feature matrix has lot of redundancy
- ✓ As a result, its covariance matrix has many zeros (or close to zero) eigenvalues (or singular values)
- ✓ Such values can be discarded safely

Since orthogonal transform, resulting transformation gives uncorrelated features

- ✓ Why is uncorrelated features desirable?



I used build-in functions. The links are provided below:

You can use built-in PCA Object/function

OpenCV
https://docs.opencv.org/ref/2.4/d3/d8d/classcv_1_1PCA.html

Matlab
https://www.mathworks.com/help/stats/pca.html

The mathematical procedure for PCA using singular value decomposition is shown below:

Compute PCA via SVD

- Consider feature matrix X of size nxm
  - n is the number of features
  - m is the number of data points
  - Let Xi denotes the ith column of matrix X
- Compute Mean mX (vector of size n x 1)
  - $mX = (\Sigma i \in [1,m]\ Xi\ )/\ m$
- Compute zero mean feature matrix ZX of size nxm
  - $\forall\ i \in [1,m]$, $ZXi=Xi-mX$
  - ZX is also of size n x m

Compute PCA via SVD

- Compute SVD of ZX
  - $ZX = U \cdot S \cdot VT$
  - U is of size nxn, S is of size nxm and V is of size mxm
- Dimensionality Reduction
  - Equivalent to discarding small eigen/singular values
  - Assume S has singular values in descending order of magnitude
  - Then, if we want to reduce dimensionality from n to n'
  - We just retain first n' singular values. This corresponds to retaining first n' columns of U, S

Compute PCA via SVD

- Compute reduced transformation matrix UR
  - $\forall\ i \in [1,n'] \ni n' \leq n$, URi =Ui
  - UR is of size nxn'
- Finally, transformation step; let RX be dimensionality reduced matrix, then
  - $RX = URT \cdot ZX$
  - RX is of size n'xm
- Use RX instead of X

(Source: Discussion lecture dated 03/05/2019)

Part (b) may not show some very good and clear segmentation results. So in part (c), we try different things to get better results. Thus, we can use PCA and develop a post processing technique to merge small holes like in large region, they have same label but there are some small holes with different labels. For boundary cases, the boundary of two other joint regions with different labels can be focused on. Do not care about other textures at that time, just focus on those two textures.

II. Approach and Procedure

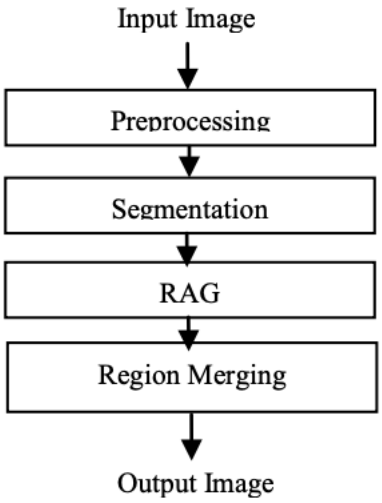
(Source: Question PDF – HW #4)

We do not get good texture segmentation results for the complicated image in the figure in part(b). Develop various techniques to enhance the segmentation result. Several ideas are sketched below.
1. Adopt the PCA for feature reduction and, thus, cleaning.
2. Develop a post-processing technique to merge small holes.
3. Enhance the boundary of two adjacent regions by focusing on the texture properties in these two regions only.

(Source: http://iieng.org/images/proceedings_pdf/7795E0314085.pdf)

This process is used to merge small holes and enhance the boundary of the two adjacent regions.

The overall procedure used for the advanced technique is shown below in the flowchart.
RAG stands for region adjacency graph.

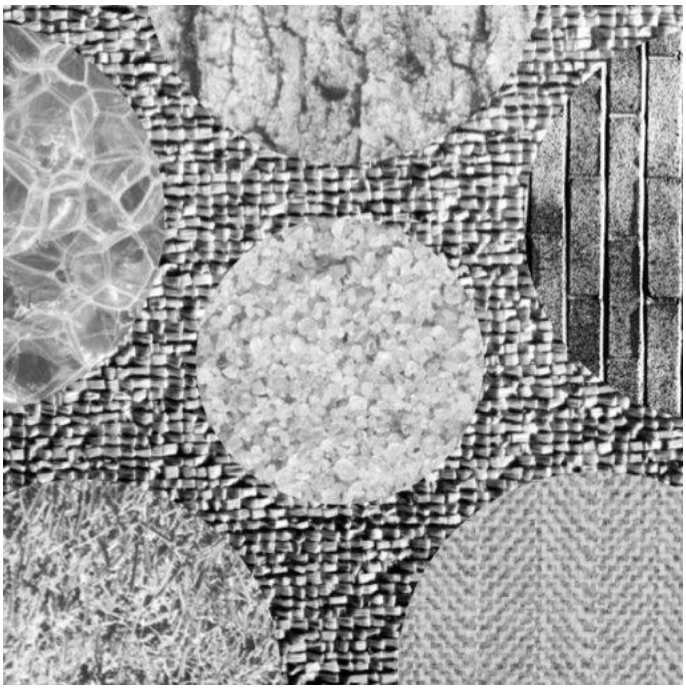


In this paper, region merging algorithm for image segmentation is successfully enhanced. Using opening-closing reconstruction and finding minimal minima, the oversegmentation problem is overcome. Region Adjacency Graph is constructed and the edge weight of adjacent region pair is found. Consistency of the adjacent regions with minimum edge weight is calculated by randomly choosing half-size of the pixels of each region pair with respect to Nearest Neighboring Region. We can implement the above and get better segmentation results.

Algorithm implemented:

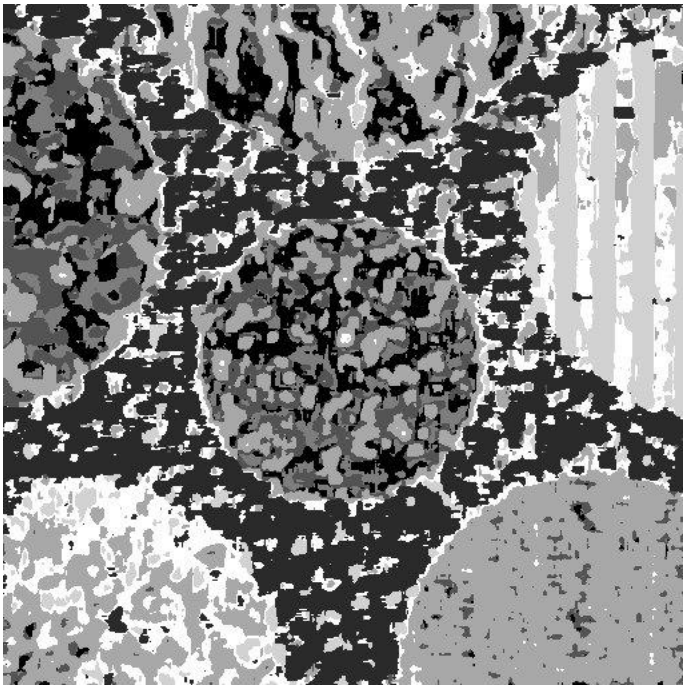
- Read the input image “comb.raw” whose dimensions are height_size = 510, width_size = 510, BytesPerPixel = 1 using fread() function
- The window size variable is defined and the output is observed for different window sizes say 7, 15, 31
- Subtract the mean of all pixel values in the window or average of the pixel values in the window in the input image from the original input image center pixel
- Perform the boundary extension by pixel replication
- Define the 5 1D masks for 5x5 Laws filter – L5, E5, S5, W5, R5
- Calculate the 5x5 masks of Laws filter by finding tensor products of the above 5 masks in different combinations
- Apply the 25 Laws filters on the input images
- Calculate the average energy features of the filtered image for the 25 filters and do feature normalization
- The input image has 25 feature energy
- Compute the localized energy for each pixel of the Laws filter output images
- Localize the energy of each pixel, this is done by dividing each pixel value by L5TL5 pixel energy value in the 25 energy arrays
- Reduce the 25D to 15D using the logic above (repeat each similar pair with average) and normalize using (x-mean)/standard deviation scaling formula
- Perform PCA to do the feature reduction
- Initialize using the kmeans++ algorithm
- Use k means clustering algorithm mentioned above for k = 7 on the feature vectors
- Calculate the Euclidean distance of each pixel with 7 centroids and label that pixel
- Update the new centroid values by averaging the feature values in each cluster and iteratively repeat the steps until the centroid converges
- Assign the 7 labels these 7 gray levels (0, 42, 84, 126, 168, 210, 255) mentioned above, to each cluster after k means algorithm execution
- Display the output image for different window sizes

III. Experimental Results

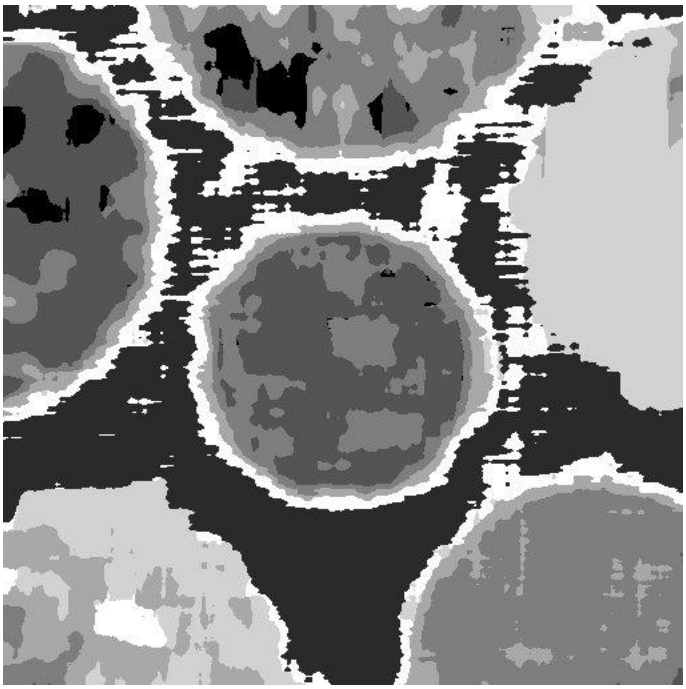


Input image – comb.raw

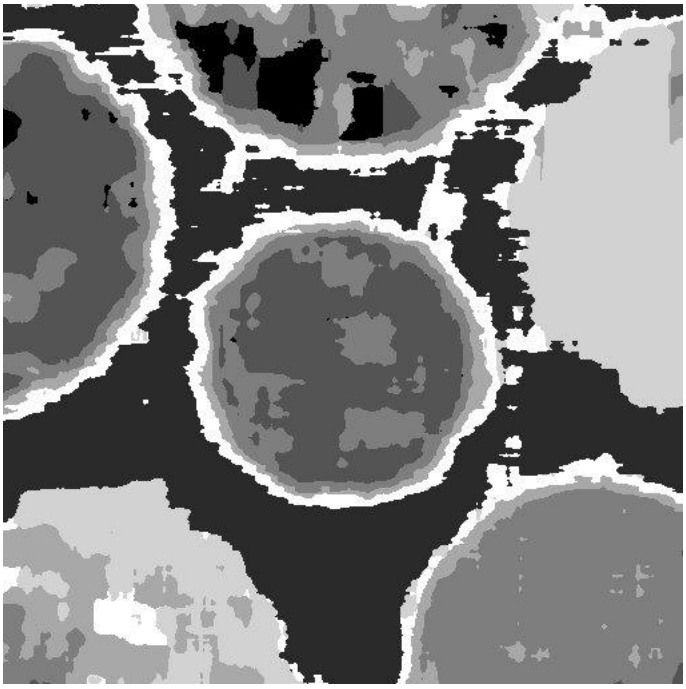
Output images showing texture segmentation for different window sizes after applying advanced texture segmentation techniques



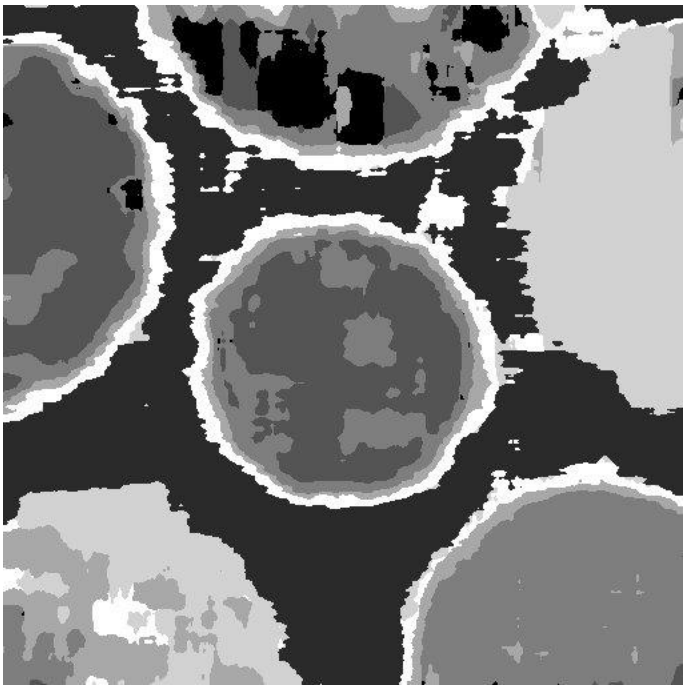
Window size = 11



Window size = 17



Window size = 19



Window size = 21

IV. Discussion

I used advanced texture segmentation techniques to develop cleaner and better outputs.

I performed the boundary extension of the image by the pixel replication method. 5 by 5 filters are larger than 3 by 3 Laws filters ones, making them able to differentiate more complex textures and hence, this leads to better results.

In the outputs displayed above for different window sizes, we can observe that there are 7 regions and hence the segmentation results look quite good by adopting the algorithm discussed above. We can see that the as the window size is increased from 17 to 21 gradually, the output results become better and the quality of segmentation is also improved as compared to the outputs obtained in the 1(b) case. However, once a certain window size is reached, after changing furthermore we observe that the quality of segmentation does not vary much and it is not affected.

The best quality for segmentation is obtained after the window size 15 to 21, however, even 21 gives quite improved segmented results, showing the 7 regions in forms of different gray levels helping to differentiate the boundary of the individual regions in the texture comb.raw image. The outputs with the lower

window size have more distorted regions inside, and do not give a clean boundary of showing different texture regions segmented. However, the output image chosen for segmentation to determine how well the results show really depends on the application the image is being used for. But after applying the advanced techniques, we do observe that the boundaries of different texture regions are well defined and also less distortions appear inside those regions. Few patterns inside the region are also being visible.

I carried out PCA even in the 1(a) problem and found good results. Dimension reduction using PCA is not extremely powerful, however the outputs obtained are quite decent. The feature dimension was reduced from 25D to 3D using the principal component analysis (PCA). However, this did not cause a huge difference in the performance of clustering. It surely affects the speed, the k means clustering algorithm with lower dimensional features converges faster than the k means clustering algorithm with high dimensional features, as there are lesser number of computations. The accuracy is improved.

In PCA, we select the number of principal components, multiply it with the input so as to obtain the low dimensional data and later get the low dimensional input energy feature extracted data. This is performed before clustering as it will help to improve results as it reduces the dimension of data. PCA basically helps as it contains only the data with high variance and hence gives clear and cleaner cluster formation. The principal components are highly uncorrelated variables.

For method 2 and 3, a post-processing technique to merge small holes and enhance the boundary of two adjacent regions by focusing on the texture properties in these two regions only, I implemented the method of region adjacency graph (RAG) and region merging by referring the paper mentioned above. This method helps to get better segmentation which leads to better understanding of the region and also able to differentiate the region boundary for analyzing the image in an accurate way.

Also, other advanced techniques which can be performed are denoising, Gabor filters, run length matrix to obtain a clean better output for texture segmentation.

Problem 2: Image Feature Extractor

(a) SIFT

I. Abstract and Motivation

In images we consider mostly boundary and surface, texture as discussed above. Another important point we consider in the image is key points/ interesting points. Key point is like the corner point which has large curvature.

(Source: https://en.wikipedia.org/wiki/Scale-invariant_feature_transform)

The **scale-invariant feature transform** (**SIFT**) is a feature detection algorithm in computer vision to detect and describe local features in images. It was patented in Canada by the University of British Columbia[1] and published by David Lowe in 1999.[2] Applications include object recognition, robotic mapping and navigation, image stitching, 3D modeling, gesture recognition, video tracking, individual identification of wildlife and match moving.

SIFT keypoints of objects are first extracted from a set of reference images[2] and stored in a database. An object is recognized in a new image by individually comparing each feature from the new image to this database and finding candidate matching features based on Euclidean distance of their feature vectors. From the full set of matches, subsets of keypoints that agree on the object and its location, scale, and orientation in the new image are identified to filter out good matches. The determination of consistent clusters is performed rapidly by using an efficient hash table implementation of the generalised Hough transform. Each cluster of 3 or more features that agree on an object and its pose is then subject to further detailed model verification and subsequently outliers are discarded. Finally the probability that a particular set of features indicates the presence of an object is computed, given the accuracy of fit and number of probable false matches. Object matches that pass all these tests can be identified as correct with high confidence.[3]

Main stages
- Scale-invariant feature detection
- Feature matching and indexing
- Cluster identification by Hough transform voting
- Model verification by linear least squares
- Outlier detection

Algorithm
- Scale-space extrema detection
- Keypoint localization
  - Interpolation of nearby data for accurate position
  - Discarding low-contrast keypoints
  - Eliminating edge responses
- Orientation assignment
- Keypoint descriptor

Features

The detection and description of local image features can help in object recognition. The SIFT features are local and based on the appearance of the object at particular interest points, and are invariant to image scale and rotation. They are also robust to changes in illumination, noise, and minor changes in viewpoint. In addition to these properties, they are highly distinctive, relatively easy to extract and allow for correct object identification with low probability of mismatch. They are relatively easy to match against a (large) database of local features but, however, the high dimensionality can be an issue, and generally probabilistic algorithms such as k-d trees with best bin first search are used. Object description by set of SIFT features is also robust to partial occlusion; as few as 3 SIFT features from an object are enough to compute its location and pose. Recognition can be performed in close-to-real time, at least for small databases and on modern computer hardware.

Applications

- Object recognition using SIFT features
- Robot localization and mapping
- Panorama stitching
- 3D scene modeling, recognition and tracking
- 3D SIFT-like descriptors for human action recognition
- Analyzing the Human Brain in 3D Magnetic Resonance Images

(Source: Question PDF – HW #4)

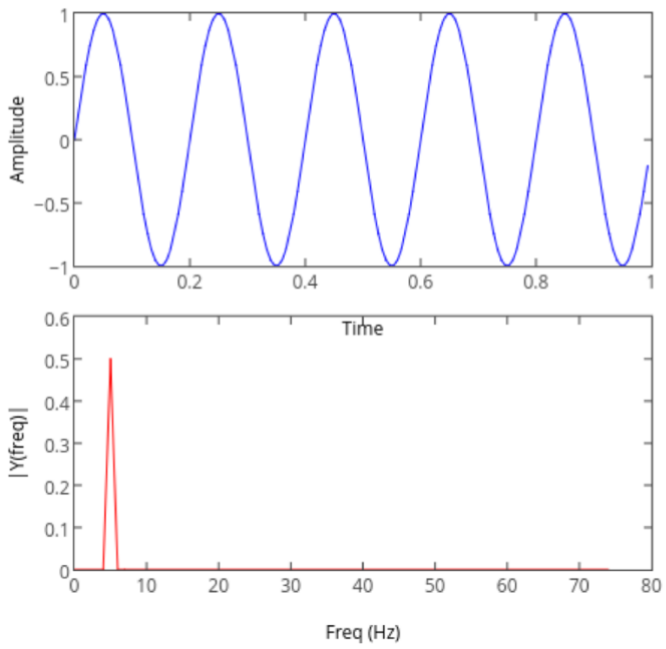
Image feature extractors are useful for representing the image information in a low dimensional form.

(Source: Discussion lecture dated 03/05/2019)

SIFT stands for Scale Invariant Feature Transform. SIFT is basically image feature extractor. Why do we need feature extractor? It is because for example, of the Fourier transform in signal processing, we use the Fourier transform to characterize what is happening for the input signal in the frequency domain from the time domain. So the transform really helps us to find it's semantic property of the input signal so we can understand better what is happening there. And similar things if we use the image feature extractor to characterize what is happening inside the image. It is picturing that say we have a cat image, it digitizes the image. There is a whole bunch of numbers. We are dealing with high resolution images and dealing with millions of pixels. Basically, there are millions of high resolution, high dynamic range of numbers. And that is impossible to process. Even for a single image, the computation is too heavy. Hence, the solution is we reduce the dimension. Reduce the ocean of numbers into smaller number of numbers. And that's where we need to use the image feature extractor.

Image Feature Extractor

- Why feature extractor?



The graph above shows a concept diagram of what image, feature extraction and advantage, benefit can we obtain.

(Source: Lowe, David G. "Object recognition from local scale-invariant features." iccv. Ieee, 1999)

SIFT was very famous and widely used before the neural networks came up. It was a very popular traditional method for image feature extraction. SIFT is an image describing feature vector. Input to the system is 2-D image. And the output is 1-D fixed size feature vector. It has 3 properties, scale, rotation and translation invariant, which means if we do these geometric modifications to the input image, the SIFT will characterize them and it will be robust to any geometrical manipulation to the input image.

- An image describing feature vector
- Scale-, rotation- and translation-invariant
- Scale-invariant through multi-scale filtering

The way to achieve all these each individual geometrical modification for example for the scale-invariant property, first is through the multi-scale filter. Although the name indicates it kind of like a transformation, but when you break starting from stage to stage, it is a very complicated system and it is patented. Hence, we use the open source code to implement this problem.

The internal procedure of SIFT is explained: First we feed the input image to SIFT, it would decompose it into multi-scaled images. So first do the difference of Gaussian on input images. The difference of Gaussian is that we apply Gaussian kernels into each individual dimension of input image. And it would take the difference between the two, which ends up into the difference image. And that difference image would serve as the first level pyramid, and it would apply the same algorithm again and again, until it builds the multi-scale image. So, we input a single image in and when it comes out from the difference of Gaussian kernels, it becomes pyramid of images. The purpose of this is that we want different scales representing the original images. So higher the level of pyramid, it represents lower resolution of the input image. The lower one represents the higher resolution. So when we zoom in, it images into different scales. Each level of pyramid would represent different scales of the input image. By doing this, it achieves the scale-invariant, which means if we shrink the image i.e. doing the scaling operation on input image using SIFT and the SIFT is going to be able to characterize similar contents.

- Rotation- and translation-invariant through locating key-points as extrema of difference-of-Gaussian function
- Each key-point is assigned a canonical orientation

The rotation and translation invariant property are achieved by two things: the first is difference of Gaussian (DoG) functions. SIFT is achieving by DoG that is to locate the extrema of the resulting pyramid of images, so the extrema which is the local, predefined moving window will find the maximum or the minimum in that local moving window. And it will find these extrema points level by level and if we find this consistently, the maximum or the minimum of that local patch, we define that as the key point. When we use the SIFT software, if we want to plot or visualize what it characterizes as images, it will generate a whole bunch of key points scattered all around the image. And those key points are the local extrema of the moving window. These key points are used to characterize some of the invariant property of the image, hence they are invariant to the rotation and translation operations. Each key point is described by an orientation information. So this orientation is calculated

by looking around it's 8 neighborhoods. And these 8 neighborhoods, each one will generate a directional information. So basically just take the difference between the neighbors along each direction. The resulting numbers are being recorded and used as the orientation information. By doing this, the vector generated is invariant to both rotation and translation.

- Illumination robustness achieved by thresholding the gradient magnitudes at a value of 0.1 times the maximum possible gradient value
- Canonical orientation is determined by the peak in a histogram of local image gradient orientation

The SIFT is also designed in addition to the robustness to the geometrical modifications to the illumination change.

- Advantage of DoG against LoG: speed

The Difference of Gaussian is different than the Laplacian of Gaussian. We used LoG for the edge detection problems in HW #2. That is high pass filtering of the low frequency signals. But the DoG is quite similar to LoG, both are doing the same thing i.e. first smooth the input, then take the difference between these 2 smooth signals. But the advantage of the DoG over LoG is that DoG is implemented in SIFT, the advantage is it's speed, it is much faster, because we only take the simple subtraction operation, so it is faster than LoG.

- Vector size: 8x4x4 + 8x2x2 = 160

In the open source code, the output dimension is generally 108, which is different than the vector size in the original paper, which is 160. It consists of 2 resolutional vectors, first vector is 8 neighborhoods of 4x4 little patch and 8 neighborhoods of 2x2 little patch. So, this 4x4 and 2x2 are 2 different resolutions. They also indicate the orientation information of each local key point. So each key point is described by either the 4x4 or 2x2 neighborhood. Once we get these 160 numbers, these are all floating numbers. We can use that to characterize on what is happening inside the image.

For Python – use OpenCV Sift function.
For Matlab – use Vlfeat function.

(Source: http://yann.lecun.com/exdb/mnist/)

II. Approach and Procedure

(Source: Question PDF – HW #4)

In this problem, read the original SIFT paper in link mentioned below and answer the following questions.

I. From the paper abstract, the SIFT is robust to what geometric modifications?
II. How does SIFT achieves its robustness to each of them?
III. How does SIFT enhances its robustness to illumination change?
IV. What are the advantages that SIFT uses difference of Gaussians (DoG) instead of Laplacian of Gaussians (LoG)?
V. What is the SIFT's output vector size in its original paper?

(Source: Lowe, David G. "Object recognition from local scale-invariant features." iccv. Ieee, 1999)

(Source: https://www.cs.ubc.ca/~lowe/papers/iccv99.pdf)

III. Discussion

1) From the paper abstract, the SIFT is robust to the following geometric modifications – scale-invariant, rotation-invariant and translation-invariant. The features are invariant to image scaling, translation, and rotation, and partially invariant to illumination changes and affine or 3D projection.
It has 3 properties, scale, rotation and translation invariant, which means if we do these geometric modifications to the input image, the SIFT will characterize them and it will be robust to any geometrical manipulation to the input image.

2) The way to achieve all these each individual geometrical modification for example for the scale-invariant property, first is through the multi-scale filter. The first stage identifies key locations in scale space by looking for locations that are maxima or minima of a difference-of-Gaussian function. Firstly, we feed the input image to SIFT, it would decompose it into multi-scaled images. So first do the difference of Gaussian on input images. The difference of Gaussian is that we apply Gaussian kernels into each individual dimension of input image. And it would take the difference between the two, which ends up into the difference image. And that difference image would serve as the first

level pyramid, and it would apply the same algorithm again and again, until it builds the multi-scale image. So, we input a single image in and when it comes out from the difference of Gaussian kernels, it becomes pyramid of images. The purpose of this is that we want different scales representing the original images. So higher the level of pyramid, it represents lower resolution of the input image. The lower one represents the higher resolution. So when we zoom in, it images into different scales. Each level of pyramid would represent different scales of the input image. By doing this, it achieves the scale-invariant, which means if we shrink the image i.e. doing the scaling operation on input image using SIFT and the SIFT is going to be able to characterize similar contents.

The rotation and translation invariant property is achieved through locating key points as extrema of difference-of-Gaussian function. Each key point is assigned a canonical orientation. In order to make this as stable as possible against lighting or contrast changes, the orientation is determined by the peak in a histogram of local image gradient orientation. The rotation and translation invariant property are achieved by two things: the first is difference of Gaussian (DoG) functions. SIFT is achieving by DoG that is to locate the extrema of the resulting pyramid of images, so the extrema which is the local, predefined moving window will find the maximum or the minimum in that local moving window. And it will find these extrema points level by level and if we find this consistently, the maximum or the minimum of that local patch, we define that as the key point. When we use the SIFT software, if we want to plot or visualize what it characterizes as images, it will generate a whole bunch of key points scattered all around the image. And those key points are the local extrema of the moving window. These key points are used to characterize some of the invariant property of the image, hence they are invariant to the rotation and translation operations. Each key point is described by an orientation information. So this orientation is calculated by looking around it's 8 neighborhoods. And these 8 neighborhoods, each one will generate a directional information. So basically just take the difference between the neighbors along each direction. The resulting numbers are being recorded and used as the orientation information. By doing this, the vector generated is invariant to both rotation and translation.

Scale-invariant edge groupings that make local figure-ground discriminations would be particularly useful at object boundaries where background clutter can interfere with other features. The indexing and verification framework allow for all types of scale and rotation invariant features to be incorporated

into a single model representation. Maximum robustness would be achieved by detecting many different feature types and relying on the indexing and clustering to select those that are most useful in a particular image.

3) The illumination robustness is achieved by thresholding the gradient magnitudes at a value of 0.1 times the maximum possible gradient value. This reduces the effect of a change in illumination direction for a surface with 3D relief, as an illumination change may result in large changes to gradient magnitude but is likely to have less influence on gradient orientation.
   Canonical orientation is determined by the peak in a histogram of local image gradient orientation.

4) The main advantage that SIFT uses difference of Gaussians (DoG) instead of Laplacian of Gaussians (LoG) is speed. The Difference of Gaussian is different than the Laplacian of Gaussian. We used LoG for the edge detection problems in HW #2. That is high pass filtering of the low frequency signals. But the DoG is quite similar to LoG, both are doing the same thing i.e. first smooth the input, then take the difference between these 2 smooth signals. But the advantage of the DoG over LoG is that DoG is implemented in SIFT, the advantage is it's speed, it is much faster, because we only take the simple subtraction operation, so it is faster than LoG. Also, another disadvantage of LoG is that it is costly.

   To achieve rotation invariance and a high level of efficiency, we have chosen to select key locations at maxima and minima of a difference of Gaussian function applied in scale space. This can be computed very efficiently by building an image pyramid with resampling between each level. Furthermore, it locates key points at regions and scales of high variation, making these locations particularly stable for characterizing the image.

5) The SIFT's output vector size in its original paper is 8x4x4 + 8x2x2 = 160. In the open source code, the output dimension is generally 108, which is different than the vector size in the original paper, which is 160. It consists of 2 resolutional vectors, first vector is 8 neighborhoods of 4x4 little patch and 8 neighborhoods of 2x2 little patch. So, this 4x4 and 2x2 are 2 different resolutions. They also indicate the orientation information of each local key point. So each key point is described by either the 4x4 or 2x2 neighborhood. Once we get these 160 numbers, these are all floating numbers. We can use that to characterize on what is happening inside the image. This means that approximately the same image region will be examined at both scales, so that

any nearby occlusions will not affect one scale more than the other. Therefore, the total number of samples in the SIFT key vector, from both scales, is 8x4x4 + 8x2x2 or 160 elements, giving enough measurements for high specificity.

(b) Image Matching

I. Abstract and Motivation

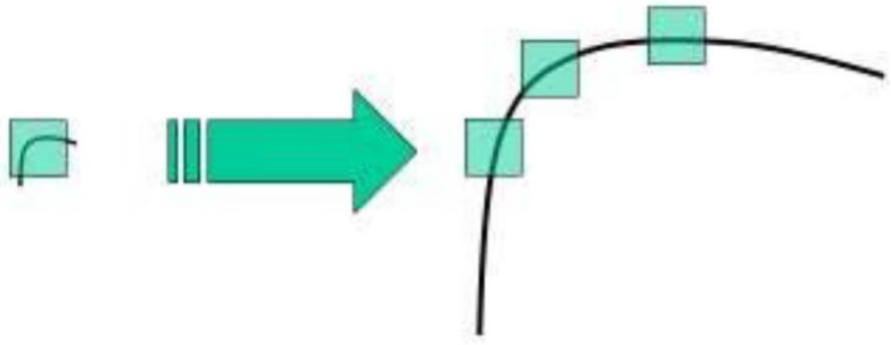
Find the key points of one image and locate the key points in another image even if it's size and rotation are different. Do the key point matching. Find the key point pair such that it is rotation invariant and scale invariant. Thus, we can find the matching between the two images and build the correspondence between the similar key point. This is important for image matching. Key point is very valuable to build the correspondence for matching between the two images.

There are two issues with the key point: 1. Detection, 2. Description (Embedding). Try to embed the neighborhood (pixels) of a key point into a high dimensional vector. This is fundamental in modern machine learning. Further can find distance, clustering, correlation in high dimensional space.

(Source: http://www.cs.ucf.edu/courses/cap4453/sift/)

SIFT Detection

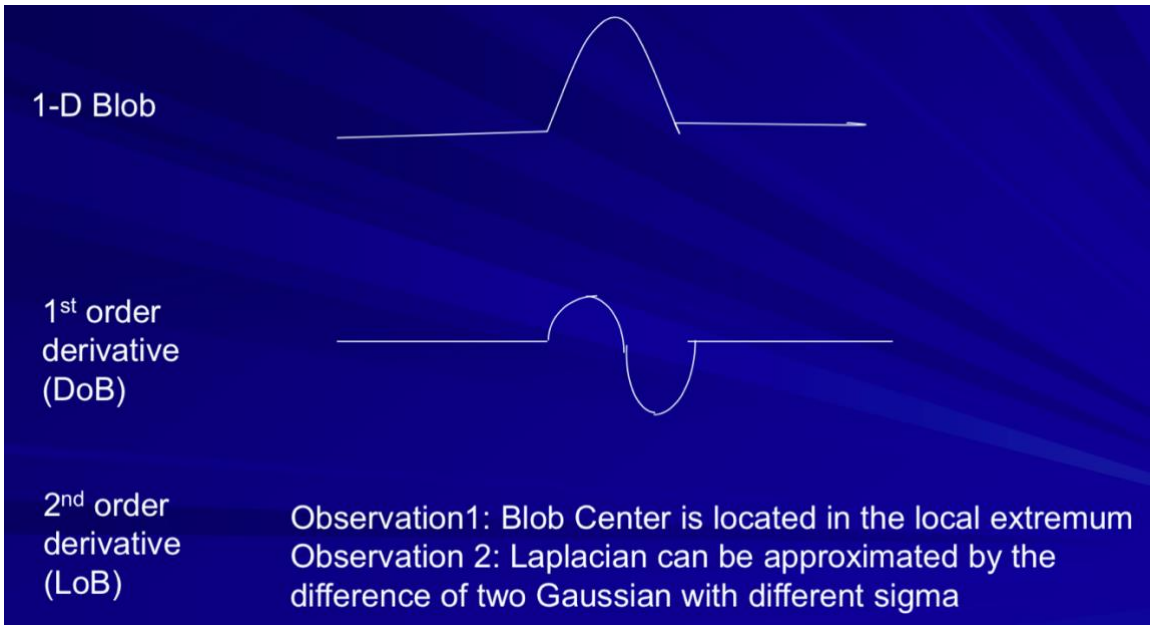
Edge/corner detector – separator between two regions
Blob detector – separator of a thin line in a background region
Corner detector – separator of a salient point in a line, example: Harris corner detector



Harris corner detector is rotation invariant but not scale invariant.

- Proposed by David Lowe in ICCV 1999
- Generates image features, "key points"
    - Invariant to image scaling and rotation
    - Partially invariant to change in illumination and 3D camera viewpoint
    - Many can be extracted from typical images
    - Highly distinctive

1D blob/spot manipulation

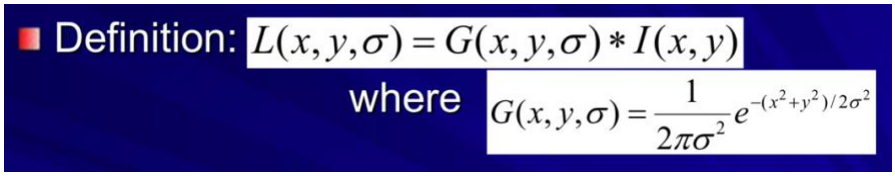


Laplacian is the approximate difference of 2 Gaussian kernels.

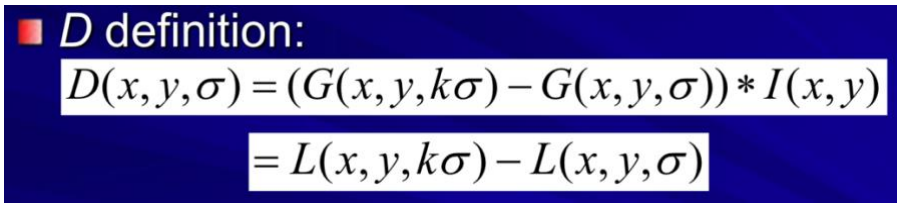
Algorithm Overview:

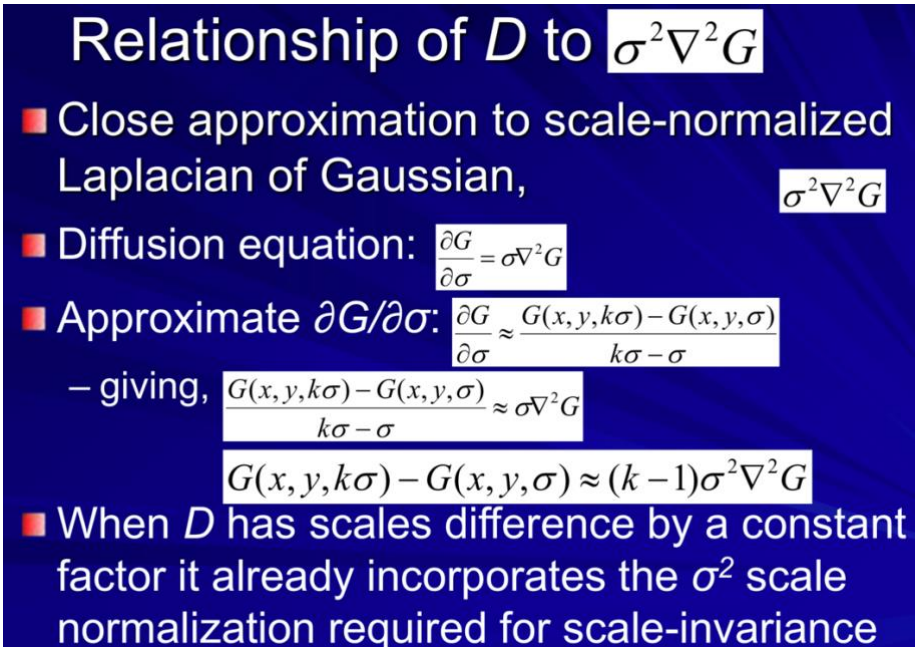
- Scale-space extrema detection (rough location) – related to detection
    - Uses difference of Gaussian function to approximate the Laplacian operator
- Key point localization (exact or fine tune location) – related to detection
    - Sub-pixel location and scale to fit to a model
- Orientation assignment – related to representation
    - 1 or more for each key point
- Key point descriptor (embedding) – related to representation
    - Created from local image gradients

Scale space

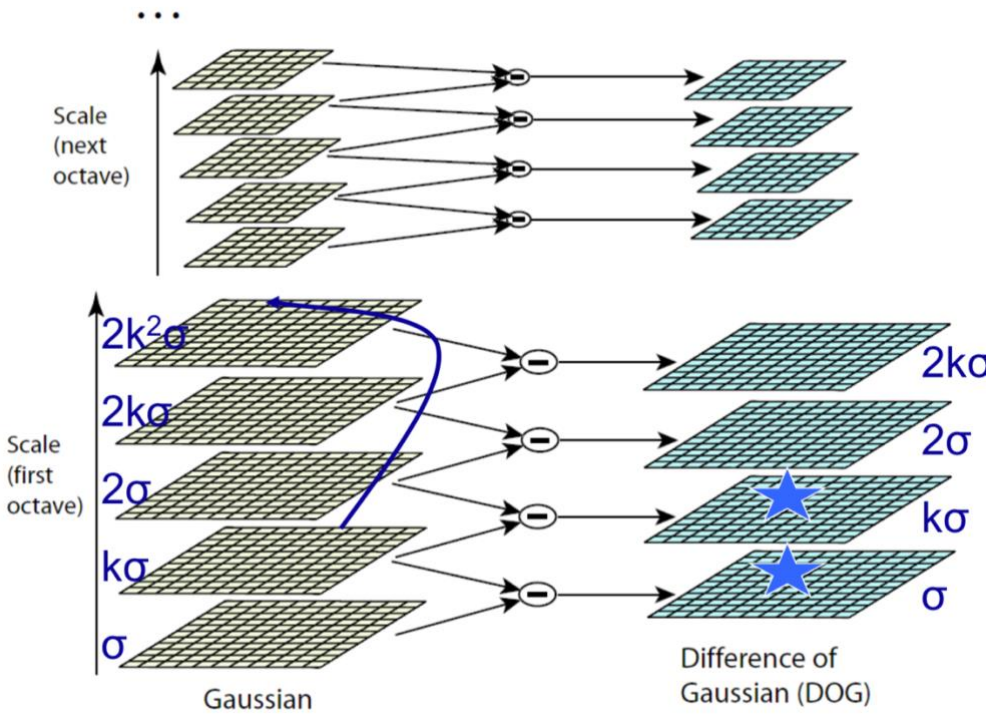


Implement as a filter bank with multiple sigma in parallel. Very large sigma causes blur and smaller sigma produces a clean image.
Key points are detected using scale space extrema in difference of Gaussian function D. Efficient to compute.



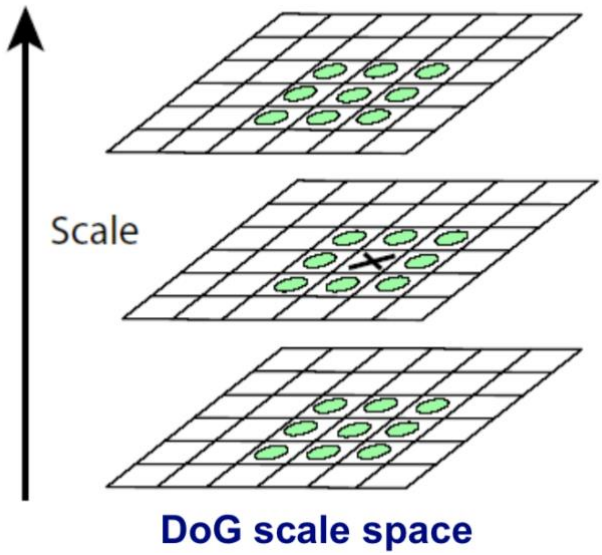


Scale space construction
Octave below mean different sampling.



Typically, k = $\sqrt{2}$ = 1.414, $\sigma = 1.6$, no. of scale = 5 and no. of octaves = 4.

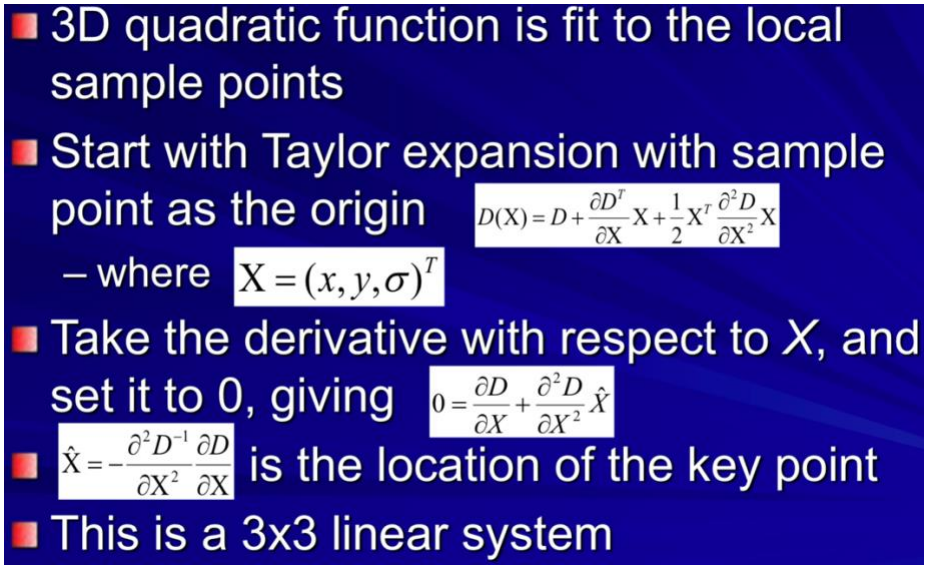
Key point selection
Downsample the original images and go up to next octave and downsample again. In first octave, we have 5 sigma and do convolution to get the result. Then find the difference. Find the extrema i.e. local minimum and maximum.



A key point is selected only if it is a local minimum or maximum in the 3D DoG scale space.

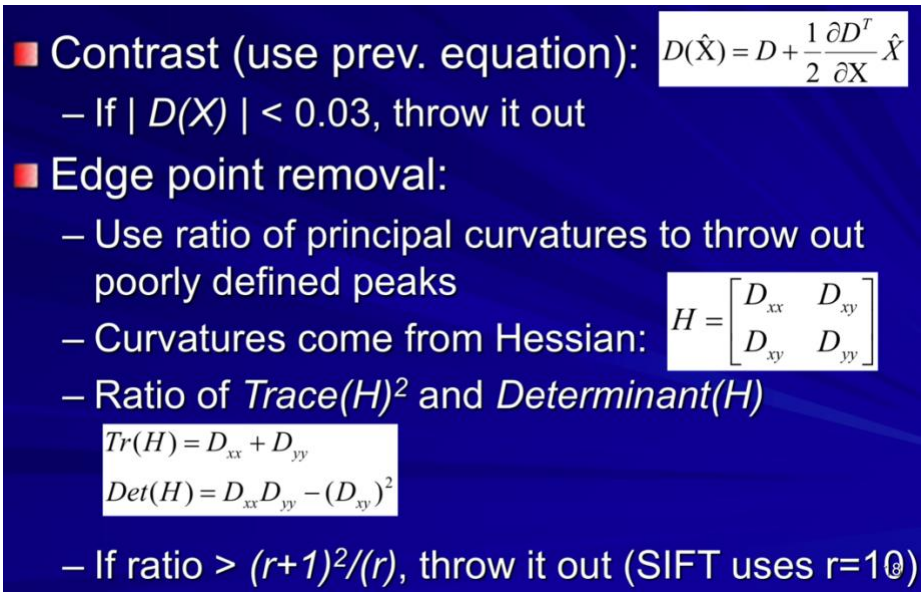
Key point localization

If we have the first octave, it has only the original image resolution. But if we go to the second, third, … octave then the local minimum and maximum are in the coarse image resolution. Use the Laplacian space – extrema points. Go to the fine image resolution.



Post processing

1. Remove points in the planar regions (low contrast i.e. low pixel variation regions).
2. Remove points in the non-salient edge regions. This is related to the curvature that defining the corner point as the key point. Remove only the corner points.



Key point orientation assignment

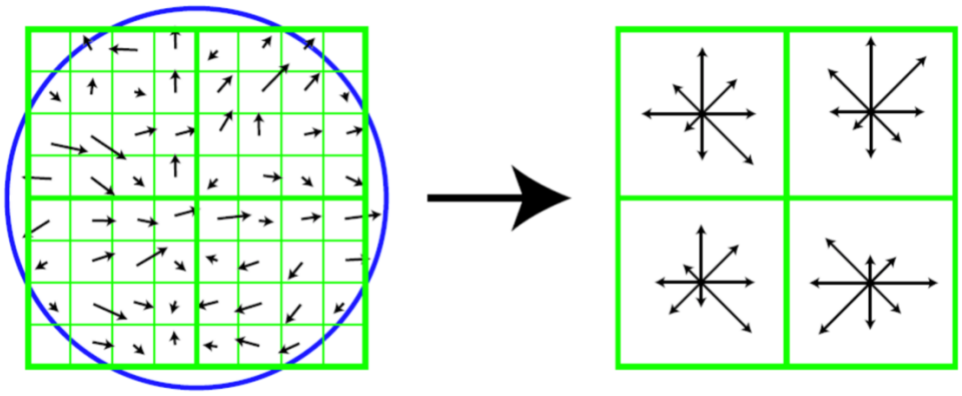
- An orientation is assigned to each keypoint to achieve invariance to image rotation.
- A neighborhood is taken around the keypoint location depending on the scale and the gradient magnitude and direction is calculated in that region.
- An orientation histogram with 36 bins covering 360 degrees is created. It is weighted by gradient magnitude and Gaussian-weighted circular window with its sigma equal to 1.5 times the scale of the keypoint.
- The highest peak in the histogram is taken and any peak above 80% if it is also considered to calculate the orientation.
    - It creates key points with the same location and scale but different directions.
- It contributes to stability of matching.

Say there a keypoint of interest and it is defined as (X, Y, σ). Then we consider Gaussian weighted neighborhood where sigma prime = 1.5 (sigma). For surrounding points calculate gradient magnitude and angle (every 10 degree). Calculate the histogram, where magnitude is greater than 80%. Then look at the angle histogram. Each key point points to something. Looking at the angle, we find the orientation.

SIFT Descriptor

- A 16x16 neighborhood around the key point.
    - Divided into 16 sub-blocks of size 4x4.

- For each sub-block, 8 bin orientation histograms (16 samples) are created.
    - A total of 128 bin values are available.
    - It is represented as a vector to form key point descriptor.
    - Besides, several measures are taken to achieve robustness against illumination changes, rotation, etc.



- Descriptor has 3 dimensions (x, y, θ).
- Position and orientation of each gradient sample rotated relative to keypoint orientation.
- Best results achieved with 4x4x8 = 128 descriptor size.
- Normalize to unit length.
    - Reduces effect of illumination change.
- Cap each element to 0.2, normalize again for fine tuning.
    - Reduces non-linear illumination changes.
    - 0.2 determined experimentally.

Object detection

- Create a database of keypoints from training images
- Match keypoints to a database
    - Nearest neighbor search

PCA-SIFT

- Different descriptor (same keypoints).
- Apply PCA to the gradient patch.
- Descriptor size is 20 (instead of 128).
- More robust, faster.

Summary of the SIFT -

- Scale space
- Difference-of-Gaussian
- Localization
- Filtering
- Orientation assignment
- Descriptor, 128 elements

One application of SIFT is Image matching.

(Source: Question PDF – HW #4)

One implementation of SIFT is image retrieval. I do nearest neighbor search in the searching database for the query image which is represented as a SIFT extracted feature vector.

Figure 2.1 River images

II. Approach and Procedure

(Source: Question PDF – HW #4)

Used open source SIFT tool (OpenCV, VLFeat, etc.) to find the key-points in the two river images shown above. Pick the key-point with the largest scale in river image 1 and find its closest neighboring key-point in river image 2. Discussed my results, esp. the orientation of each key- points.

(Source: http://www.vlfeat.org/overview/sift.html)

(Source:https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_sift_intro/py_sift_intro.html#theory)

The largest scale for keypoint corresponds to the largest L2 norm of descriptors. That is the L2 norm of the entire 128 feature vector. L2 norm of a vector is simply the sqrt of the sum of squares. We can also use the one with the largest radius. They both are different.

Keypoint matching

Keypoints between two images are matched by identifying their nearest neighbours. But in some cases, the second closest-match may be very near to the first. It may happen due to noise or some other reasons. In that case, ratio of closest-distance to second-closest distance is taken. If it is greater than 0.8, they are rejected. It eliminaters around 90% of false matches while discards only 5% correct matches, as per the paper.

Each keypoint is a special structure which has many attributes like its (x,y) coordinates, size of the meaningful neighbourhood, angle which specifies its orientation, response that specifies strength of keypoints etc.

(Source: https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_matcher/py_matcher.html)

Algorithm implemented in Python using OpenCV:

- Read the input image river1.raw and river2.raw from the directory using imread()
- Convert the image from color to gray scale using BGR2GRAY
- Implement the sift function using cv2.xfeatures2d
- Detect the keypoints, sift.detect() function finds the keypoint in the images
- Draw the keypoints on the input image using cv2.drawKeypoints
- Find the orientation of each keypoint by using cv2.drawKeypoints
- Draw a circle with size of keypoint and it will even show its orientation
- Compute the descriptors from the keypoints using sift.compute
- Repeat the same procedure for the second input image
- Find the L2 norm using the descriptors

- Find the key point with the largest scale in river1.raw image using the L2 norm
- Find it's closest neighboring key point in river2.raw image using the key point or feature matching, Brute force technique cv2.BFMatcher and cv2.drawMatchesKnn or Flann based technique cv2.FlannBasedMatcher and cv2. drawMatchesKnn
- Display the images using imwrite()
- Write the computed image data array on output.raw file using the fwrite() function

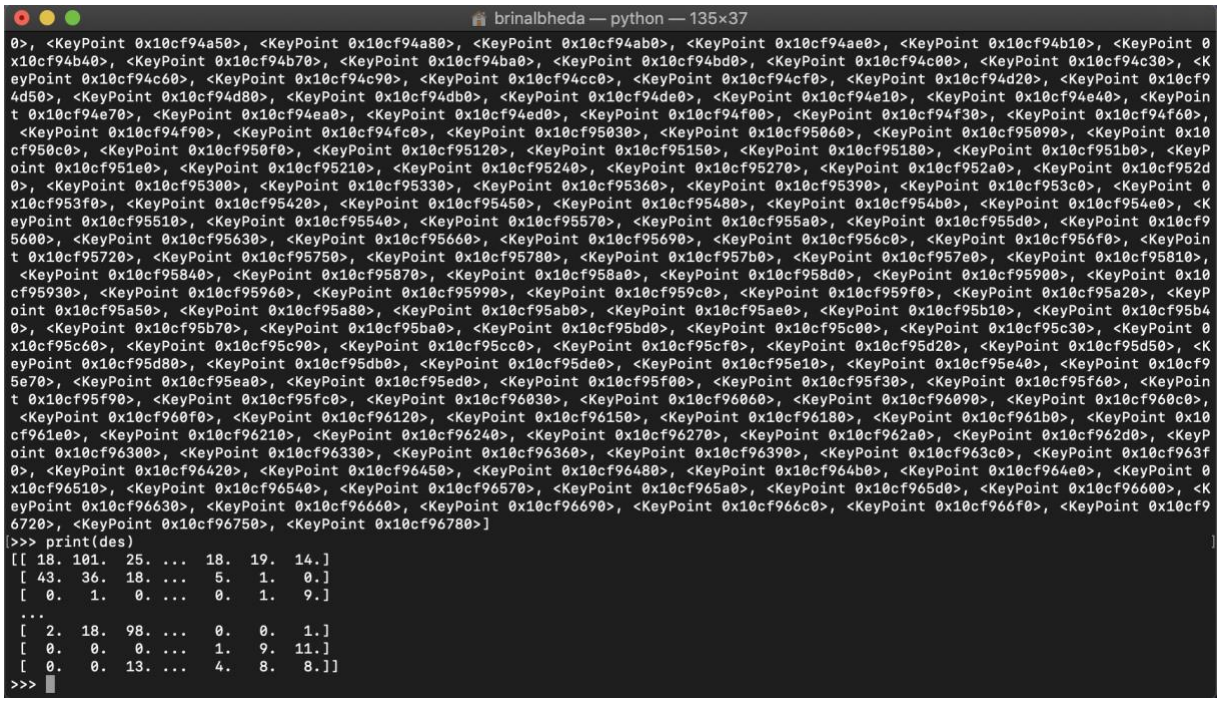
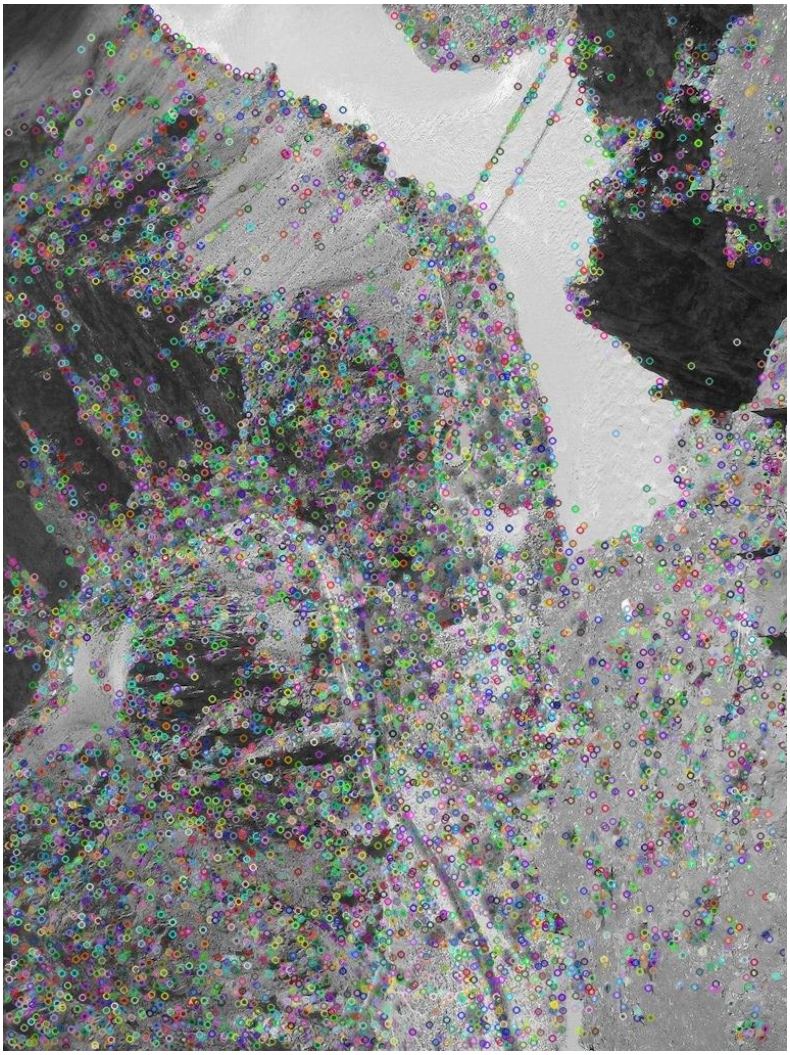


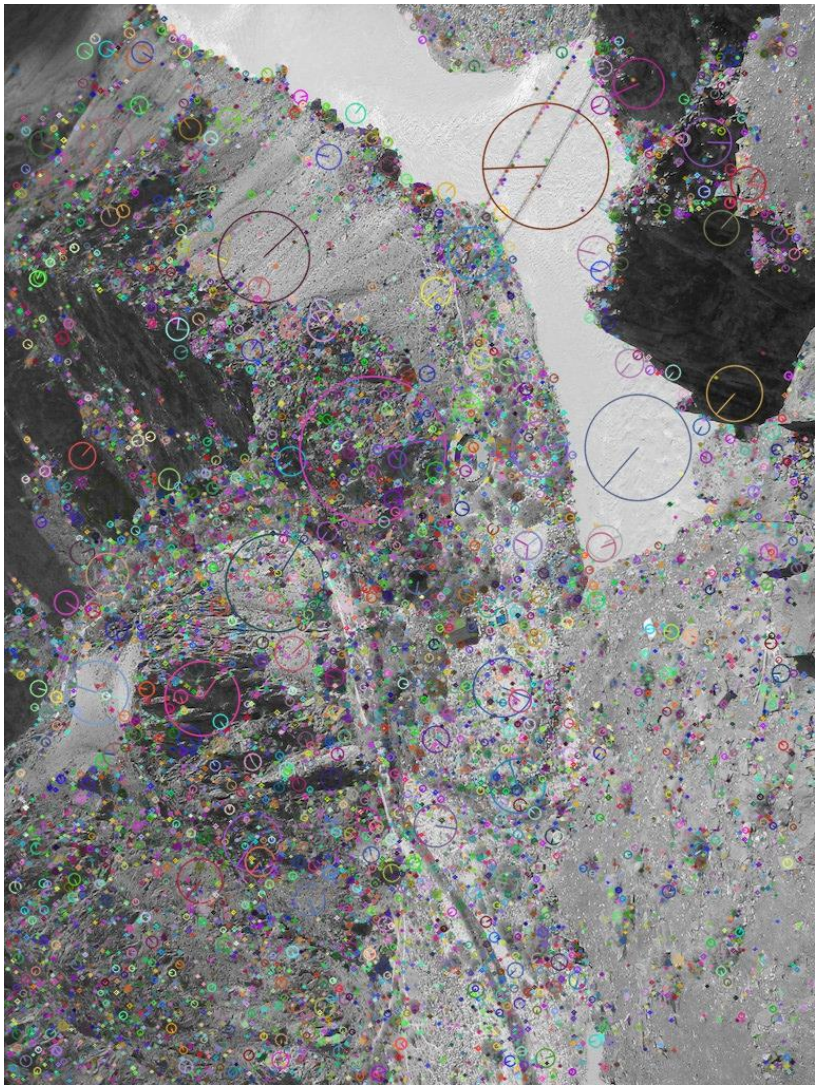
Figure represents the keypoints and descriptors of river1.raw image

III. Experimental Results

Input image – river1.raw

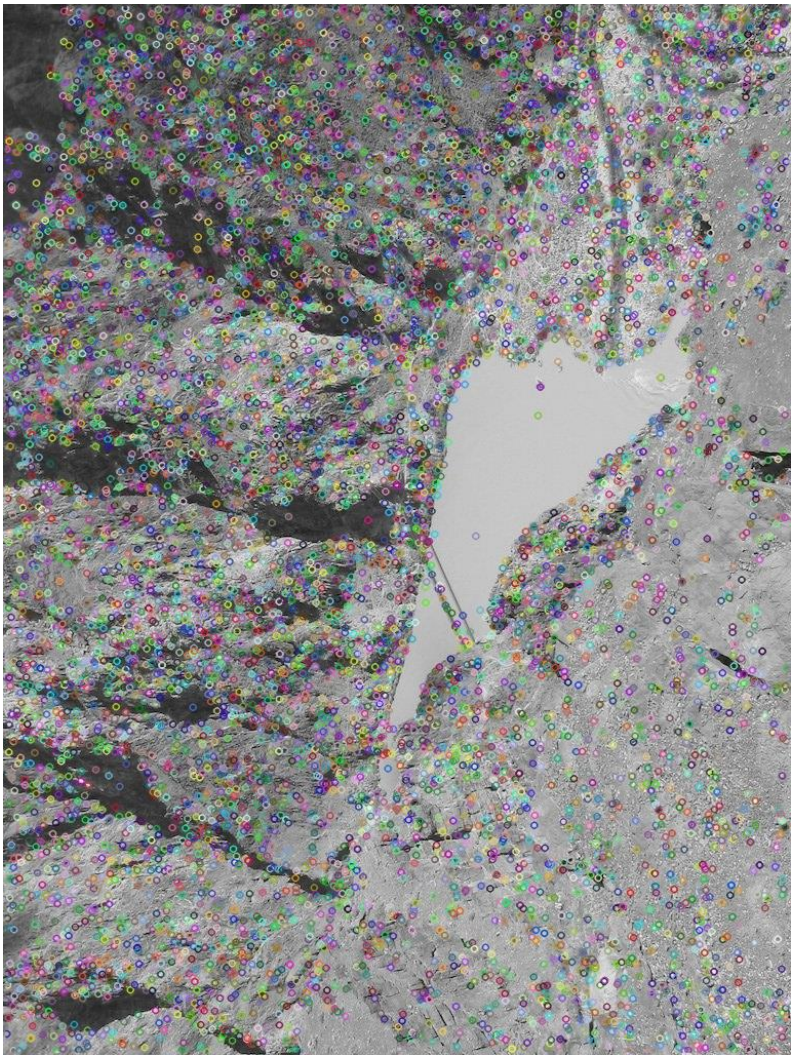


River1.raw image with SIFT keypoints

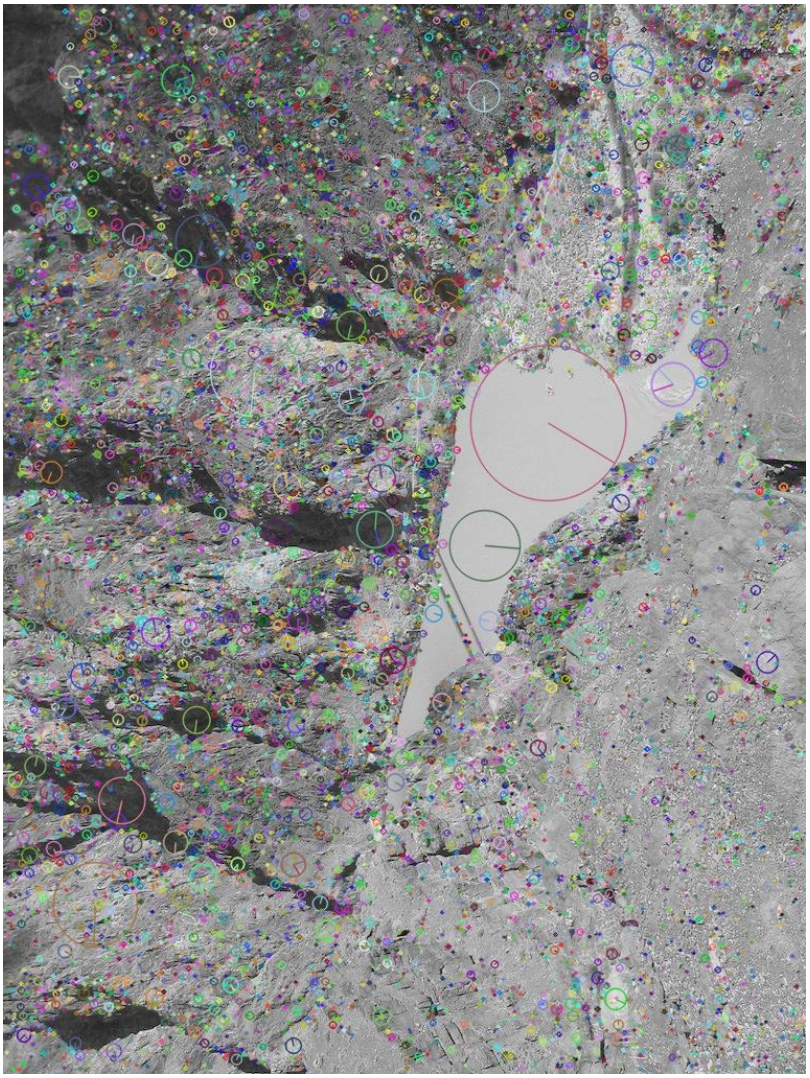


River1.raw image with SIFT keypoints showing orientation

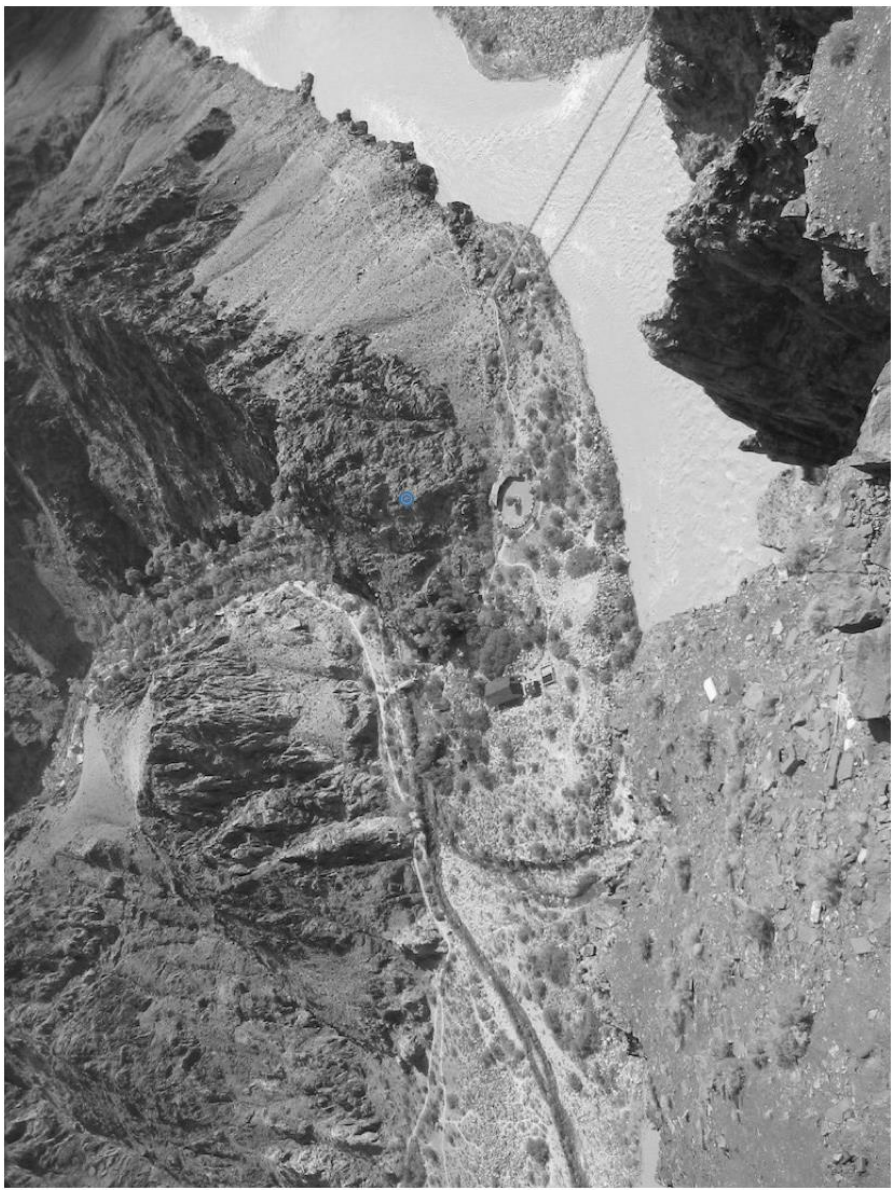
Input image – river2.raw



River2.raw image with SIFT keypoints



River2.raw image with SIFT keypoints showing orientation



River1.raw image showing the key point with the largest scale

Image matching between the two river images for the key point with the largest scale in river1.raw image

IV. Discussion

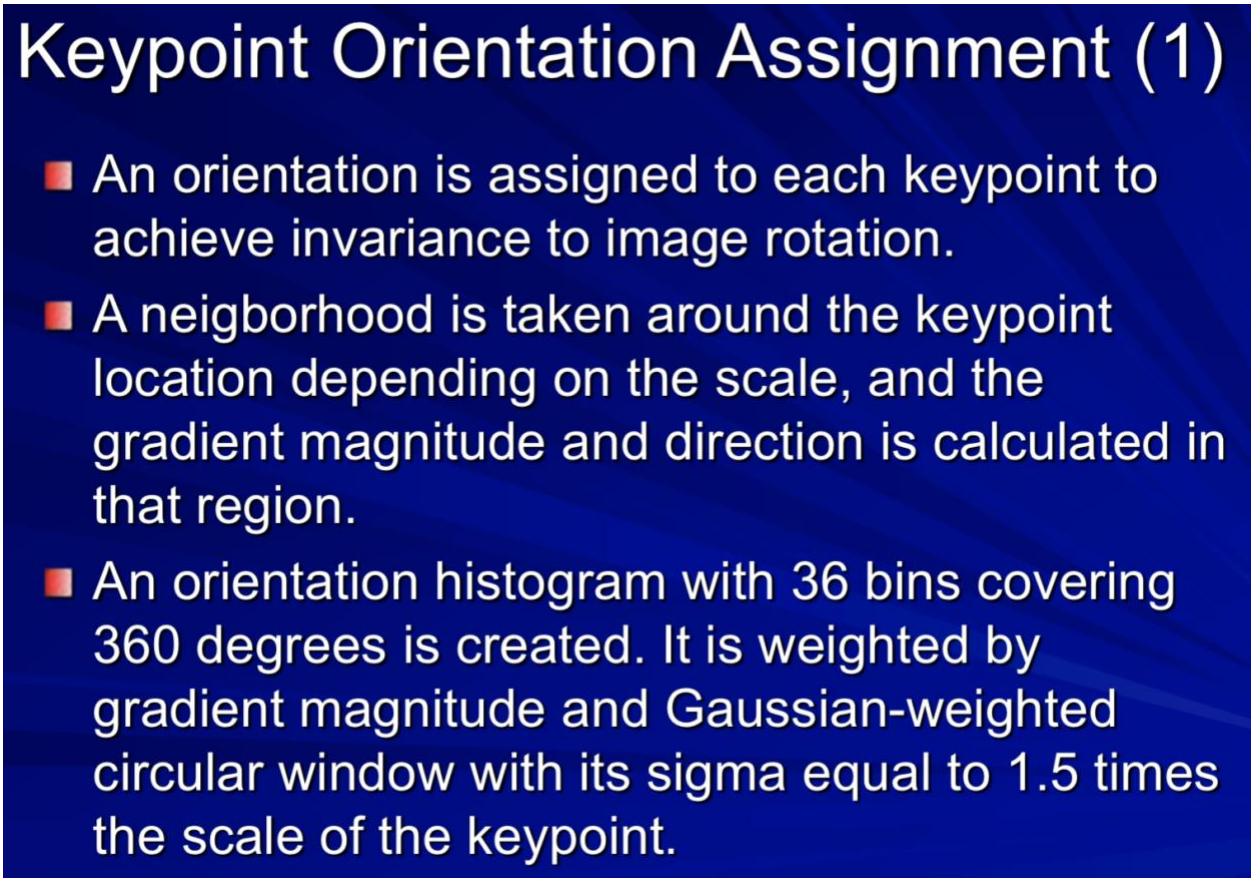
Implementing the open source function code SIFT leads to little different outputs than implementing the method directly from the paper. This is because there are some differences in the key points, also since it is patented the open source code cannot be exactly the same. We see that the output obtained is quite decent and gives good accuracy. However, it is slower.

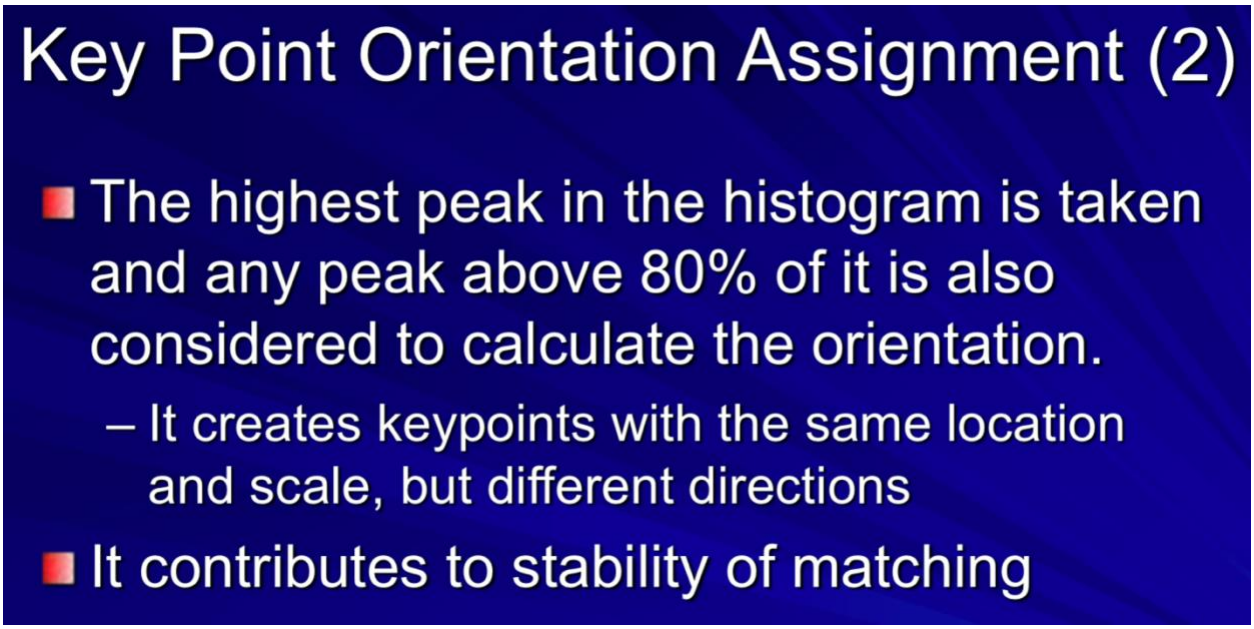
We perform SIFT to find the matching points in two different images. I found the keypoints in both the images storing the descriptors as well. I showed the orientation of each in the output image above. I found the strongest key point in river1 image using L2 norm and also found the matching point in the river2 image using the key point matching by the Brute Force Matcher and Flann (Fast Library for Approximate Nearest Neighbors) Matcher. Flann method is better than Brute Force.

(Source: http://aishack.in/tutorials/sift-scale-invariant-feature-transform-keypoint-orientation/)

(Source: https://www.researchgate.net/post/I_am_trying_to_implement_SIFT_I_have_queries_related_to_orientation_assignment_I_am_giving_current_status_of_my_implementation_with_queries)

Talking about the orientation of key points and the viewing angel, these are different at times. It leads to wrong key points beings matched as the orientation is in a different direction and causes an improper match. The orientation of the key points in SIFT is computed by the weighted orientations of the neighboring pixels. The orientation provides rotation invariance. When the invariance is more, it is better. The calculations done on the key points are done relative to the orientation, as it ensures the rotation invariance. The size of the "orientation collection region" around the keypoint depends on it's scale. The bigger the scale, the bigger the collection region. According to Lowe's original SIFT paper it is possible to have 2 main orientations per keypoint. If there is another histogram bin with a value higher than 80% of the detected peak, a second descriptor is computed with this main orientation.





Here is the paragraphs of the Lowe paper on SIFT:
"An orientation histogram is formed from the gradient orientations of sample points within a region around the keypoint. The orientation histogram has 36 bins covering the 360 degree range of orientations. Each sample added to the histogram is weighted by its gradient magni- tude and by a Gaussian-weighted circular window with a σ that is 1.5 times that of the scale of the keypoint.
Peaks in the orientation histogram correspond to dominant directions of local gradients. The highest peak in the histogram is detected, and then any other local peak that is within 80% of the highest peak is used to also create a keypoint with that orientation. Therefore, for locations with multiple peaks of similar magnitude, there will be multiple keypoints created at the same location and scale but different orientations. Only about 15% of points are assigned multiple orientations, but these contribute significantly to the stability of matching. Finally, a parabola is fit to the 3 histogram values closest to each peak to interpolate the peak position for better accuracy. "

This SIFT algorithm is computationally expensive especially for a larger database. SURF is better to implement as it is 3 times faster as compared to SIFT and takes lesser computation time. However, when there are illumination changes, SIFT performs better than SURF. Hence, it depends on what application is the code being used for and we can decide which method will be good enough to get better results for the given images.

(c) Bag of Words

I. Abstract and Motivation

For object recognition, before CNN, used Bag of Words (BoW). This is also used for image retrieval. To find similar images, create bag of words for input images, also have bag of words for the query images. Then do the hashing, hashing tries to reduce the dimension and also try to find the unique representation. Then compare the hash values. We have one query image and lot of images in the dataset. So compute the bag of words and then hash and find a concise representation for the whole set of images and repeat the same for the query image and try to search which one is similar and thus, do the image retrieval.

(Source: Bag of Visual Words in a Nutshell – Towards Data Science, Bethea Davida)

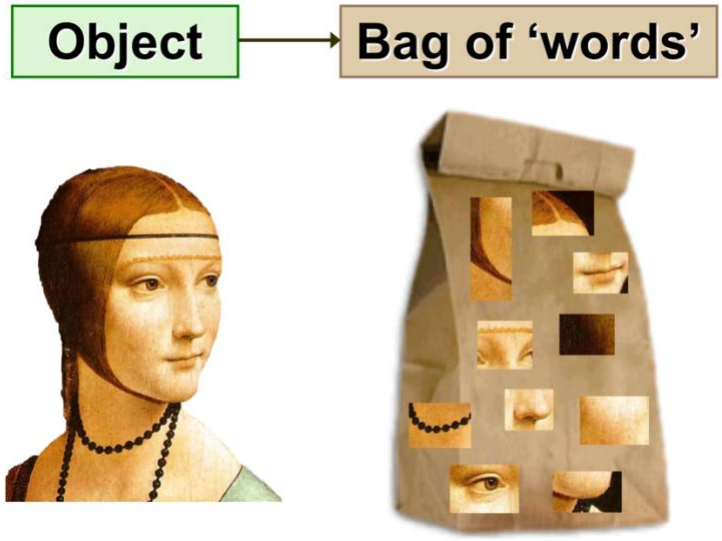
Bag of visual words (BOVW) is commonly used in image classification. Its concept is adapted from information retrieval and NLP's bag of words (BOW). In bag of words (BOW), we count the number of each word appears in a document, use the frequency of each word to know the keywords of the document, and make a frequency histogram from it. We treat a document as a bag of words (BOW). We have the same concept in bag of visual words (BOVW), but instead of words, we use image features as the "words". Image features are unique pattern that we can find in an image.

The general idea of bag of visual words (BOVW) is to represent an image as a set of features. Features consists of keypoints and descriptors. Keypoints are the "stand out" points in an image, so no matter the image is rotated, shrink, or expand, its keypoints will always be the same. And descriptor is the description of the keypoint. We use the keypoints and descriptors to construct vocabularies and represent each image as a frequency histogram of features that are in the image. From the frequency histogram, later, we can find another similar images or predict the category of the image.

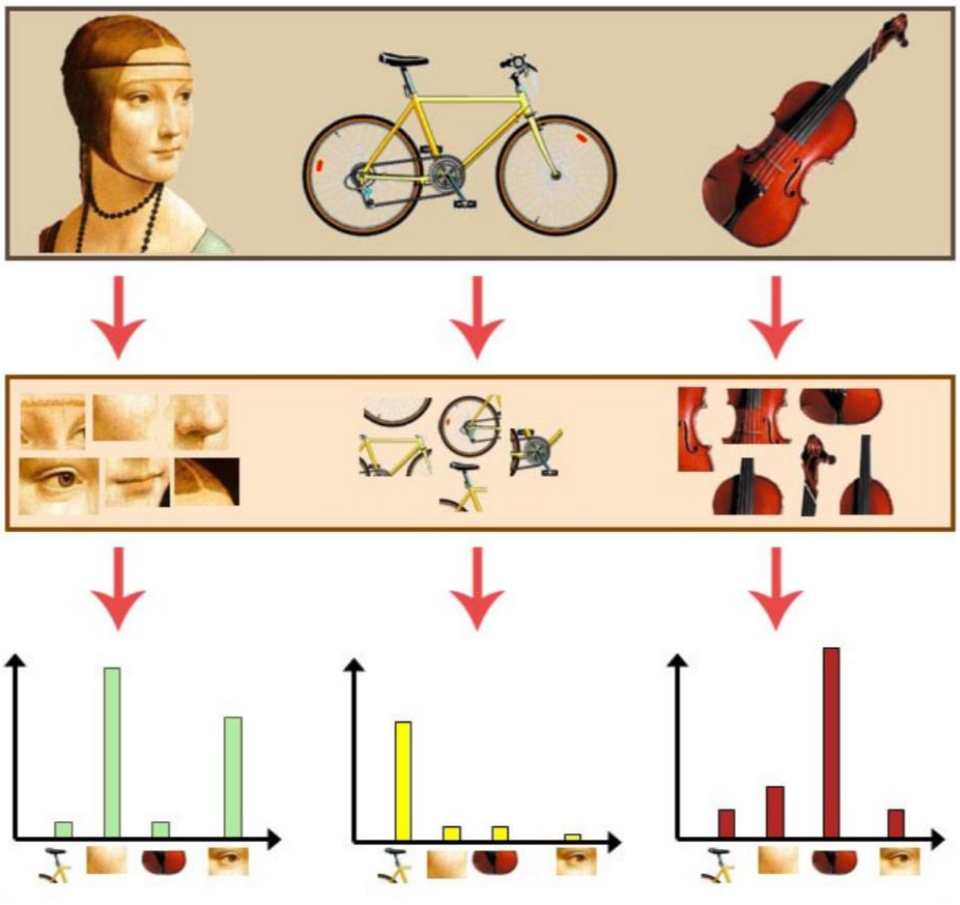
(Source: https://slideplayer.com/slide/5152307/)

This basically contains a bag of visual words (denotes visual units) as shown above. And we want to perform object recognition. Say there is an image of an object and it is decomposed into smaller visual parts say mouth, eyes, nose and so on as shown below.

(Source: Bag of Words – 16-721: Advanced Machine Perception, A. Elfros, CMU, Spring 2009)



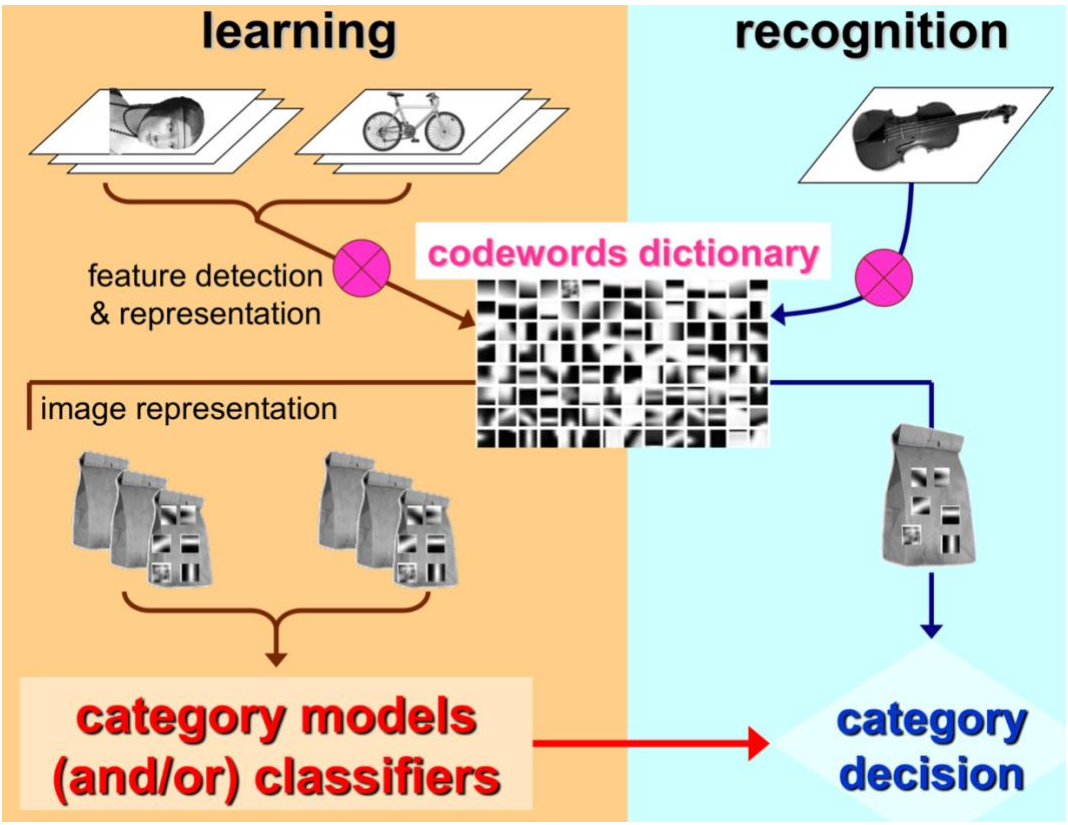
The weakness of bag of words is that they do not differentiate discriminant words. They get idea from the text processing. Say, to retrieve the document, there are a lot of words and look at the histogram of words. But in images it is a bit difficult as there is different scaling and rotation of words. Hence, it is not robust and will not work well.



Histogram of visual words

In the picture above for example, there are different objects and say there is a dictionary containing different visual words (small image patches cut from the original image). Now look at the distribution by doing the histogram. The ones having the largest peak denote that the corresponding patch or words belong to that class. Using the histogram, try to determine the object.

Here below we can see the steps or algorithm for the recognition using the bag or words method:



The patch size cannot be too big. Since the patch size becomes larger and larger the operation, computation and representation become more expensive. Also, it becomes difficult to find another exact match. Compare the different bag of words of each class and find the best match and check which one is more close.

Feature detection is done using a sliding window and generate regular grids. A lot of background words also become a part of the bag of words other than the main object. This leads to two issues – one we collect the patches which have nothing to do with the object and second is we collect the patches from the object which are not discriminating enough. Use interest point detector to get the salient points or key points. So need to pick up points from the object contour. Thus, find the features first and then representation of features. For feature representation, we can use the SIFT descriptors, image patches or filter bank responses (do some transform like DCT or Fourier).

However, bag of words was used a lot for the object recognition applications before the boom of CNN. These days Convolutional Neural Networks are used for image object recognition as they give much better results and they are more robust models.

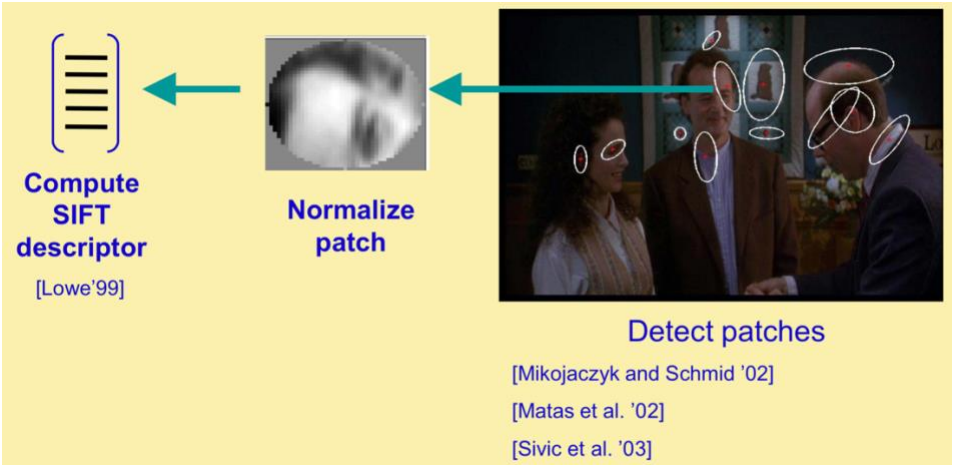
II. Approach and Procedure

(Source: Bag of Visual Words in a Nutshell – Towards Data Science, Bethea Davida)

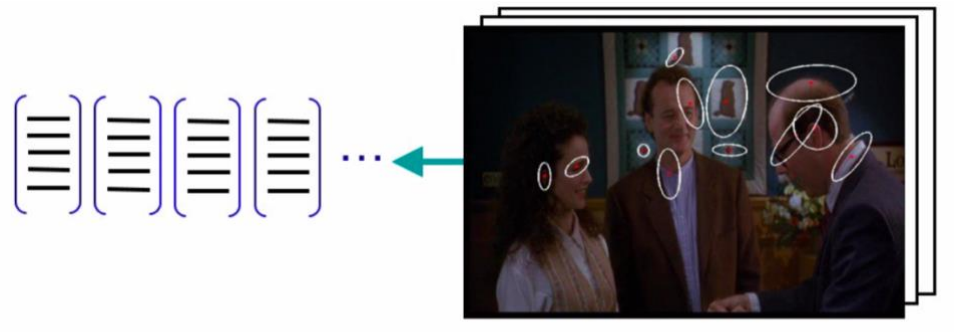
How to build a bag of visual words (BoVW)?

Steps –

Once we get the interest point, get the neighborhood, the normalize that region to 16x16 pixels and use the SIFT computation. In that use grayscale and gradients. Then use the standard description of SIFT embedding and find the orientation and get 128. Thus, get the visual words and compare the histograms.



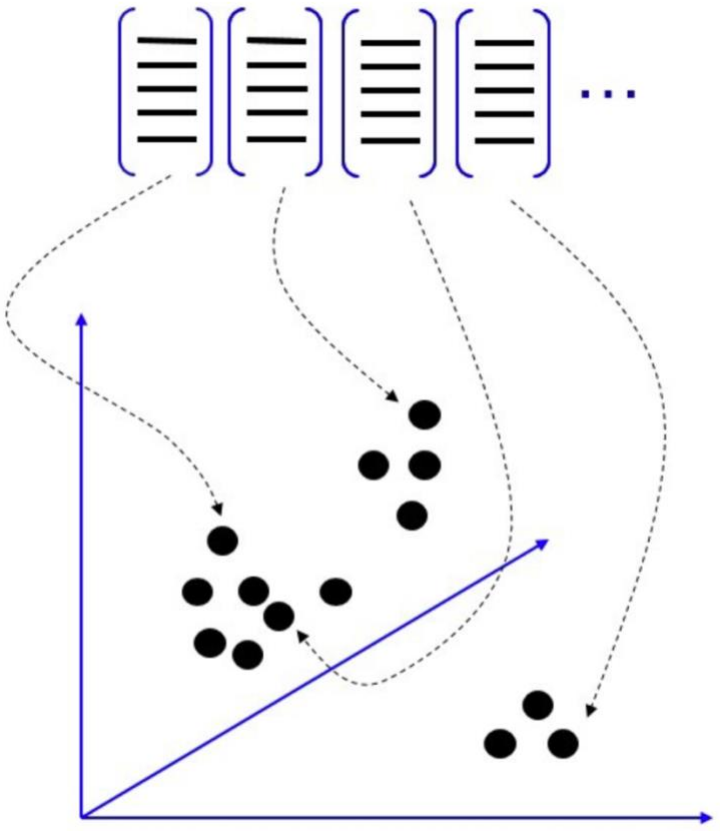
We detect features, extract descriptors from each image in the dataset, and build a visual dictionary. Detecting features and extracting descriptors in an image can be done by using feature extractor algorithms (for example, SIFT, KAZE, etc).



Detecting features and extracting descriptor

Say, given a collection of training images. Build a dictionary. To do this, get the interesting points or key points and then find the SIFT representation (128D or through PCA → 20D). Then cluster some SIFT description with very close distance. Do the k-means clustering in the feature space. Create k visual words, where k = 512, 1024, 2048… and so on. Use multiple training images of the same object class to compute the histogram of the visual words. Thus, called as visual word distribution.

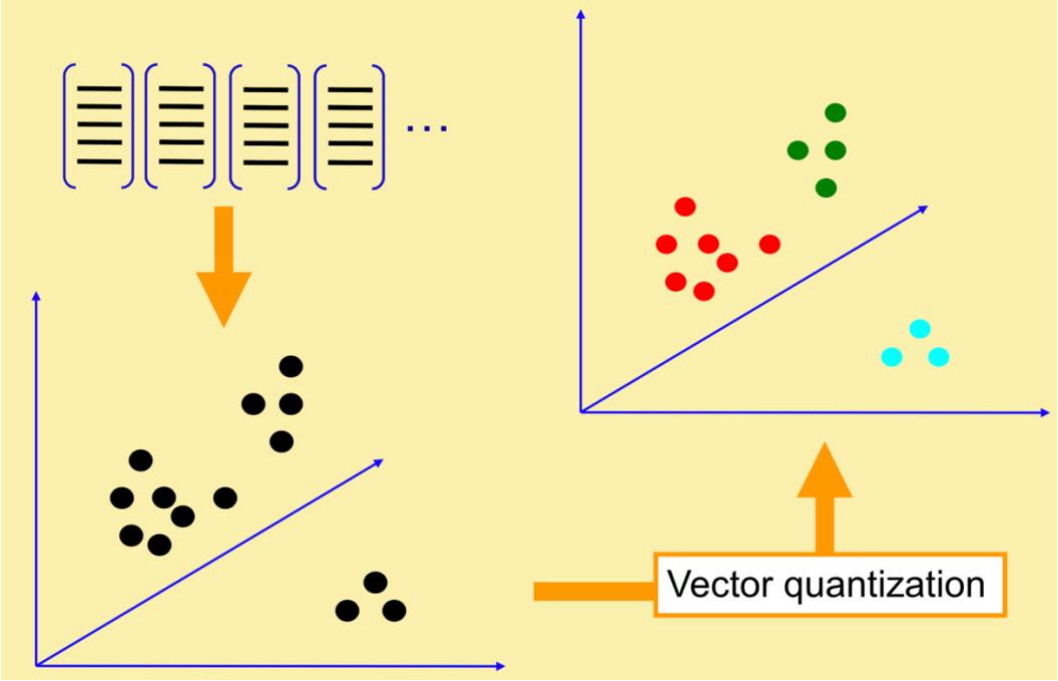
Next, we make clusters from the descriptors (we can use K-Means, DBSCAN or another clustering algorithm). The center of each cluster will be used as the visual dictionary's vocabularies.



Descriptors clustering

Finally, for each image, we make frequency histogram from the vocabularies and the frequency of the vocabularies in the image. Those histograms are our bag of visual words (BOVW).

Cluster them into say 1024 clusters and get the centroid. Centroid becomes the visual words representation. So that is the dictionary from the training data. Then collect the training data histogram.



The above image shows the k-means vector quantization. Vector means high dimensional space. Centroid is the average, the representation of visual words. Thus, we get the clustered image patterns. Project the visual words from the training images. After we collect a number of objects then we project them and see the distribution average. Then collect the patches according to the key points and average them. We get the object representation.

For testing, given test images, compute visual word histogram. Use the histogram intersection method. For a given bin, say consider ith bin, where i = 1, 2, ….1024. Find the minimum and maximum for each class type at that ith bin using the formula below.

$$intersection\ at\ ith\ bin = \frac{\min(X_i A_i)}{\max(X_i A_i)}$$

Here, the weakness is that the visual word is not discriminant features. Also, the spatial relationship between the visual words is lost.

(Source: Question PDF – HW #4)

Given samples of zeros and ones from the MNIST [3] dataset as shown below,

Figure showing images of zeros and ones

Used these images as the training dataset to form a codebook with bin size of 2 (two clusters for k-means clustering). Then extract the SIFT feature vector for the image of eight and show the histogram of the Bag of Words for this image. Discussed the observation.

Figure showing image of eight

(Source: https://gurus.pyimagesearch.com/the-bag-of-visual-words-model/)

Building a bag of visual words can be broken down into a three-step process:

- **Step #1:** Feature extraction
- **Step #2:** Codebook construction
- **Step #3:** Vector quantization.

Step #1: Feature extraction

The first step in building a bag of visual words is to perform feature extraction by extracting descriptors from each image in our dataset.

Feature extraction can be accomplished in a variety of ways including: detecting keypoints and extracting SIFT features from salient regions of our images; applying a grid at regularly spaced intervals (i.e., the **Dense** keypoint detector) and extracting another form of local invariant descriptor; or we could even extract mean RGB values from random locations in the images.

Step #2: Dictionary/Vocabulary construction

Now that we have extracted feature vectors from each image in our dataset, we need to construct our vocabulary of possible visual words.

Vocabulary construction is normally accomplished via the **k-means clustering algorithm** where we cluster the feature vectors obtained from **Step #1**.

The resulting cluster centers (i.e., centroids) are treated as our *dictionary* of visual words.

Step #3: Vector quantization

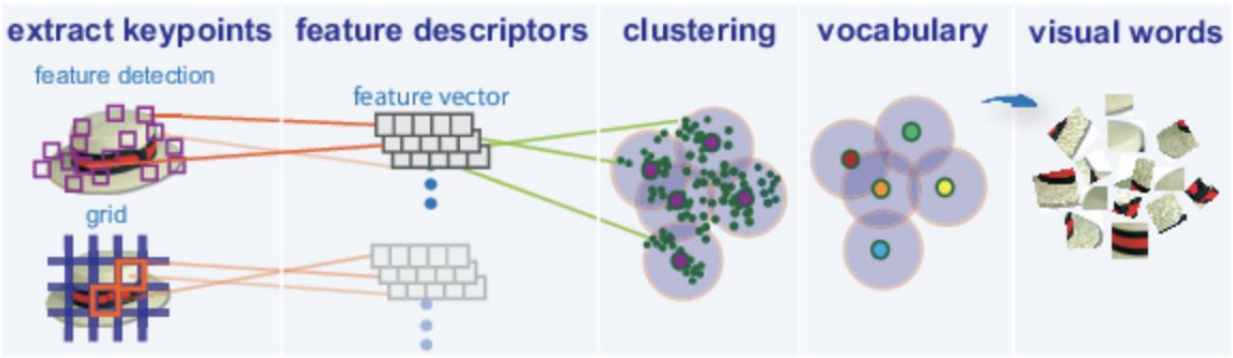
Given an arbitrary image (whether from our original dataset or not), we can quantify and abstractly represent the image using our bag of visual words model by applying the following process:

1. Extract feature vectors from the image in the same manner as **Step #1** above.
2. For each extracted feature vector, compute its nearest neighbor in the dictionary created in **Step #2** — this is normally accomplished using the Euclidean Distance.
3. Take the set of nearest neighbor labels and build a histogram of length *k* (the number of clusters generated from k-means), where the *i*'th value in the histogram is the frequency of the *i*'th visual word. This process in modeling an object by its distribution of prototype vectors is commonly called *vector quantization*.

Finally, given the bag of visual words representation for *all* images in our dataset, we can apply common machine learning or CBIR algorithms to classify and retrieve images based on their visual contents.

Simply put, the bag of visual words model allows us to take highly discriminate descriptors (such as SIFT), which result in *multiple* feature vectors per image, and *quantize* them into a single histogram based on our dictionary. Our image is now represented by a *single* histogram of discriminative features which can be fed into other machine learning, computer vision, and CBIR algorithms.

(Source: https://www.mathworks.com/help/vision/ug/image-classification-with-bag-of-visual-words.html)



The data is in the interval [0,255] and is of data type single if using VLfeat.
vl_sift gives out features and descriptors. Features have the keypoint locations, scale and orientation.

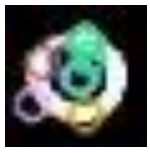
Algorithm implemented:
Resize all the ones and zeros input images as well as the eight image to a larger size to get the sift features extract in a good way.
Load the training data images.
1. Find certain key-points and descriptors (extract features) using SIFT from training data.
2. Use kmean (with k=# of classes, in this case k=2) on the key-points. First initialize the cluster. Add descriptors to bag of visual words. The resulting centroids are put into a "codebook".
3. Now, get eight.raw key-points using SIFT, find their closest centroid in the codebook and plot the "count of closest centroid" histogram.
The histogram for the bag of words shows the number of counts for each centroid in the trained codebook.

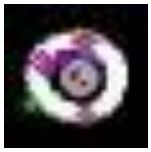
III. Experimental Results

Input images – ones.raw

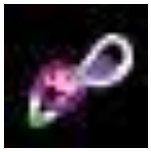
Input images – zeros.raw

Input image – eight.raw

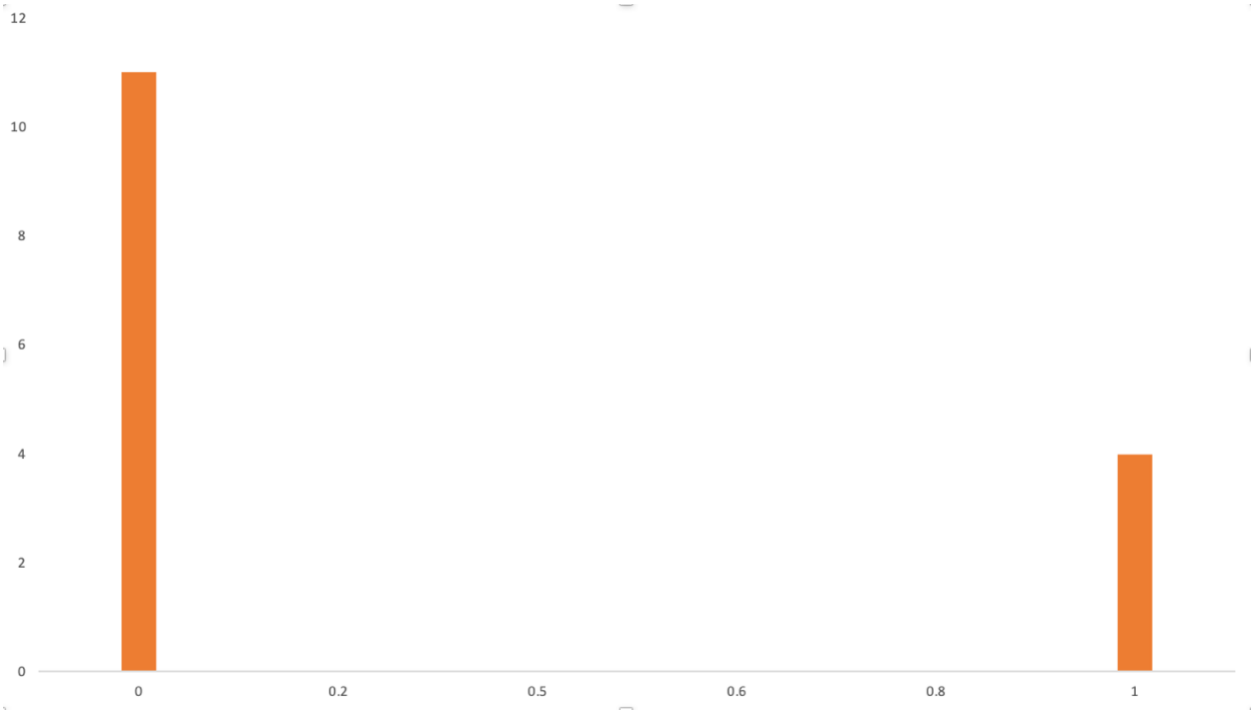




one.raw and zero.raw images with SIFT keypoints and showing orientation



eight.raw images with SIFT keypoints and showing orientation



Histogram of bag of words showing the classes or cluster number i.e. 0 and 1 versus the count of features i.e. count for each cluster number

IV. Discussion

The Bag of Visual Words model is implemented to find the observation on the image eight.raw to find which features are closest to the training images of ones and zeros. The output obtained above as we see, is quite similar to what I thought logically. 8 is similar to 0 more than 1. This is because it has 2 curved circles like 0 has one curved circle, whereas 1 is a straight line having a steep edge which is not so similar looking to 8. Thus, the output got is quite satisfactory.

The corresponding histogram is also plotted showing the application of Bag of Words and we infer that the eight image has more number of features similar to 0 cluster i.e. 11 and lesser number of features similar to 1 cluster i.e. only 4. Here we do histogram matching instead of key point matching for comparison between the closeness of the two images.

Sometimes, we see even if the images are quite similar, there is a difference in the frequency of values in each bin. This happens because of the orientation shift in both the images as I had stated this point, in the discussion of 2(b).

However, the answers would be even more accurate and robust model if there were greater number of training data images available.

Overall, Bag of Words works pretty well and gives accurate and efficient results and hence, was considered as the best approach for Image matching, feature comparison, object recognition and image retrieval, before the boom in CNN which is the state of art for many Computer Vision applications.