EE569: Homework #4

Yifan Wang
wang608@usc.edu #3038184983

March 4, 2019

1 Object Analysis

1.1 Texture Classification

Before extracting features some pre-processing is needed. To avoid disturbance from luminance change and contrast variation, histogram equalization from $HW1$ is applied to all 12 images, after that it can be found that $texture3, 9$ are actually the same (visualized in $Appendices A.1$ $Figure 11$). Then subtract mean (DC part).

25 Laws filters are applied on zero mean images (normalized energy maps of each filter are shown in $Appendices A.1$ $Figure 12, 13, 14$). Frequency response of five $1D$ Laws filters are shown in $Appendices A.1$ $Figure 10$. Their functions and cut-off frequencies are as follows:

- L5: Low pass filter, cut-off frequency at $\pi/2$;
- E5: Band pass filter, lower cut-off: $\pi/10$, upper cut off: $7\pi/10$;
- S5: Band pass filter, lower cut-off: $\pi/5$, upper cut off: $4\pi/5$;
- W5: Band pass filter, lower cut-off: $3\pi/10$, upper cut off: $9\pi/10$;
- R5: High pass filter, cut-off frequency: $\pi/2$;

Convolving each $2D$ filter with one image ($H \times W$), a $25 \times H \times W$ feature stack is achieved, then averaging each feature from $H \times W$ to one single value, result in a feature vector (length 25)that represents this texture image. Repeat this process for all images, a 12×25 feature vector is derived. To make each feature same weight, features are whiten among column (zero mean, unit standard deviation). They are visualized in $Figure 2$ (normalized using $255 * \frac{v-min(v)}{max(v)-min(v)+10e-7}$).

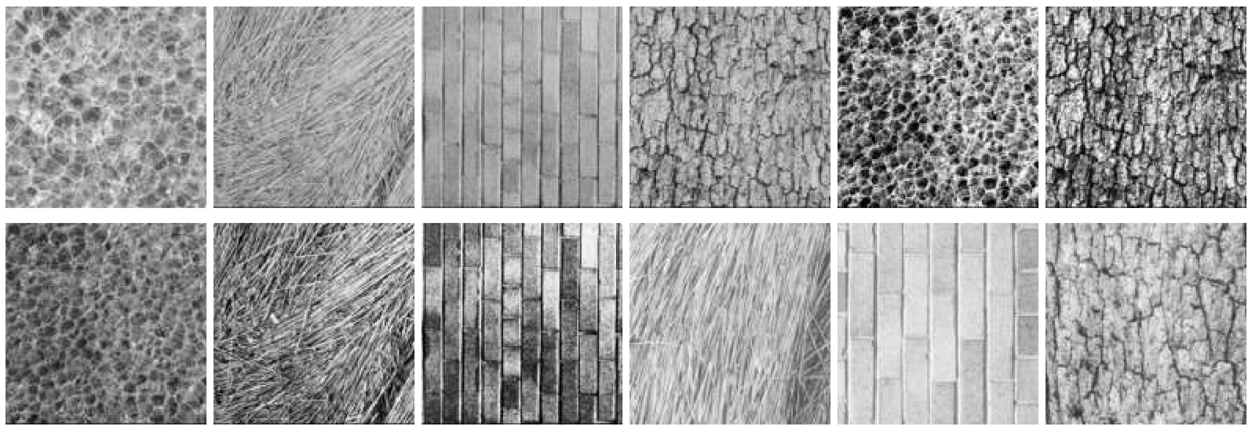
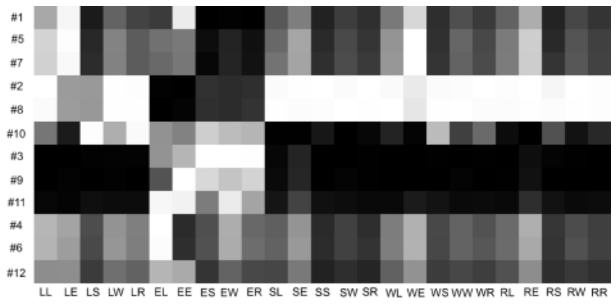
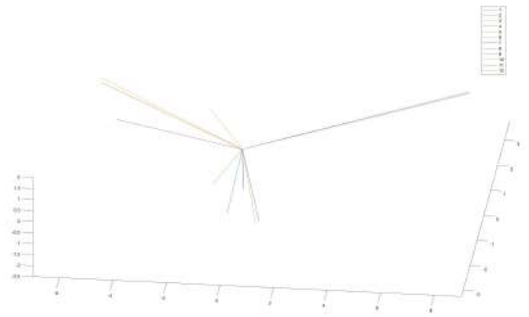


Figure 1: $texture1 - 12.raw$



(a) Normalized Feature Map



(b) $3D$ feature visualization

Figure 2

To decide which feature have strongest discriminant power, it can be found from $Figure2(a)$, where x-axis represents different filters' responses, y-axis is different images. Same feature is normalized to grayscale (normalized along y-axis). To determine which filter have most discriminant power, it can be understand that each feature is one dimension, clusters must be separated in order to successfully discriminate them . For example, a filter's response on 4 image is $1, 2, 4, 6$, ground truth separation is $\{1, 4\}$, $\{2, 6\}$ which beyond K-Means algorithm's ability, so that this filter would have less discriminant power. While consider another filter generates $1, 2, 5, 7$ with ground truth $\{1, 2\}$,$\{5, 7\}$, it has much powerful discriminant ability. Which is the same case in deciding these 25 filters. In $Figure2(a)$, each true classes are arranged together. It can be found that $L5R5$ filter is the answer. Since grayscale is separable among each different cluster.

Cluster	Result#1	Result#2	Result#3	Result#4	Result#5
#1	1, 5, 7	1, 5, 7	1, 5, 7	1, 2, 5, 7, 8	1, 5, 7
#2	2, 8	2, 8	2, 8	3, 9	2, 8
#3	3, 9, 10, 11	3, 9, 11	3, 9, 11	4, 6, 10, 12	3, 9, 10, 11
#4	4, 6, 12	4, 6, 10, 12	4, 6, 10, 12	11	4, 6, 12

Table 1: Texture Clustering using 25 features

Cluster	Result#1	Result#2	Result#3	Result#4	Result#5
#1	1, 5, 7	1, 5, 7 ,12	1, 5, 7	1, 5, 7, 10, 12	1, 5, 7, 12
#2	2, 8	2, 8	2, 8	2, 8	2, 8
#3	3, 9, 10, 11	3, 9, 11	3, 9, 11	3, 9, 11	3, 9, 11
#4	4, 6, 12	4, 6, 10	4, 6, 10, 12	4, 6, 10	4, 6

Table 2: Texture Clustering using 3 features

Then K-Means [1][2] algorithm (realized in $basic/KMeans.hpp$) is applied on both 25 dimension features and 3 dimension features ($Figure2(b)$ using PCA from OpenCV). In this process, K-Means is initialized by choosing randomly 4 feature vectors (uniform distribution). If using maximum distance initialization [3], result trends to be stacked by some extreme images. $Table1, 2$ show 5 random initialization result. $Table3$ shows ground truth and clustering accuracy.

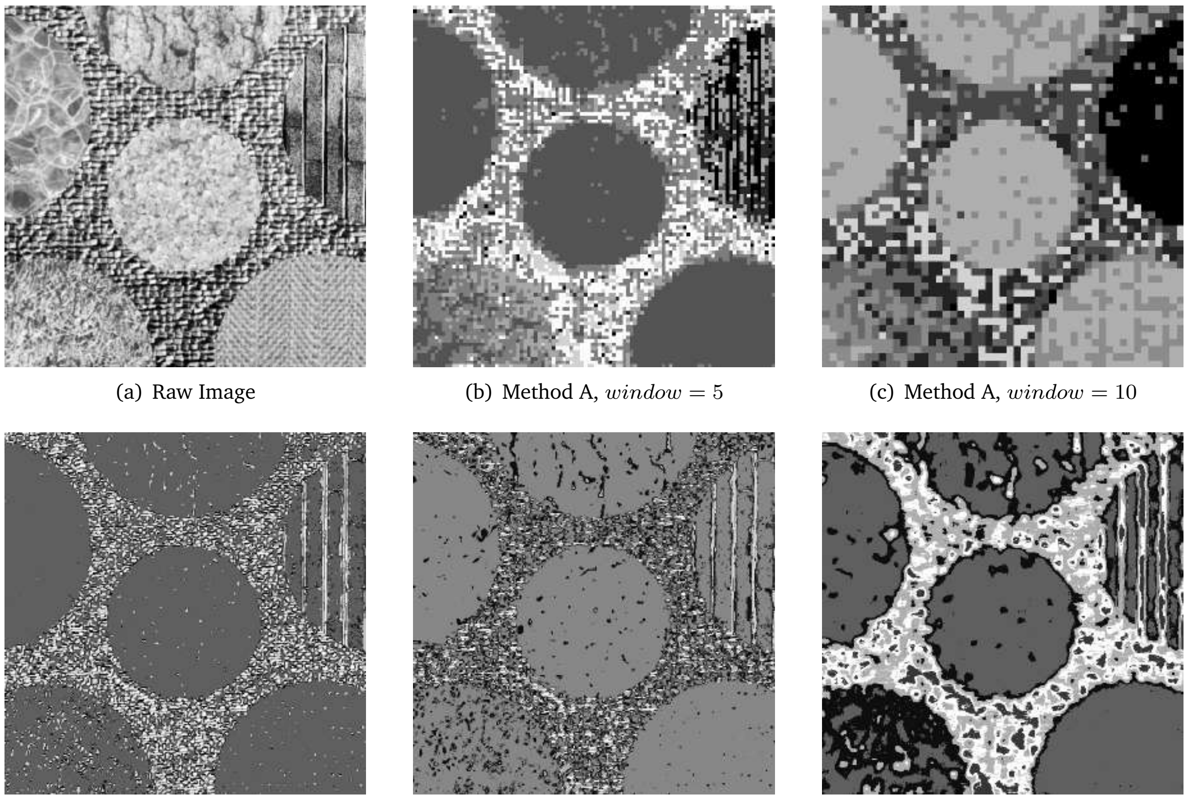
From this result, it can be found generally a good clustering result can be derived. Most error comes from $texture10$ and $texture11$, especially $texture10$. Reason for this might that in $texture10$ right bottom corner is darker than other part, even though textures are the same, un-uniform brilliance in some parts of this image would have impact on result.

While considering using how many features and the function of PCA, it can be found from $Table3$ that it not true more feature would give a better result. After using PCA, most energy is concentrated in these 3 features (part of noise energy can be rejected, rather than least mean square method need to have a feature to be supposed noise-free, in PCA all features can by noisy. Due to the independence of each noise, their energy would not accumulate to one direction, which means most of them can be rejected when choosing largest 3 energy features). Lead to a good performance. Using only 3 feature would make K-Means more robust (less noise) and converge much faster (less computation).

Texture	True Label	25 features			3 features		
		Precision	Recall	F score	Precision	Recall	F score
$bubbles$	1, 5, 7	0.92	1	0.96	0.84	1	0.894
$straw$	2, 8, 10	0.8	0.53	0.64	1	0.67	0.802
$brick$	3, 9, 11	0.92	0.93	0.92	0.95	1	0.974
$bark$	4, 6, 12	0.88	1	0.936	0.82	0.8	0.810
Overall		0.88	0.87	0.875	0.90	0.87	0.885

Table 3: Texture Classification

1.2 Texture Segmentation



(a) Raw Image (b) Method A, $window = 5$ (c) Method A, $window = 10$

(d) Method B, $window = 11$ (e) Method B, $window = 31$ (f) Method B+feature blur, $window = 31$

Figure 3: Segmentation Result

Texture segmentation mainly has the same flavor as previous one, using same method to extract 25 texture features and normalized by $L5L5$ filter's response. Then to compute label of each pixel, K-Means should be perform on each pixel (for computational efficiency and noise redution, features are reduce to three dimensions through PCA). Doing directly on normalized features would result in a mess segmentation which is caused by different energy in different feature in a same texture. In that case, some average is need to get average energy for a pixel in a local region. Here 2 different averaging methods is used.

$MethodA$: Separate image in to small patches to use an average energy vector represent this region, one get clustered label, assign the same label to all pixel in this region.

$MethodB$: Using the one provide in discussion section, replace energy vector by using average energy in a local region. Averaging is weighted by Gaussian ($\sigma = 50$).

Result are show in $Figure3$, for $MethodA$ is not acceptable to using a large average window, otherwise would lower image resolution lead to a mosaic segmentation (less accuracy on edges). Small windows can get a nice result, even though with some extreme case. While considering $MethodB$, a relative large window can be used, however, even though it can perform well in texture center, several part would occur in two texture border which can be regard as mix two texture and generate a new texture. This would be a negative factor that reduce performance. And lots of noisy points appear on images. Besides, due to the existence of some extreme noisy point, some different textures would be clustered as the same.

1.3 Advanced Texture Segmentation Techniques

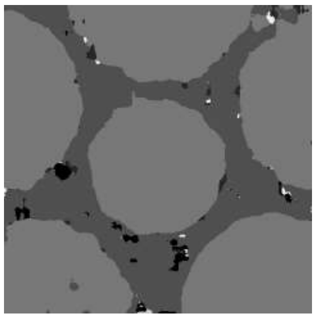


Figure 4: Multi-Scale

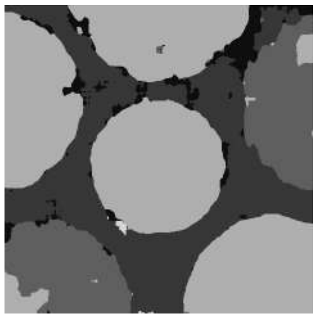


Figure 5: Multi-Scale + Iterative Clustering

1.3.1 Averaging Same Texture Only

From previous section, it can be found major problem occurs when averaging across multi-textures. To address this problem, besides using Gaussian weight, an energy weight is used as well which is inspired by Non-Local Mean.

1.3.2 Label Re-arrangement

There would have this case that some extreme cases in one texture would be assigned another label in a small local region, to solve this problem, using some merging methods. A score based on neighbour pixels' label is computed and using a threshold to rearrange a pixel's label if its neighbour are another labels or simply using voting method in a local region.

1.3.3 Multi-Scale Label Combination

This is inspired by both $MethodA$ and B, A separate texture nicely, while B gives a smooth edge on texture border while can not separate well. The idea is that combine both method together, compute average energy for each pixel as $MethodB$, and several different window size in $MethodA$. Concatenate all these features for K-Means. After get labels, combine these labels through voting method.

Realized with the combination of $Section 1.3.1$ and $Section 1.3.2$ which adds a feature weight ($\sigma = 50$) when using averaging and label re-arrangement. From $Figure 4$ which uses 2 different size layers, it can be found that some small holes are merged to some extent, and edges are merged as well which improves segmentation result.

One drawback of voting method is that it would result clusters less than required. Another is this idea is so fancy that there are so many parameters needs to be specify. Some better result would be achieved if discover more on these parameters. Besides, only one downsample is used, it is likely that using more different scale images would produce better results.

1.3.4 Iterative Clustering

Rather than clustering pixel features directly to 7 clusters. Iterative method can be used. First clustering pixel features to N_1 clusters ($N_1 > 7$), generate labels, than compute average features of a pixel only using the one belong to same cluster in a window (weighted by Gaussian). Then cluster pixels with new average features into N_2 clusters ($N_1 > N_2 > 7$). Repeat this process until $N_n = 7$.

Another similar idea is rather than compute new average feature for each pixel, only compute one average feature to represent pixel belongs to same cluster and connected to each other. This requires less computation than above idea. However, it seems that error would accumulate and would not have any chance to be corrected.

First method is realized, result is shown in $Figure 5$. Edge effect is reduced as well. It performs better than raw method. Besides, to give this method more ability to correct some error clustering, downsampling method is used. This method averaging features in a window without considering labels which serves as adding some noise to make it more robust. This method produce nice segmentation result, however, it can be found after label re-arrangement, remaining cluster is less than what we need as above section, and some textures are clustered in to same class.

1.3.5 Largest K clusters

To solve above problem that label re-arrangement would cause clusters less than number of clusters (K) needed which comes from some extreme noise account for one single cluster. Let K-Means cluster in to K_1 clusters ($K_1 > K$) then using label re-arrangement method. After that, choose K largest clusters as final result, other small clusters can be assigned to its neighbours label. Or it seem possible to using an idea I called $K + 1$ Means. Using some justification method to put some ambiguous data one rubbish cluster, while using normal K-Means in other K clusters. Here are some possible justification ideas: (1) one data point is almost equally close to several cluster center, (2) point whose distance is too far from cluster center compared with minima distance in this center (avoid let these points contributes too much to new cluster center), (3) clusters have less than particular points (in large dataset, they tend to be noise).

2 Image Feature Extractor

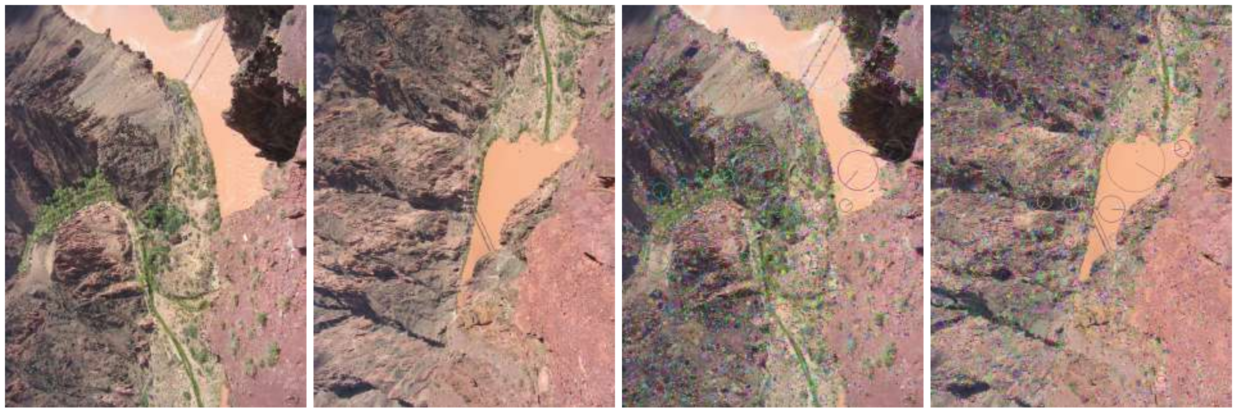
2.1 SIFT

Features extracting by SIFT are invariant to image scaling, translation and rotation. Besides, these features are robust to uniform distortion (partly affine distortion), viewpoint change, addition of noise and illumination variation.

General processes of SIFT are as follows:

Given an image, generate multi-scale images through multi-scale sampling (down sampling and one up sampling) to form an image pyramid. Then convolve these images with Gaussian filters with different σ, for example N ($k_1\sigma, k_2\sigma ... k_N\sigma$) on each image scale, resulting N different level blurred images in each scale.

Then subtract two neighbour blurred image in same image scale to approximate difference of Gaussian.



(a) from first octave (b) from first octave (c) All Keypoints (d) All Keypoints

Figure 6: SIFT on river images

Considering DoG and LoG, in mathematics [7]:

$$LoG(x,y,\sigma) = \sigma^2\nabla^2 G \approx \frac{G(x,y,k\sigma) - G(x,y,\sigma)}{\sigma^2(k-1)}$$

which means that DoG can approximate to scale-normalized LoG ($\sigma^2\nabla 2G$). It has been proven that maxima and minima of $\sigma^2\nabla^2 G$ provide most stable image features.

The reason why using DoG rather than LoG is that LoG requires far more computation than DoG. Suppose there K image used for find minima and maxima points, for DoG, needs $K+1$ convolution with different σ and subtract two neighbour image. While directly using LoG only need S convolution, but generating LoG kernel would be computational heavy (more multiplications and additions (about twice more from some rough tests)). Due to the fact that both method can achieve same effect, using DoG would save much time and computational resources.

Besides, to get more stable features, pre-smooth is need. However, rather than using some filters, they doubling input image using linear interpolation to achieve same effect, while obtaining 4 times more feature points.

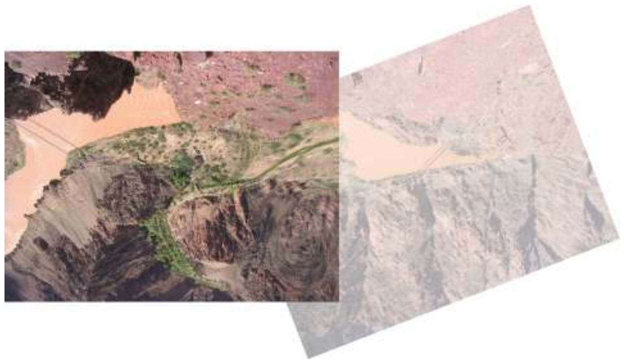
Above process would result $N-1$ difference image in each size. Key points are selected if they are local minima or maxima of its eight neighbours and 9 corresponding neighbours on neighbour scale. This selection criterion lead to its scale invariant characteristic.

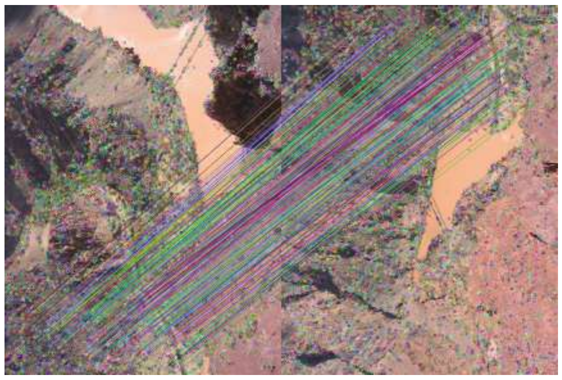
In 1999 paper [4], key points are located in where it being found. While in 2004 paper [6] it use a improved method [8] which can enhance matching accuracy and stability. Key points are accurately localized by using a truncated Taylor expansion. During this stage points low in contrast are discarded because of their high senseity to noise.

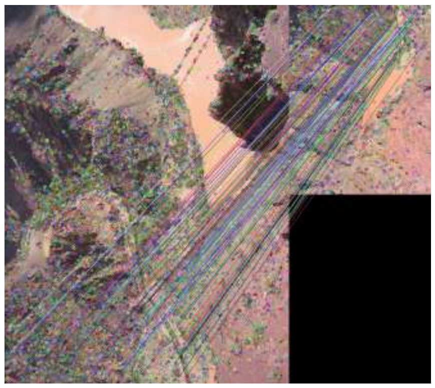
Edge points are selected by compute largest two eigenvalue of Hessian α, β, $r = \alpha/\beta$, if r is larger than $r_{th} = 10$, they would be discarded. This process can be explained that removing points on edges and only keep edge points locates on corners. For example, a white image with a black square on it, due to DoG's function, all the edges can be detected. Key points without any selection would be a contour of square. When using these point for key point matching, lots of mismatch would occur since most key points on same border (except corners and points near corners) are equal, and can match with each other. Key point selection serves as rejection these points and only keep 4 vertexes which can provide correct matching results.

To make remaining key points to be rotation invariant, each one is assigned one or several orientation. A gradient magnitude and direction is computed among all of its neighbours. Neighbours' orientation are combined with both gradient and Gaussian weight and separated in to 36 directions. Key points' orientation is defined as $Peak$ larger than $0.8 \times max(Peak)$. This also contributes to stability of matching.

To reduce the effect from changing in illumination and contrast, vector is normalized to unit length to cancel change of contrast. Since contrast change comes from a constant times pixel value, and brightness


(a) hand matched ground truth


(b) SIFT good matched, $r_d = 3$


(c) SIFT $r_d = 2$, downsample by 2

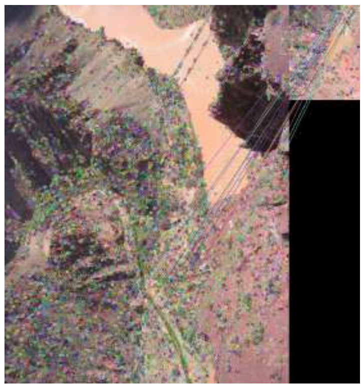

(d) SIFT $r_d = 3$, downsample by 4

Figure 7: SIFT matching

change comes from adding a constant. The first one can be conceal using normalization, while second one would not effect gradient. In that case, SIFT would be affine illumination invariant. When non-linear illumination occurs, it would affect some gradient magnitudes (less likely to affect orientation). To reduce this effect, they thresholding gradient values to be no larger than 0.2, and renormalize them. In that case, large gradient would not contribute more when it is superior to threshold.

SIFT descriptor are created to make each point unique. They are computed from a 16×16 neighbourhood (on image scale key points derived). Each 4×4 sub-region computes a 8 direction orientation. With total 16 sub-region (weight by Gaussian and normalized as well), a 128 length descriptor is derived. Leading to good matching result, since this descriptor is robust to change of illumination and $3D$ viewpoint. Above parameters come from experience, if more orientation or sub-region added, it would be likely to cause some negative effect. Through experiment, these descriptor generate brilliant performance even when database is super large. In 1999 paper, it uses a 160 elements vector from 128 vector above and $8 \times 2 \times 2$ elements from a higher level scale.

2.2 Image Matching

OpenCV' SIFT is used for feature extraction. Lots of features can be derived from rive image ($Figure6$). ORB and SURF features extractors are used as well ($AppendicesB.1\ Figure16$).

Due to its scale invariant propriety, even using a downsampled image ($Figure7(c,d)$), SIFT feature performs well even shrink it 4 times. Orientations of one single feature point in different image remain reactively the same even with huge scale change which contributes to correctly matching. But it can be found that with the shrinking of scale, number matching points become less, but with carefully choosing r_d, its performance remains pleasing.

From comparing of these different feature extractors. They are all robust to change of rotation, noise, viewpoints, and distortions (uniform and affine) in different extent.

SIFT provides most accuracy and abundant key points but it requires lots of time to derive key points. SURF's accuracy and amount are not as good as SIFT provides, while it need much less time which is achieved by further approximate convolution with Gaussian which is used to get Hessian matrix by using box filters and a pre-computed integral image. Convolution is replace by addition. Besides, it uses a 64 element descriptor (using $\sum dx$, $\sum dy$, $\sum |dx|$, $\sum |dt|$ to replace 8 direction used in SIFT) in the same 4×4 sub-region as SIFT, shorter descriptors save time for matching.

Considering ORB, it uses fast corner detector by comparing grayscale in a local region. It is super fast that can perform in real time in some further application (for example ORB-SLAM2 used for camera trajectory estimation, for monocular camera, $30fps$ on TUM dataset, larger images like KITTI $10fps$, half of time is used to computing image features). But key point accuracy and number can not compare with SURF or SIFT (but is enough for general re-localization tasks like computing essential or homograph matrixes). It's a kind of bargain among speed and accuracy.

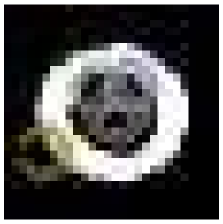
When using these feature points for matching. To avoid matching error, a threshold r_d is used to discard a matching pair when it's distance is larger than $r_d \times minDistance$ (without this restrict lots of error matches would occur $AppendicesB.1\ Figure15(a), 17(a, d)$). From $Figure7$ it can be seem that SIFT performs really brilliant, lots of points are correctly matched. Compare with matching results of ORB and SURF, SIFT ($AppendicesB.1\ Figure15.16.17$) provides more matching pairs and can accept a relative high r_d which indicates that all key points are distinct to each other, make it more robust and accuracy.

Robustness of SIFT can be proved from $Figure7(c, d)$ and $AppendicesB.1\ Figure15(d, e, f)$ where distortion, scale, brilliance and noise in $river2$ is changed. It can be found that SIFT performs well under these disturbance.

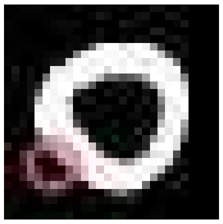
While ORB needs much smaller r_d to give accurate matching and far less matching point paris. This mainly comes from BRIEF descriptors it used. It is a 128 elements binary sequence which can be super fast to compute distance from each key point though using Hamming distance. However only $0/1$ alone would make it less representative compare with SIFT descriptors, resulting higher matching error (more sensitive to scale change, robust to rotation compare with SIFT).

With a good r_d all these methods result a good matching result comparing with with hand matched ground truth ($Figure7(a)$).

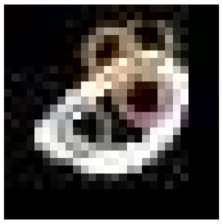
2.3 Bag of Words


(a)

(b)

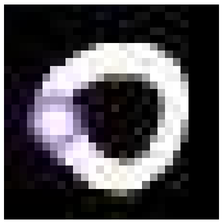

(c)

(d)

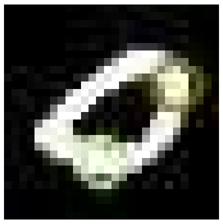

(e)

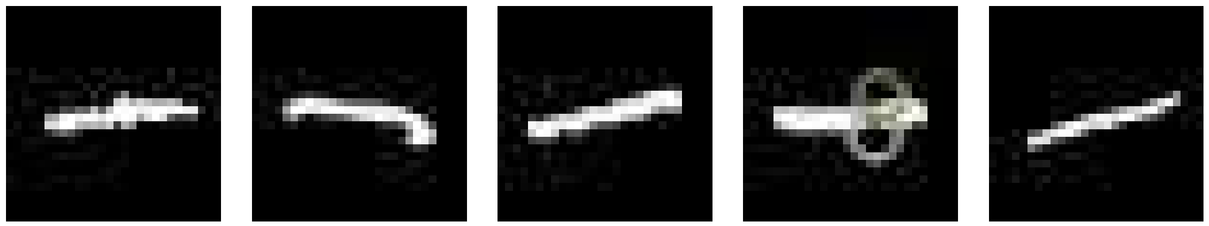
(f)

(g)


(h)

(i)


(j)


(k) (l) (m) (n) (o)

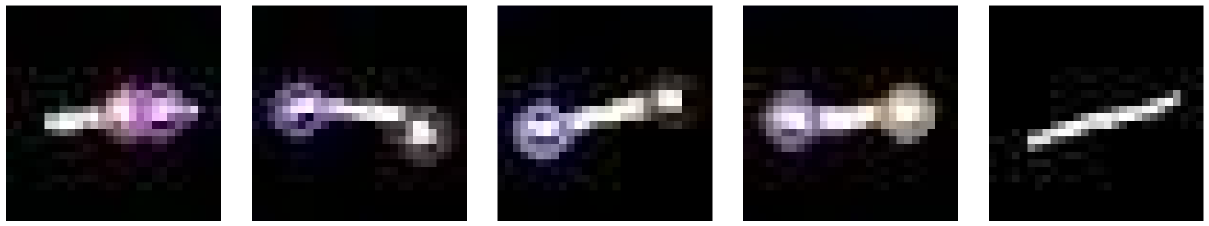

(p) (q) (r) (s) (t)

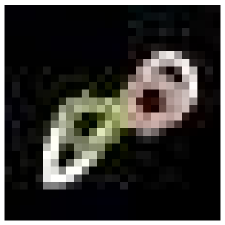
Figure 8: SIFT points contribute to different bin
$(a-d)$ and $(k-n)$ vote for $Bin\#0$
$(f-i)$ and $(p-t)$ vote for $Bin\#1$

Image	$Bin\#0$	$Bin\#1$	Image	$Bin\#0$	$Bin\#1$
$zero1$	**8**	5	$one1$	0	**2**
$zero2$	5	5	$one2$	0	**3**
$zero3$	1	1	$one3$	0	**3**
$zero4$	**7**	6	$one4$	2	**4**
$zero5$	**5**	2	$one5$	0	0

Table 4: Features' voting histogram

To use Bag of Words, SIFT features are derive from all train data ($AppendicesB.2$ $Figure18$). Then using K-Means to cluster all these features. This time K-Means from OpenCV is used (my implementation is based on $vector$, it would be in-efficient to convert Mat to $vector$ back and forth for K-Means). Two cluster centers become two words in dictionary.

Then training data is checked by this dictionary. Compare features with each word, choose one has lower L_2 distance with this feature to represent it's label. A voting histogram is derived after doing this process for all features in image. Histograms are shown in $Table4$ and features that vote for different labels are visualized in $Figure8$.


(a) 7 key points vote for $Bin\#0$

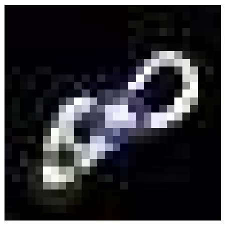

(b) 5 key points vote for $Bin\#1$

Figure 9: Apply on $eight.raw$

Than apply same test method on $eight.raw$ ($Figure9$). There are 7 key points vote for $Bin\#0$ and 5 key points vote for $Bin\#1$. In that case, it can be concluded that this eight image is more resemble to zero. Which can be regarded as eight is a combination of two zeros, in that case, key points would be much closer to these extracted from zeros.

References

[1] Hartigan J A, Wong M A. Algorithm AS 136: A K-Means Clustering Algorithm[J]. Journal of the Royal Statistical Society, 1979, 28(1):100-108.

[2] Kanungo T, Mount D M, Netanyahu N S, et al. An efficient k-means clustering algorithm: analysis and implementation[J]. IEEE Transactions on Pattern Analysis & Machine Intelligence, 2002, 24(7):881-892.

[3] Katsavounidis, I., Kuo, C.-C. J., & Zhang, Z. (1994). A new initialization technique for generalized Lloyd iteration. IEEE Signal Processing Letters, 1(10), 144–146.

[4] Lowe, David G. (1999). "Object recognition from local scale-invariant features" (PDF). Proceedings of the International Conference on Computer Vision. 2. pp. 1150–1157. doi:10.1109/ICCV.1999.790410

[5] `https://en.wikipedia.org/wiki/Scale-invariant_feature_transform`

[6] Lowe, David G. (2004).Distinctive Image Features from Scale-Invariant Keypoints. International Journal of Computer Vision. pp. 91–110

[7] `http://homepages.inf.ed.ac.uk/rbf/HIPR2/log.htm`

[8] Brown, M. and Lowe, D.G. 2002. Invariant features from interest point groups. In British Machine Vision Conference, Cardiff, Wales, pp. 656–665.

[9] `https://docs.opencv.org/3.4/d5/d3c/classcv_1_1xfeatures2d_1_1SIFT.html`

[10] `https://docs.opencv.org/3.2.0/d2/d29/classcv_1_1KeyPoint.html`

[11] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool, "Speeded Up Robust Features", ETH Zurich, Katholieke Universiteit Leuv

[12] Herbert Bay, Andreas Ess, Tinne Tuytelaars, Luc Van Gool "SURF: Speeded Up Robust Features", Computer Vision and Image Understanding (CVIU), Vol. 110, No. 3, pp. 346–359, 2008

[13] Rublee, Ethan; Rabaud, Vincent; Konolige, Kurt; Bradski, Gary (2011). "ORB: an efficient alternative to SIFT or SURF" (PDF). IEEE International Conference on Computer Vision (ICCV).

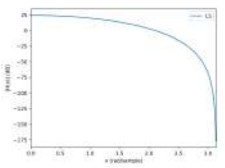
[14] `https://en.wikipedia.org/wiki/Bag-of-words_model_in_computer_vision`

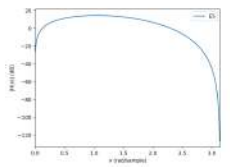
[15] `http://people.csail.mit.edu/fergus/iccv2005/bagwords.html`

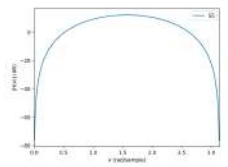
Appendices

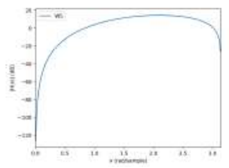
A Object Analysis

A.1 Texture Classification


(a) $L5$


(b) $E5$


(c) $S5$


(d) $W5$

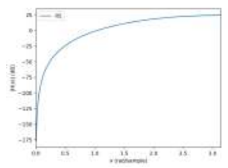

(e) $R5$

Figure 10: Filter Response

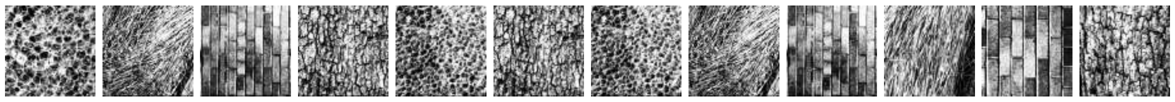


Figure 11: Histogram Equalization

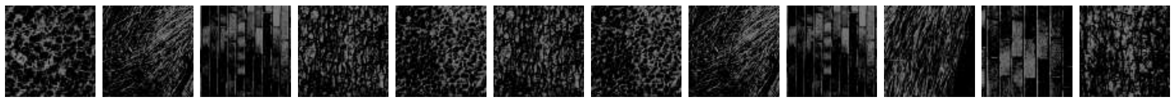


Figure 12: Subtract Mean

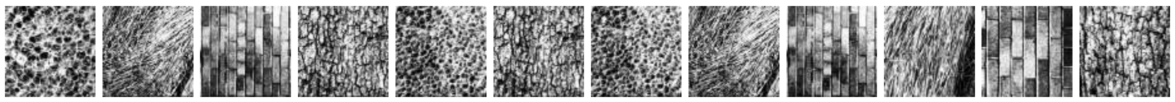
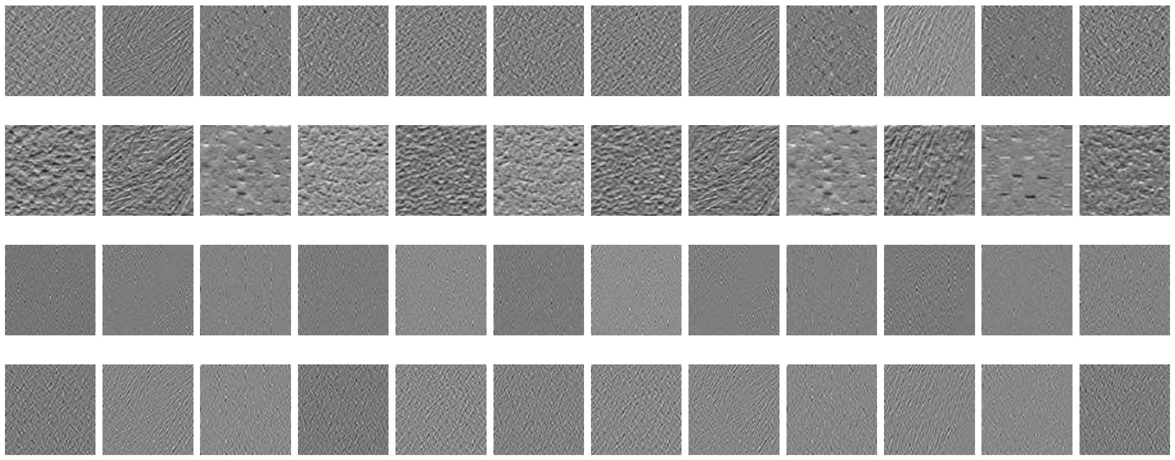
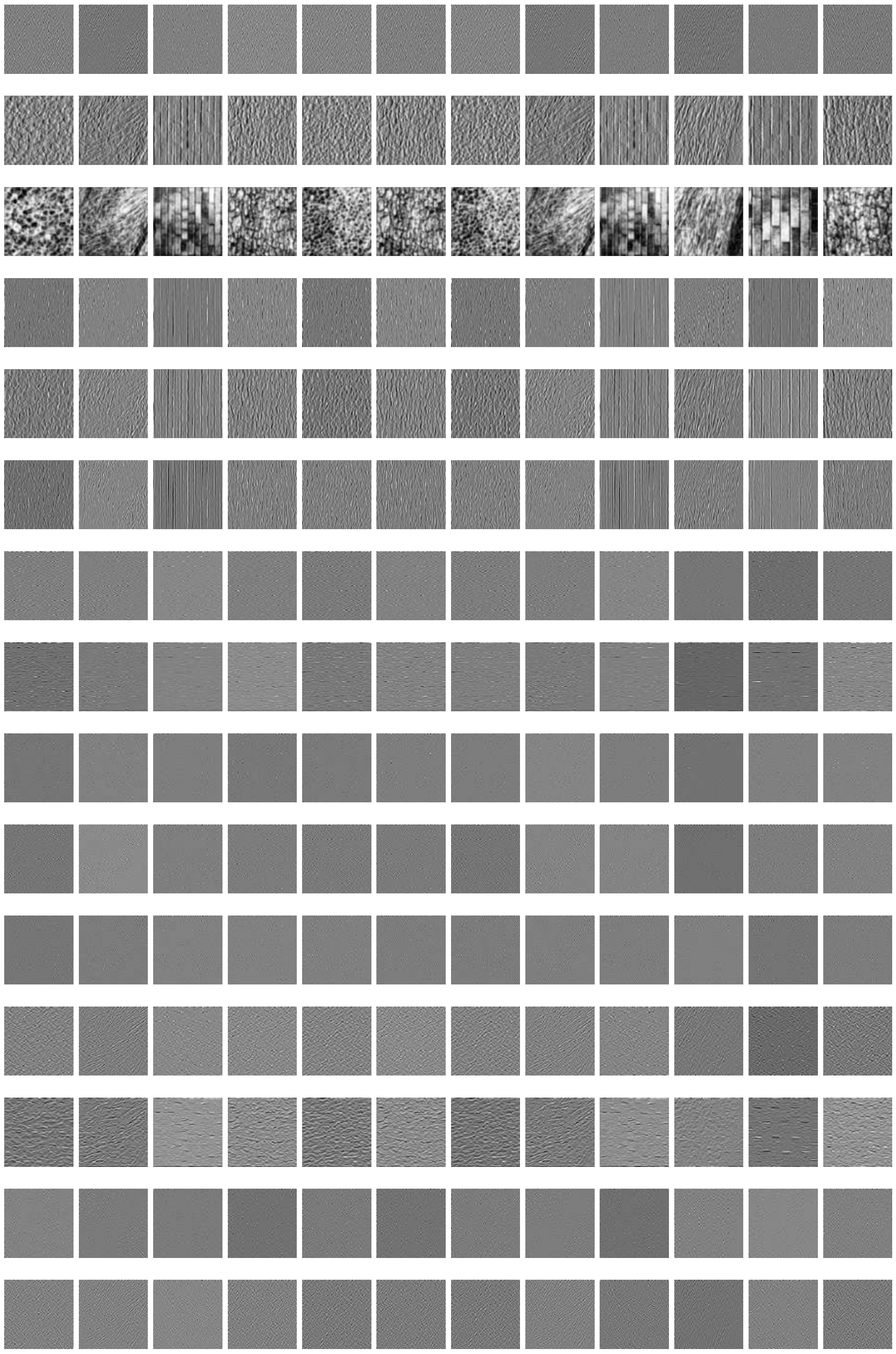


Figure 13: Subtract Mean (Normalized)





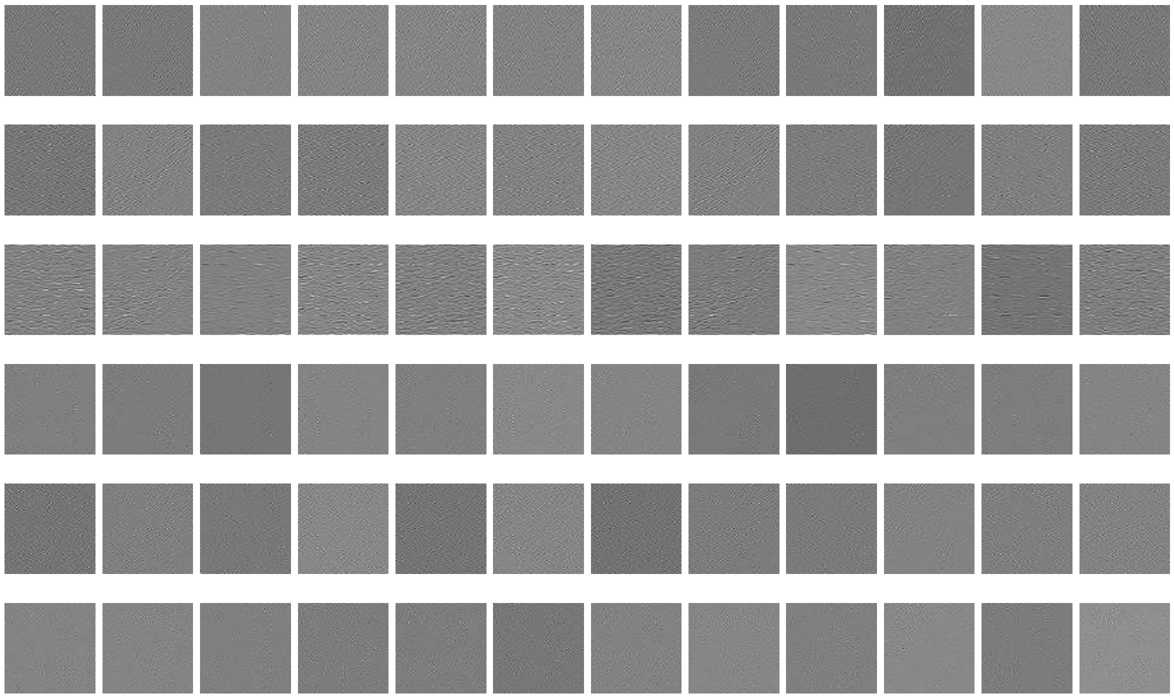
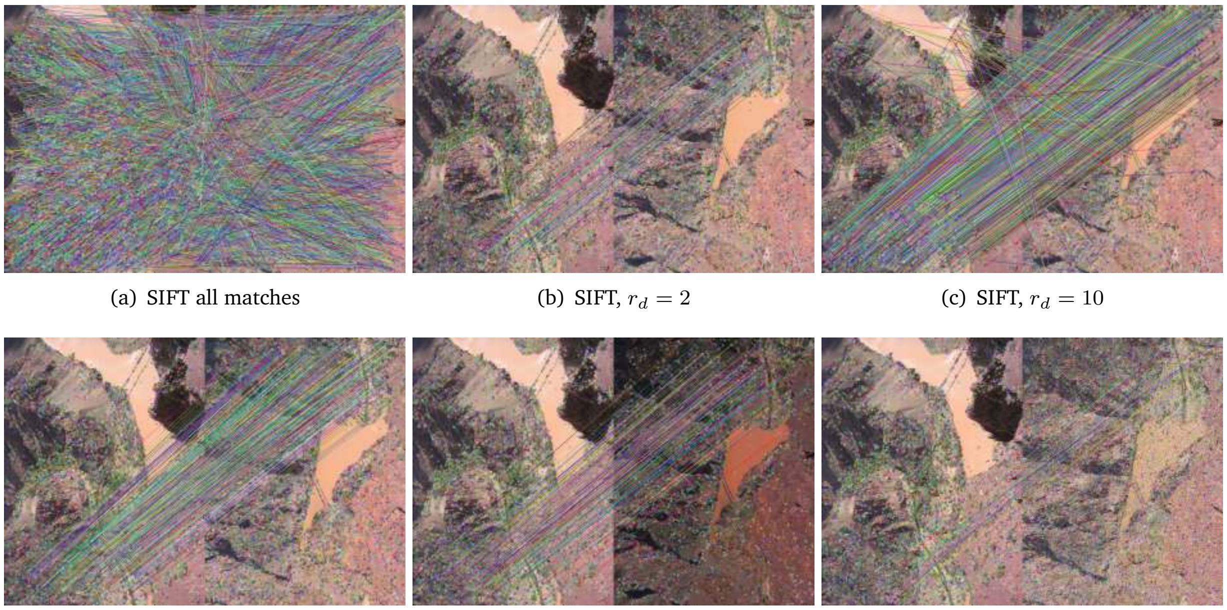


Figure 14: $texture1 - 12$'s features (colimn)
$ELRSW * ELRSW$ (row)

B Image Feature Extractor

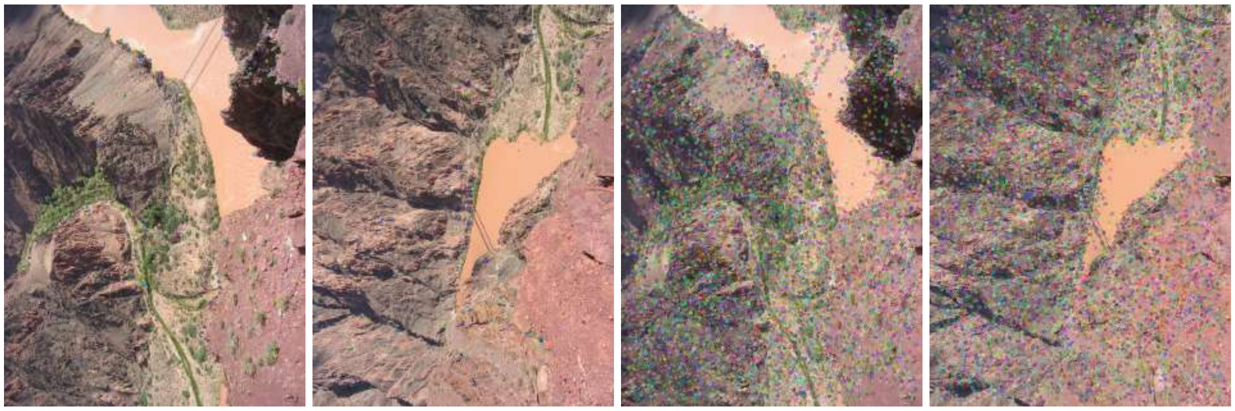
B.1 SIFT



(a) SIFT all matches (b) SIFT, $r_d = 2$ (c) SIFT, $r_d = 10$

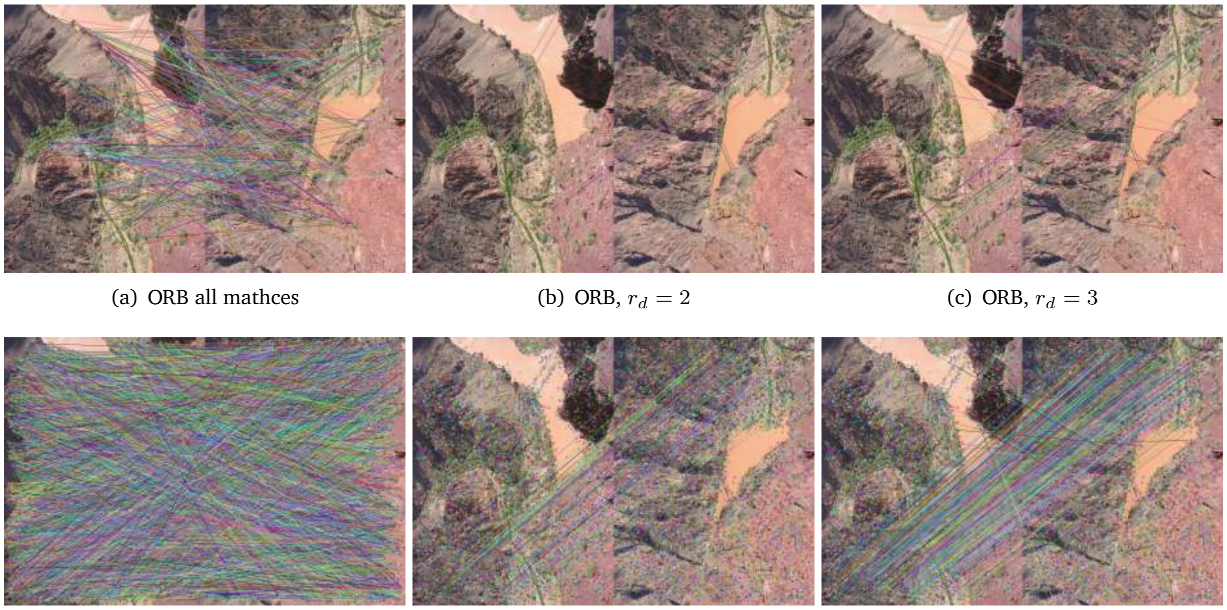
(d) Distortion (e) Brilliance Change (f) Gaussian Noise $\sigma = 0.1$

Figure 15: SIFT



(a) river1 ORB (b) river2 ORB (c) river1 SURF (d) river2 SURF

Figure 16: ORB and SURF on river image

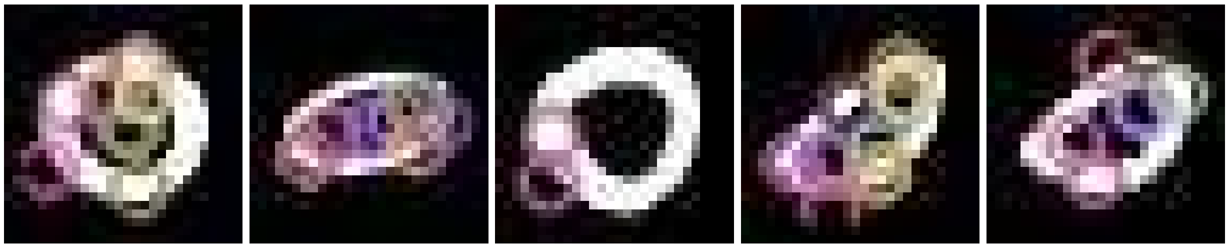


(a) ORB all mathces (b) ORB, $r_d = 2$ (c) ORB, $r_d = 3$

(d) SURF all matches (e) SURF, $r_d = 2$ (f) SURF, $r_d = 3$

Figure 17: ORB and SURF good match

B.2 Bag of Words



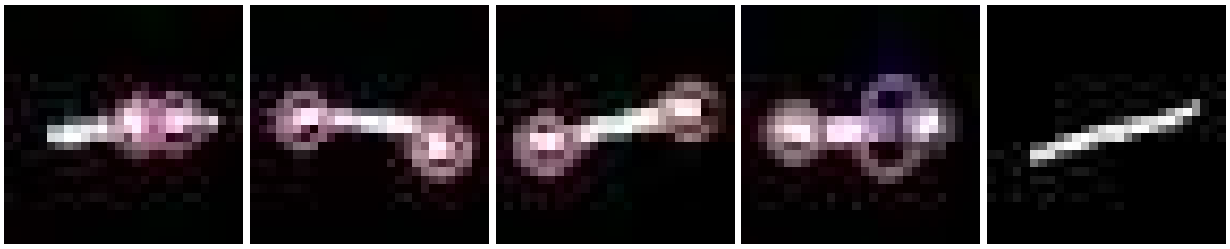


Figure 18: SIFT on numbers