Homework 4 for 20 spring EE569
Texture Analysis & Segmentation, Image Feature Extractors
Chengyao Wang
69615991816
chengyao@usc.edu

Problem 1, Texture Analysis:

1. Motivation & Approaches:

Overall Description:

Texture analysis has long been a topic in computer vision as being a major part of "Image analysis", especially "object recognition". It is primarily concerned with the unique "patterns" of different objects and tries to perform other downstream tasks, e.g. object recognition, information extraction from remote sensing snapshots and surface defect detection. To the best of our knowledge, there is no formal definition of ***textures***. Our naïve attempt may be stated as: recurrent, yet noisy patterns of an object surface. Surface of interest contains certain repetitive patterns which are not exactly the same. These patterns may vary a lot between different objects in size ratios, colors and spacial distributions. A typical programmable texture analysis pipeline could be presented as:

$$(pre-processing) -> Feature\ Extraction -> Decision\ Maker$$

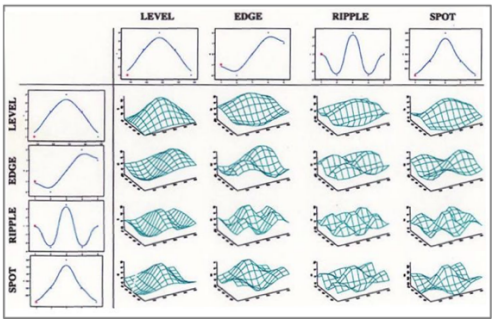
"Decision makers" are responsible for taking the "information" provided by downstream algorithms and perform what the task asks for. They are often machine learning (classification) algorithms such as K-means, Random Forest & SVM, which are implemented in this assignment.
Nowadays, modern deep learning models provides end-to-end solutions to those tasks and generally have better performance. But they are not part of this assignment.

Feature Extraction:

Feature extraction extracts information from input images and often organize them into vectors for each instance. Pixels & images itself could all be instances, depending on the tasks we're facing. There are many ways to perform feature extraction and we are adopting the method introduced by Laws[1] in his PhD thesis. The approach is very similar to what we do in edge detection & denoising, where we pre-determine a "filter" or "kernel" to swipe across the image and do linear convolution operations. The *filters* we use carries the numerical patterns we are interested and often can also be interpreted as 2D digital filters in digital signal processing. They are defined in 1D form and can be used to construct 2D *filters* using tensor product. Notice that for vectors, tensor product is equivalent to outer product. There are 5 of them and they take the name ***Laws filters***.

$$L5(level) = [1\ 4\ 6\ 4\ 1]$$
$$E5(edge) = [-1\ -2\ 0\ 2\ 1]$$
$$S5(spot) = [-1\ 0\ 2\ 0\ -1]$$
$$W5(Wave) = [-1\ 2\ 0\ -2\ 1]$$
$$R5(Ripple) = [1\ -4\ 6\ -4\ 1]$$

There are obviously 25 2D-filters as a result of pairwise tensor product and since it's a trivial process, they are not presented here. From the nature "measuring similarity" of inner product, we could try to match the name *Laws* gave to each filter to what pattern it's seeking. A continuous version of such is presented in the discussion[2].



Now we convolve each with one single image (using certain padding tricks) and stack the response mapping up:

$$Image_{MxN} * Kernal_{25} = Response_{MxNx25}$$

They are the direct results of feature extraction. Then we could synthesize / combine the per-image-per-pixel responses to construct dataset for the downstream tasks which may include energy spatial averaging, PCA dimensionality reduction or handcraft significance analysis & feature selection.

[1] "Textured Image Segmentation", Kenneth Ivan Laws, Jan 1980.
[2] "EE 569 Discussion 8", Min Zhang, Mar 2020

For texture classification, we are focusing on single images thus we average all MxN vectors of that image to construct feature vector. For texture segmentation, every pixel needs to be classified thus we adopts a window-approach to assign feature vectors to every pixel by averaging over its spatial neighbors.

Discriminant power analysis:

For discriminant power analysis, ***one simple yet valid approach would be determining the empirical variance of the features in unsupervised manner.*** The main idea comes from *PCA*, where we also take variance of the features as information and try to perceive it to the most. Larger the variance, more discriminant power it possesses. For one certain feature:

$$\sigma_i{}^2 \propto \sum_i (x_{ij} - \mu_j)^2$$

Another better-founded approach is using Fisher Linear Discriminant analysis (LDA)[3]. It assumes identical covariance matrix for every class, and the scatter between class variability may be define by:

$$\Sigma_b = \frac{1}{C}\sum_{i=1}^{C}(\mu_i - \mu)(\mu_i - \mu)^T$$

Where μ_i is the mean of the class-means. The class separation along a certain direction $\vec{w}$ could be derived as:

$$S = \frac{\vec{w}^T \Sigma_b \vec{w}}{\vec{w}^T \Sigma \vec{w}}$$

We could estimate Σ by:

$$\Sigma = (X - E[X])^T (X - E[X])$$

Then we could input any query axis $\vec{w}$ to get the discriminant power along that axis as wish. But here since we are interested in raw features, $\vec{w}$ are all one-hot vectors and we can use mathematical tricks to efficiently compute Ss:

$$S_j = \frac{\frac{1}{C}\sum_{i=1}^{C}(\mu_{ij} - \mu)^2}{\sum_{i=1}^{36}(x_{ij} - \mu_{ij})^2}$$

Both of them are implemented and results are shown in the following sections. Identical variance of classes is an assumption of *Fisher LDA*, which is quite strong. Here we have no choice but to violate with a high probability. And also worth pointing out, this process should also be limited to training set.

Machine learning Algorithms:

This assignment is mostly concerned with applications of machine learning models. Their detailed mechanics / properties & tuning process is not discussed here. ***We implemented K-Means Clustering & PCA.***

K-Means Clustering & K-Means++:

K-Means Clustering is an unsupervised machine learning algorithm trying to cluster the data into different clusters. We could interpret the clustering process as "gathering similar instances". *K-Means Clustering* is an unsupervised variant of *Expectation Maximization (EM)*, thus also suffers from convergence problems. ***Two simple way to alleviate such drawbacks are: K-Means++ & restarts. They can be combined, and both of them are implemented here.***

K-Means++[4]:

1. Randomly select the first centroid from the data points.
2. For each data point compute its distance from the nearest, previously chosen centroid.
3. Select the next centroid from the data points such that the probability of choosing a point as centroid is directly proportional to its distance from the nearest, previously chosen centroid. (i.e. the point having maximum distance from the nearest centroid is most likely to be selected next as a centroid)
4. Repeat steps 2 and 3 until k centroids have been sampled

Restart[5]:

Use different pseudo random seeds for random number generation and repeat the process multiple times. We adopt the ***inertia / in-cluster-variance*** metric for measuring performance.

---

[3] https://en.wikipedia.org/wiki/Linear_discriminant_analysis
[4] https://www.geeksforgeeks.org/ml-k-means-algorithm/
[5] https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html

$$Inertia = \sum_{i \in \{C\}} \sum_{x_j \in C_i} \|x_j - \mu_i\|_2^2$$

Obviously, ***inertia*** is measuring the compactness of the clusters thus smaller the better. In implementation after a pre-specified number of restarts (default 20), we adopt the result that has the smallest ***inertia***. But worth noticing, it's not such a strong metric as opposed to accuracy, precision, recall etc. in supervised cases. It is not a universally applicable and convincing metric.

Support Vector Machines / Kernel Machines[6] & Random Forest[7]:

Random forests (or in OpenCV terminology Random Trees) are already discussed in assignment 2 thus is not repeated here.
The theoretical core of *Support Vector Machines* lies in constrained convex optimizations and Karush–Kuhn–Tucker (KKT) conditions. Now with the era of machine learning, *linear support vector classifier (SVC)* also gets a name for its loss function – *hinge loss*, and is believed now to be approachable using an iterative method (possibly gradient descent) just like *logistic regression (LR).* It shares the same overall objective as *logistic regression* - finding the best hyperplane to separate different classes. Its dual form also makes it possible to extend to *kernel methods* which in many cases boosts the model performance by a considerable amount. *Kernels* are beyond the scope of this course and the approach's feasibility is supported by the *Representor theorem*[8].

Principle Component Analysis (PCA):

PCA is a dimensionality reduction algorithm and can be stated as: empirical variance decomposition. It uses an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables called principal components.[9] ***It takes variance as a measurement of information and tries to preserve the most variance given certain number of principal components***. Here are some comments over *PCA*:

1. *PCA* naturally outputs uncorrelated features, which is resulted from the orthogonality between eigenvectors of symmetric matrices, which is good for downstream machine learning algorithms.
2. PCA could easily filter out redundant features, BUT NOT UNRELAVANT FEATURES. If some features could be expressed or nearly as a linear combination of others, *PCA* will put them at lowest priority of selection.
3. *PCA* is extremely sensitive to the scale of the raw features. Thus, standardization is often suggested as a previous procedure of *PCA*.
4. ***PCA* welcomes mean-subtraction because the principle axis all goes pass the origin. We do another mean subtraction w.r.t features before sending data matrix to PCA module.**

To perform PCA, we could either solve for standard eigenvalue / eigenvector pairs for $X^T X$, or do SVD on X which are equivalent. Here we adopted the latter.

Some Implementation Details:

1. We adopted a procedure of doing **classification** theoretically equivalent as suggested, which is:
   a. Precompute image mean and perform feature extraction by swiping *Lows Filters* over the image.
   b. Subtract mean on the feature extracted from $L5L5$, weight on the mean should be the coefficient sum 256. This is equivalent to mean pre-subtraction because all other filters have zero mean. It will result in substantially better time complexity, especially in segmentation applications.
   c. Calculate energy (taking absolute values), do $25D -> 15D$ transforms as suggested and average every pixel's 15D feature vector across the whole image. Stack all the training image up and form the data matrix with standard layout in machine learning.
   $$X_{36x15} = \begin{bmatrix} x_1^T \\ x_2^T \\ \vdots \\ x_{36}^T \end{bmatrix}$$
   d. Perform some kind of normalization. One, as suggested, for each feature vector divide all its feature by their own $L5L5$. Other attempts are stated later.
   e. (Optional) Do *PCA*. Notice here we are allowed to access training set only, thus dimensionality reduction on the test set should be done by applying transformation derived from the train set.
   f. Feed into the machine learning algorithms and get predictions.
2. The overall procedure for **segmentation** is generally the same, but:
   a. We are averaging over the predefined local window rather than the whole image. As a result, we could perform the identical feature extraction part as (1). But now, the scope we're averaging & mean calculation are all constrained within the window. From another perspective, in **classification**, the window size is 128 by 128 and the center pixel is a representative of the whole image. In **segmentation**, each pixel represents itself.
   b. The rest is identical to **classification** where we are using *K-Means* to cluster the 270000 data points into 6 clusters. The data matrix after rearranging should be:
   $$X_{270000x15}$$
3. **We adopted odd reflection for boundary extension**. Zero padding is tried in the first place but will create such boundaries in **segmentation**. This is merely for visual purposes and no convincing theoretical reasons. (Notice the boundaries)

---

[6] https://docs.opencv.org/3.4.9/d1/d2d/classcv_1_1ml_1_1SVM.html
[7] https://docs.opencv.org/3.4/d0/d65/classcv_1_1ml_1_1RTrees.html
[8] https://en.wikipedia.org/wiki/Representer_theorem
[9] https://en.wikipedia.org/wiki/Principal_component_analysis

4. Relating to edge detection applications, we could find proper motivation averaging energy responses. We are *averaging* the responses along the x-axis & y-axis. Before we took *root mean*, but this time *arithmetic mean*.

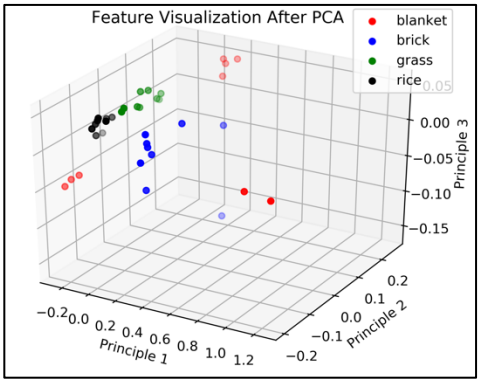
$$\frac{n}{\frac{1}{x_1}+\frac{1}{x_2}+\ldots\ldots+\frac{1}{x_n}} \leq \sqrt[n]{x_1+x_2+\ldots\ldots+x_n} \leq \frac{x_1+x_2+\ldots\ldots+x_n}{n} \leq \sqrt{\frac{x_1^2+x_2^2+\ldots\ldots+x_n^2}{n}}$$

5. ***We in total introduced four normalization methods mentioned in 1(d) for Q1(d).*** They are:
   a. For each instance, divide every feature by $L5L5$.
   b. Manually set $L5L5$ to 1, while keeping other intact.
   c. Standard normalization techniques in machine learning – normalizing *L2-norm* & zero mean.
   d. Do nothing.

   The assignment asks for direct & improved implement of the algorithms. Here, (a) corresponds to the unmodified approach suggested by the requirements while others are attempts to improve. We are not deliberately distinguishing between four of them in sections below. Experiments indicates that (b) is the best.
6. ***To make K-Means applicable to supervised learning scenarios, we could do perform majority vote within each cluster and assign cluster the most frequent label. If any test instance falls in this cluster, the prediction will be that label.*** Of course, this primal way will cause many problems, like cases where the model never raises certain class predictions which does happen as shown later. ***Notice, we should distinguish between cluster index & cluster label.***
7. SVM & RF all came from OpenCV library. There is no explicit hyperparameter tuning process since it's not the main focus of this assignment. **But we've changed linear kernels to Radial Basis Function (RBF) kernels for a better performance.**
8. ***PCA are self-implemented, but the core part of performing SVD came from Eigen3 library[10].*** In *SVD*, the right singular matrix is where our transformation lies. By stacking the first certain number of singular vectors, we could after on perform transformation simply by matrix multiplication.
9. ***Another trial made in improving segmentation results is use global mean / window mean***. We've noticed K-Mean have a hard time trading-off the performance in *bricks* region & *others*. The primary characteristic that distinguishes *bricks* from others would be brightness / luminance. Thus using *uniform mean* across all windows may help.
10. ***PCA are a third attempt in improving segmentation performance, by reducing the dimension of features from 15 to 3.***
11. ***K-Means++ is always on. We are not using vanilla random initialization in this assignment.***
12. Resources are limited to perform exclusive model parameter tuning. **Thus we are just varying the setting to see if it leads to any improvements. We are not jointly fine tuning them.** The dataset K-Means clustering is facing has 270000 instances and we are restarting 20 times. This consumes huge amount of time if multi-threading is not introduced. The number of iterations though, is lower than expected with most cases converges within 120 iterations. **In addition, we are not playing too much with 15D features again because of the exploding time complexity.**
13. We are suggested to do post-processing steps to improve the segmentation results & boundary enhancement. But, firstly due to time constraint as well as not **introducing inductive bias & human interference as much as possible**, we are not implementing the two.

2. Results & Analysis:

Feature Visualization After PCA[11]:



[10] http://eigen.tuxfamily.org/index.php?title=Main_Page

[11] Visualization using Python, by reading the csv file written by C++. Related library used: *Matplotlib.pyplot & mpl_toolkits.mplot3d*.

Discriminant power analysis (Classification):

We abbreviate unsupervised method by "*var-based*", and fisher LDA as "*Fisher*". The ranking part is done by hand. **The normalization method is (a), thus we will see L5L5 having zero predictive power.**

Absolute Values

	L5L5	L5E5	L5S5	L5W5	L5R5
Var-based	0	0.00536203	0.00473214	0.00651995	0.0225076
Fisher	0	0.000455515	0.0226535	0.0259036	0.121058
	E5E5	**E5S5**	**E5W5**	**E5R5**	**S5S5**
Var-based	0.000611723	0.000502963	0.00102923	0.00350123	0.000800455
Fisher	0.00340263	0.0015905	0.00384249	0.0104297	0.0173187
	S5W5	**S5R5**	**W5W5**	**W5R5**	**R5R5**
Var-based	0.00185721	0.00589856	0.0042715	0.0127936	0.0375618
Fisher	0.0372325	0.0170737	0.0854632	0.242781	1.10253

Relative Values (Rankings: Var-based/Fisher):

	L5L5	L5E5(6/14)	L5S5(7/7)	L5W5(4/6)	L5R5(2/3)
Var-based	0	0.0496714	0.0438364	0.0603979	0.2085
Fisher	0	0.00026926	0.0133907	0.0153119	0.0715584
	E5E5(13/12)	**E5S5(14/13)**	**E5W5(11/11)**	**E5R5(9/10)**	**S5S5(12/8)**
Var-based	0.00566673	0.00465923	0.00953436	0.0324338	0.00741506
Fisher	0.00201133	0.000940162	0.00227134	0.00616511	0.0102373
	S5W5(10/5)	**S5R5(5/9)**	**W5W5(8/4)**	**W5R5(3/2)**	**R5R5(1/1)**
Var-based	0.0172044	0.0546417	0.0395692	0.118514	0.347956
Fisher	0.0220086	0.0100925	0.0505182	0.14351	0.651715

We can see that the two methods are almost the same in most cases, but there also exist huge disagreements on some features. One possible reason may be *Fisher LDA* is a supervised method, thus has more information.

Singular Value Decomposition (SVD) Results (Classification):

Normalization Type	(a)	(b)	(c)	(d)
1	1.80028	4245.29	0.171093	12887.8
2	0.745901	2747.46	0.0656309	4224.41
3	0.270873	1418.28	0.0277065	2598.82
4	0.0935497	498.731	0.00899964	1338.93
5	0.0735953	300.451	0.00603023	430.247
6	0.0239415	94.3935	0.00354736	298.95
7	0.0219296	75.2269	0.00291644	92.1258
8	0.0140045	54.9825	0.00184116	74.4497
9	0.00606674	23.2577	0.001342	54.9752
10	0.00275678	9.22792	0.000790228	22.6555
11	0.0022154	5.53742	0.000682429	9.12269
12	0.000957129	4.12432	0.000449263	5.53271
13	0.000609394	3.1624	0.000183307	3.99569
14	0.000120211	0.649022	7.465e-05	2.9947
15	0	0	6.08026e-05	0.604791

The first 3 principle component retrieves *information* most efficiently. Thus, 3 is a good choice.

Texture Classification:

Label encoding chart: blanket = 0, brick = 1, grass = 2, rice = 3.

Predictions (& Accuracy)

True	2 0 0 1 3 2 1 3 3 1 0 2			
	Normalization(a)	**Normalization(b)**	**Normalization(c)**	**Normalization(d)**
K-Means (3D)	2 0 3 2 2 2 2 2 2 3 0 2 (5/12)	2 2 0 0 3 2 0 3 3 3 2 2 (7/12)	2 2 3 1 3 2 3 3 3 3 0 2	~~3 0 2 1 0 0 2 0 0 3 0 0~~
K-Means (15D)	2 0 3 2 2 2 2 2 2 3 0 2 (5/12)	2 2 0 0 3 2 0 3 3 3 2 2 (7/12)	2 2 3 1 3 2 1 3 3 3 0 2	~~0 1 3 1 0 1 3 0 0 0 0 0~~
Kernel Machine	2 0 3 1 2 2 1 2 2 0 0 2 (7/12)	2 0 0 1 3 2 1 3 3 1 1 2 (11/12)	2 3 3 1 3 2 3 3 3 3 2 2	~~1 1 1 1 1 1 1 1 1 1 1 1~~
RF	3 0 3 1 2 0 3 2 2 3 1 2 (8/12)	2 0 0 1 3 2 1 3 3 1 1 2 (11/12)	~~0 1 3 3 3 0 1 3 3 1 0 0~~	2 0 0 1 1 2 3 1 1 0 0 2

K-means Clustering Result (Restart = 20, 3D feature)

Normal(a)	Class0	Class1	Class2	Class3	Cluster Label	Normal(b)	Class0	Class1	Class2	Class3	Cluster Label
Cluster 0	3	0	0	9	3	Cluster 0	4	2	7	0	2
Cluster 1	0	7	9	0	2	Cluster 1	4	4	0	0	0
Cluster 2	2	0	0	0	0	Cluster 2	1	0	0	0	0
Cluster 3	4	2	0	0	0	Cluster 3	0	3	2	9	3

Four of the predictions are elided because they made too many mistakes, may resulted from *floating point precision* or the extreme magnitude of the *L5L5*. We focus more on method (a) & (b). Here are some observations & comments:

1. Most of the *K-means* trials terminates within 10 iterations.
2. Among the restarts, the smallest *inertia* appears considerable amount of times (~30%). But that doesn't necessarily mean they converges to the same clusters due to precision of *floating-point numbers.*
3. **K-Means has a worse performance than supervised methods, which is expected. Labels preserve more information than we normally expected.**
4. With regard to the three machine learning algorithms, apart from the proceeding point, it's also worth noticing that their selves are actually not on the same levels in terms of model complexity or capacity. The corresponding Hilbert Space defined by **RBF** kernel is known to be infinite dimensional, providing the model with excessive power w.r.t linear SVM. On the other hand, Random Forests are also known to be scalable to large complexity whether in terms of forest width or tree depth. K-Means with L2-distance are linear models. Performance differences will be larger in more complicated tasks that violate multi-gaussian model assumption of K-Means.
5. **Majority vote has obvious drawbacks and there are multiple better ways, but all of them are quite time consuming**. One attempt should be turning cluster assignment from hard to soft. That leaves possibility to generate "1" prediction. Notice that we are not interested in manual assigning labels to clusters, like for left results "3201".
6. ***The results from 15D feature & 3D feature are the same under K-Means from accuracy's perspective. In other words, PCA proved itself to be successful because 15D->3D is a substantial time complexity improvement. Also high dimension statistics is a huge topic which will bring about numerous mathematical issues. We should avoid them as much as possible.***

Texture Segmentation:

Results are put in the appendix.

a. The results of the standard way & default parameter can perform some kind of segmentation, but the results cannot be taken as good neither. Problems lies in:
    a. Noises / isolated areas, for sure. Mainly occurs then the window size is small, because small window size is vulnerable to variances.
    b. Boundary issues. Pixels near the boundary are crossing the features of both textures, and the final results will be uncontrollable & predictable because we have no idea about the relative location & size of the clusters. **It is not necessarily guaranteed to be assigned one the two textures.**
    c. Extreme bad performance in lower right "*bricks*".

   Actually, (c) is mostly resulted from mean subtraction. **Mean subtraction is introduced mainly to combat luminance change, but here, we actually need this information because "*bricks*" area is substantially brighter than others. This is where implementation of "*global mean*" originated.**

b. **The change in window sizes behave as expected**. On one hand, small window size is vulnerable to variances and gathers much less global information. If too big, the texture classes which cluster is corresponding to will be clustered (the clusters will be clustered) and reversely make the model again variance sensitive - A small step further will end up in another cluster. And also, performance around corners will behave badly, as shown in the results.

c. Likewise in classification, ***PCA* again helps in lowering the time complexity while retaining the general results**. There may be slight unobservable differences.

d. **Non-zero padding successfully eliminates the abnormal boundary prediction presented above.** The thought behind this is that the weights will not be normalized because we're swiping the same kernel across the whole image, thus resulting in unexpected magnitude of the *filter* responses. We could accept mirror segmentation around the boundary, but not some other label popping up.

e. **The use of global mean improves the performance in the brick area, while not worsening the other too much**. The thought behind this is explained above. But around the star, the prediction still goes to other labels different from it's both sides.

f. **The use of Normalization (b) comes from its contribution to the improvement in classification**. And approach (iii) & (iv) is the best method so far.
    a. Aside from the boundaries, predictions are mostly consistent throughout the areas.
    b. The star retained its shape, though when a rounder corner.
    c. **Small prediction mistakes can be postprocessed away using median filters**.

   But despite the fact that global mean helps "*bricks*" area getting a rather uniform prediction throughout the area, the boundaries are still eroded by the neighboring textures. To the best of our attempts, there is no valid way to address this problem without introducing prior knowledge.

g. Catching the preceding point, we didn't introduce any tricks that involve prior knowledge such as boundary locations and spatial information integrated post-processing techniques. One example of the latter is location-dependent *filters*. We could implement invariant median *filters* like aforementioned, but due to time constraint is left for future study.

h. The choice "*global mean / window mean*" doesn't matter if we adopt *Normalization (b)*. Like said in ***Implementation Details (1)***, *L5L5* is the only feature that will be affected by mean subtraction. In *Normalization (b),* we manually assigned 1 to it, thus *improved approach (iii) & (iv)* are exactly the same.

Problem 2, Image Feature Extractors:

1. Motivation & Approaches:

Scale-Invariant Feature transform (SIFT)[12]:

We will first answer the questions asked by the requirements.

1. Affine transformations, including scaling, rotation and shifting (location, scale and rotation).
2. SIFT achieves robustness to the above issues by:
    a. Scaling: Introduction of scale space. It mainly serves for two purposes: efficiently provide features for blob detection & provide scale invariance. **Specifically, performing blob detection in multiple *octaves* and direct parallel comparison between different scales is the key**. Initial (original scale) *octave* includes 5 gaussian blurred images and the next *octave* is generated by subsampling directly from the previous *octave* via taking every second pixel in rows and columns of 5 images, according to the original paper.
    Intuitively speaking, if two images have one feature at different scales, that feature would be extracted from a lower scale octave where the feature is small in scale and a higher octave where the feature is large.
    b. Rotation: **By determining & assigning each detected feature orientation and at the point for feature vector generation, orientation is taken into consideration.** Within the neighborhood of every localized feature point, we can compute gradient orientation using filters like Sobel for each pixel and it's further quantized into every 10 degrees, i.e. 0, 10, ……, 340, 350. We could build a histogram of the orientations of all the pixels traversed and assign the most frequent orientation to this feature. Notice that, there may be cases where some other orientations that have a frequency larger than 80% of the dominant one. It will create another feature for each of this orientation.
    c. Shifting: Spatial information are naturally removed at the very beginning, because blob detection doesn't encode it. All the final output features are compared or used in parallel.
3. By *thresholding + normalization*:
    a. There may be linear & non-linear illumination changes in the image. The former is what we often refer to, and the latter is often resulted from camera saturation or 3D geometric locations & orientations by original paper.
    b. After we get the feature vectors, we first threshold every element in the feature vector to be no larger than 0.2, and then perform normalization.
    c. Thresholding -> non-linear, normalization -> linear.
4. Better time efficiency, and considerably good approximation.
5. 128 per feature.
    a. Each feature vector is a histogram formed from the gradient of the grayscale image. As stated by the original paper, the size 4 by 4 of the spatial bins is determined by measuring the performance. Each spatial bin contains an angle histogram divided into 8.
    b. In total: 4 * 4 * 8 = 128.

Bag of Visual Words (BoVW):

BoVW is another way of utilizing the feature extracted from *SIFT* other than the averaging method implemented above to perform tasks like image classification where we take images as instances. **They both dedicates to provide *SIFT* feature vector synthesis and image representation generation (using one vector with acceptable size to represent the image).** *BoVW* originated from *BoW* in NLP where it's used to generate one vector to represent one text. They both discards the order of the words & spatial information respectively. The steps are as follows:

1. We perform feature extractors (SIFT, SURF, etc) to extract features from all images in the dataset and form an ensemble of "*vocabularies*".
2. We cluster the features in the *vocabularies* using K-Means clustering and gets *K* centroids.
3. For every query image, we also extract features using the same feature extractor. We use K-means model to assign all the features to the clusters and we get a frequency distribution of the number of features in each cluster.
4. This will be the raw image representation w.r.t the dataset. And when performing image classification, we could introduce metric functions for distributions like KL, Chi-square etc. to get classification results.

Some Implementation Details:

1. *SIFT & FLANN (Fast library for approximate nearest neighbor)* is called from OpenCV library[13]. Also OpenCV provides various functions to perform *NN* search. Some are direct matches, while others provide noise / false detection & elimination[14]. We are only using direct matches here. ***SIFT* itself also provides feature filtering by retain the best in terms of contrast upon object instantiation by specifying the exact number to keep. We only unable the functionalities upon detection result presentation of single images & BoW. While matching features, we limit the number to 20 to keep the visualization from being too messy.**

2. *SIFT* features generated by OpenCV library are quantized into 0 ~ 128 integer values. Mentioned above, there are *thresholding + normalization* done to combat illumination change, thus in theory, all elements in the feature vector will be [0, 1]. For memory efficiency & time complexity, we are quantizing the values to *0 ~ 127 unsigned char*.

3. **Features with the largest scale is not equivalent to the size of the features, in other words what octave it's detected.** OpenCV defined a class for the key points, named cv::KeyPoints which has attributes[15]: *angle, class_id (for object detection), octave, location, response and size*. Here, *response* refers to how strong the key point is thus is what we are looking for. **We use it to choose the largest scale.**

---

[12] "Distinctive Image Features from Scale-Invariant Keypoints" DAVID G. LOWE, 2004
[13] https://docs.opencv.org/3.4/d5/d3c/classcv_1_1xfeatures2d_1_1SIFT.html
[14] https://docs.opencv.org/3.4/d5/d6f/tutorial_feature_flann_matcher.html
[15] https://docs.opencv.org/3.4/d2/d29/classcv_1_1KeyPoint.html

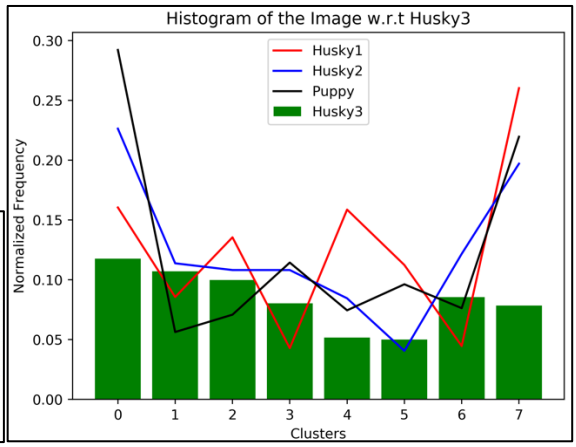
4. **When comparing different distances between histograms, we normalize the frequency distributions (histograms of clusters) by the total number of features, and present L2-distance.**

2. Results & Analysis:

Image results will be put in the appendix.

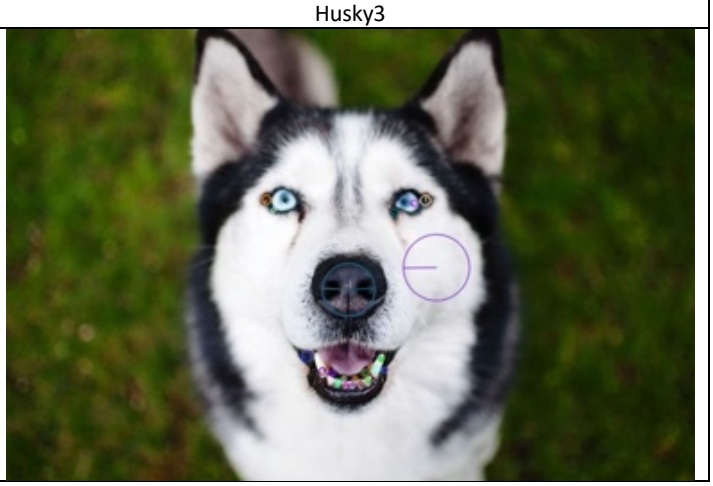
Here is the terminal output for *BoVW* result and histogram visualization[11].

```
Features Detected in Each Img
561 888 376 551
(Normalized) Histogram of the Images
0.160428 0.0855615 0.135472 0.0427808 0.158645 0.112299 0.0445633 0.26025
0.226351 0.113739 0.108108 0.108108 0.0844594 0.0405405 0.121622 0.197072
0.117647 0.106952 0.0998217 0.0802139 0.0516934 0.0499109 0.0855615 0.0784314
0.292196 0.0562614 0.0707804 0.114338 0.0744102 0.0961887 0.076225 0.219601
PairWise Distances
0 0.0307386 0.0550298 0.0375367
0.0307386 0 0.0292467 0.0148374
0.0550298 0.0292467 0 0.0577184
0.0375367 0.0148374 0.0577184 0
```



Feature Extraction & Matching:

1. **Extracted Features (We've generated another feature restricted version of the features sorted by cv::KeyPoints::response, for analysis):**

Husky2	Husky3
	

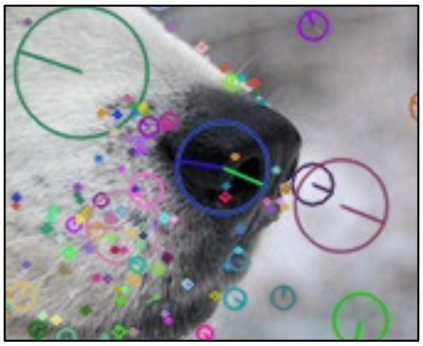
a. We can see that sorted features appear to be more reasonable and mostly lie on the object of interest.

b. The locations of features make sense too. They are mainly distributed on center of eyes, center of noses, dog's forehead etc.

c. ***<u>As afore mentioned, the orientation of the key point is the statistical majority of around the neighborhood.</u>*** **They are all determined by magnitude orientation. For example, see the features on the trees, on the forehead of the left dog where it's pointing with the directions of the fur, and ears of the left dog where the direction is pointing perpendicular to the white-black edges.**

We've mentioned before that there may be more than one feature detected for a specific location. And they distinguish their selves by orientation. See the nose part of the right dog, where we can see two orientations assigned to the "same" feature point. Here is a more obvious example in *blue circle*:

[11]Visualization using Python, by reading the csv file written by C++. Related library used: *Matplotlib.pyplot & mpl_toolkits.mplot3d*.



But those near round & uniform features such as the black circular area to the left of the right eye of the left dog was assigned some ~240 degrees and the corresponding part of the right eye as assigned ~80. SIFT seems to be confused in such cases.

d. **The locality of *SIFT* does rises problems**. **SIFT *produces* a lot of responses to every teeth of the right dog, which is not what we've wanted ideally.** We do need one feature of the tooth, but not that many. It is not a problem when we not are limiting the total number of features, but it occupies the spaces for other features in limited situations.

e. **Following the previous point, we could realize one important issue in utilizing *SIFT* features are filtering out unnecessary features. We may not want features in the background and generate more features on the object. And it turned to be difficult by merely thresholding / judging by response magnitudes**. Some features are quite important & obvious, but they do have limited responses, e.g. root of the beard & nose:

They are almost blob shaped + white-black areas, but they got still filtered away. But if we allow more features, background features will pop up. Due to time constraints, we are leaving this topic for further investigation.

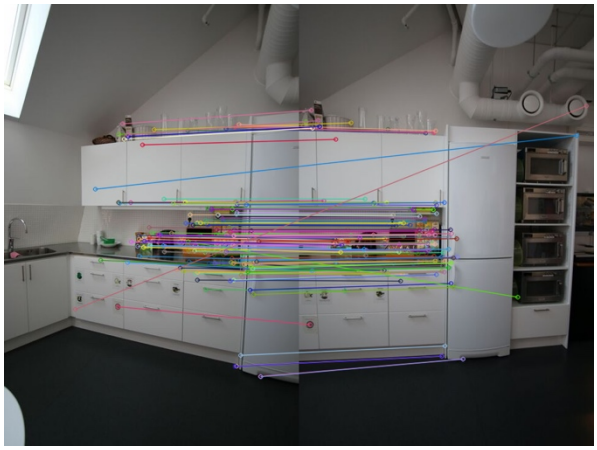
f. **Scales of features is not equivalent to responses**. This is obvious. Scales are presented in the radius of the circles, but some of the biggest features still got filtered away, because they contain too much information thus will likely have a low contrast / response.

Notice the big green circle. It's probability referring to the whole dog's face.

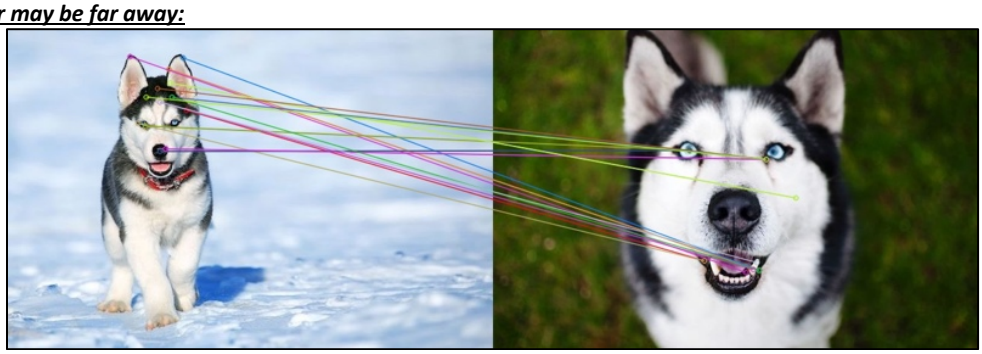
g. **Reasons are unknown why *SIFT* seems to be neglecting dog's eye most of the time.** We are mentioning it because it may be the first feature humans think of when talking about feature generation. **One reason we think of may be that eyes here are mostly irregular shaped because of angles and black fur around it. Another reason may be that scales may lie just between to *octaves* thus creating responses however weak on both octaves.**

2. Matching Results:

a. It turned out that matching is not that successful as it is in assignment 3. There could be some reason as far as we're concerned (Other than the fact that in assignment 3 we used SURF):



i. ***<u>The most important setback in this matching method using Nearest Neighbor, no matter in its vanilla form or noise robustness form[1715] or FLANN, we are anyhow establishing a match for every feature even though the nearest neighbor may be far away:</u>***



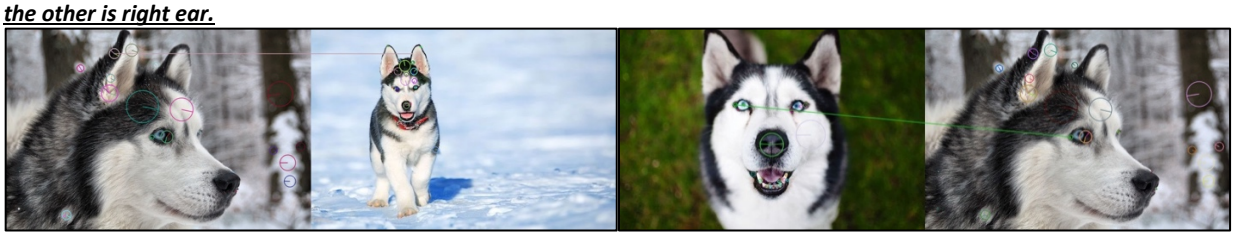
A majority of the features to the left goes to the left eye & one / two teeth on the right. This situation is worsened if we don't have high quality feature points. **Also another common situation is where there is no corresponding feature at the very start, like Husky2 in just a side photo of Husky while others are front photos.** We should allow to reject matching request for certain features, returning the message "no matching exists".

ii. ***<u>Nearest neighbor relationship is not commutative</u>***. *Feature 2* is the nearest neighbor of *feature 1,* but *feature 1* is not necessarily the nearest neighbor of *feature 2*. This situation is discussed in unsupervised learning, e.g. spectral clustering utilizes pairwise distance to cluster the dataset. **This happens when some of the pre-assumed clusters are heavily overlapping, in this case, features are alike and irrelevant factors dominate being the nearest neighbor or not, e.g. noises.** We didn't explicitly experiment to find proof for this argument, but it's likely to be the case.

iii. The colors are rather bold in all for images. For *husky*, there are few colors apart from black, white. In assignment 3, we have many different, independent object to extract features from.

b. <u>But from the matches of the largest scale features, we could see that SIFT does succeed in creating rotation invariant & scale invariant features because the two ends of 6 matching showed is not same.</u>

c. <u>There're still correct matches like "Husky2-Husky3", but sadly "Husky3-Husky2" is not correct. We could also take "Husky2-Husky1" as correct since they are both the located on the black & white fur boundary of the ear, though one is left ear and the other is right ear.</u>



3. BoVW:

a. From the third row of the pairwise results, we could see that BoVW concludes *Husky3* to be most similar to *Husky2* while the distances to *Husky1* & *Puppy* is roughly the same. This result is expected to some extent, but not ideally what we wanted

b. Husky2 & Husky3 only contains the head part of the dog, while the other two have a full picture of dogs. This is the correct part.

c. But we know that there are three huskies and BoVW is not distinguishing *Husky1 & Puppy*. And this is the incorrect part.

d. We could see the main disagreement between *Husky1 & Husky3* are *Cluster4 & Cluster5*, which is exactly where *Husky2 & Husky3* are similar. However only by *Cluster1 & Cluster2*, we could see that three huskies are similar.

e. Since we're using all features produced by *SIFT*, **the considerable amount of background features may be the reason that droves *Husky1* away from *Husky3*.**

f. Another reason could be improper values of the clusters. **The images are rather simple thus a bigger K could make the model more vulnerable to noises.**

[15] https://docs.opencv.org/3.4/d5/d6f/tutorial_feature_flann_matcher.html

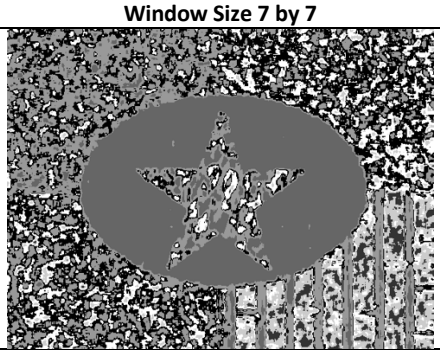
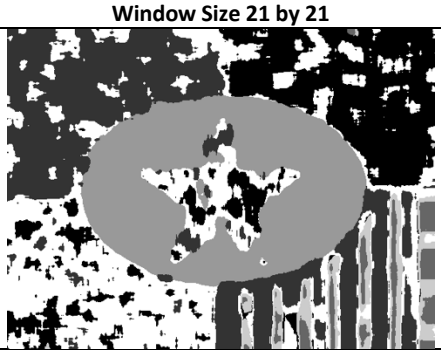
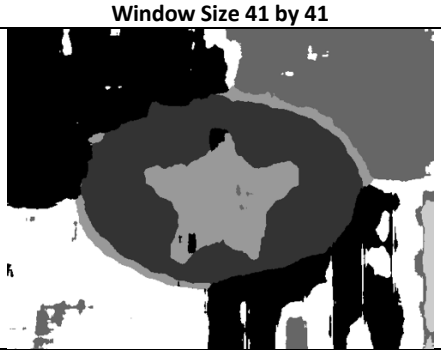
Appendix:

All referenced code is cited in the footnote in the previous sections.

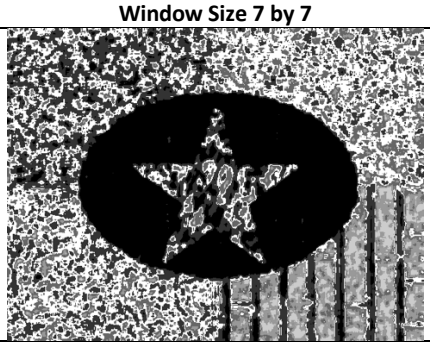
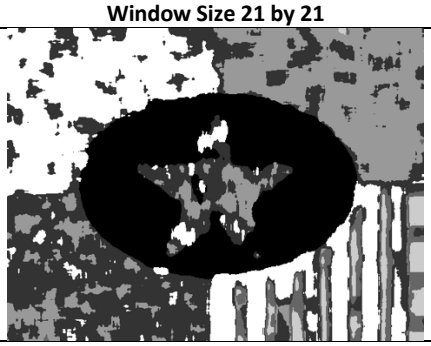
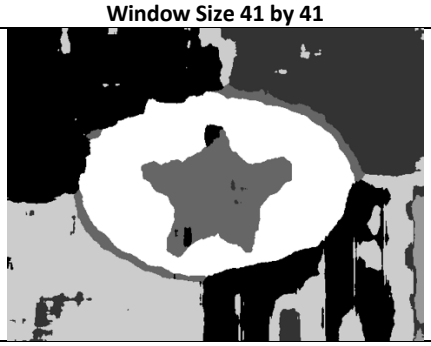
Matrix_ToolBox:	Self-implemented class with static methods for matrix allocation & calculation, including matrix-matrix / vector-matrix / matrix- vector multiplication, inversion & transpose.
myKernels:	Self-defined kernel class to encapsulate dimension information into Lows kernel used.
myUtils:	Self-implemented class with static methods for image IO, image operations like binarization, cropping, padding & copying.
myModules::FeatureExtractor myModules::PCA myModules::KMeans myModules::SVM myModules::RandomForest	Class encapsulated with self-implemented algorithms like K-Means, and also OpenCV machine learning objects to fit into the format of the data pipeline. The APIs are inspired by Scikit-Learn.
ee569_hw4_sol	Highest level class to getting results in the report.

Results:

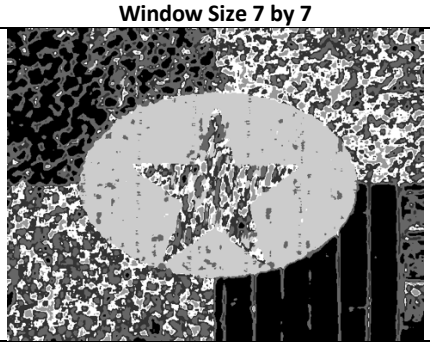
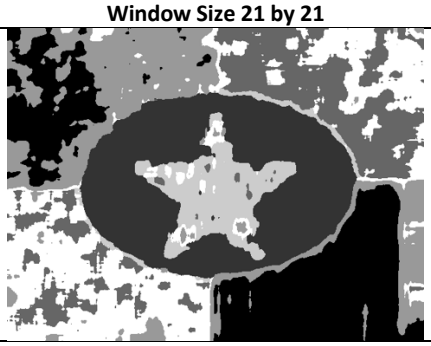
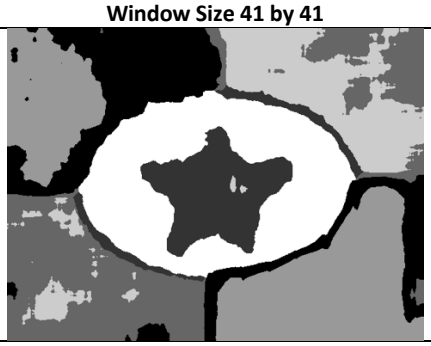
Direct approach (Normalization (a) + Window Mean + 15D features)

Window Size 7 by 7	Window Size 21 by 21	Window Size 41 by 41
		

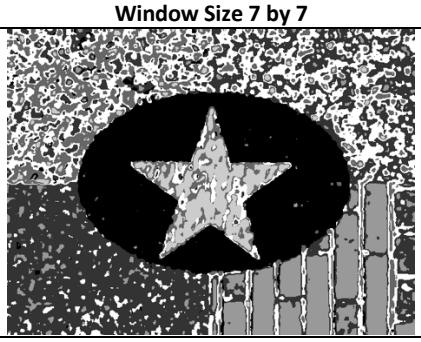
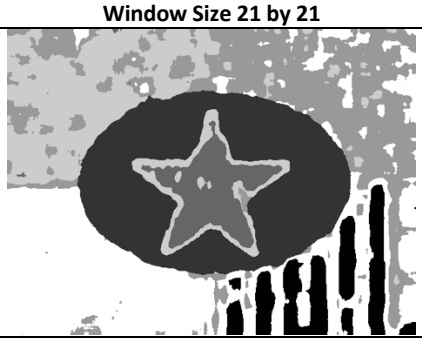
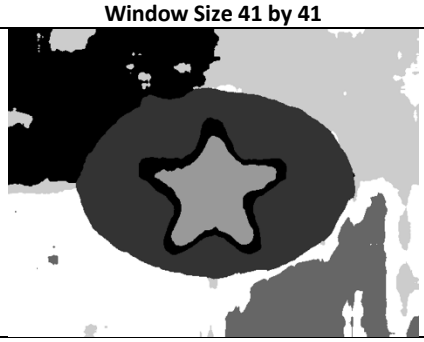
Improved approach (i) (Normalization (a) + Window Mean + 3D features)

Window Size 7 by 7	Window Size 21 by 21	Window Size 41 by 41
		

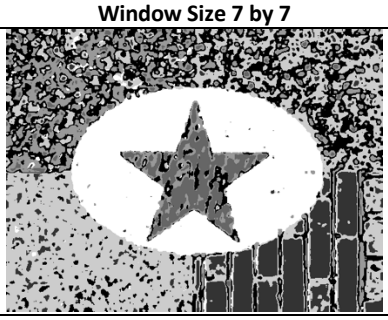
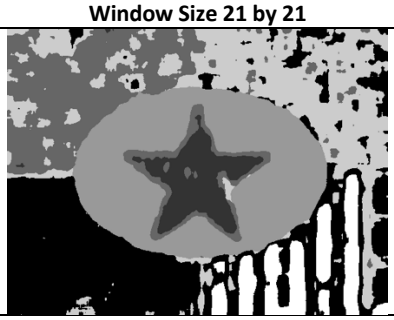
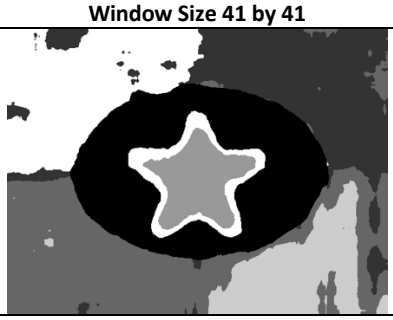
Improved approach (ii) (Normalization (a) + Global Mean + 3D features)

Window Size 7 by 7	Window Size 21 by 21	Window Size 41 by 41
		

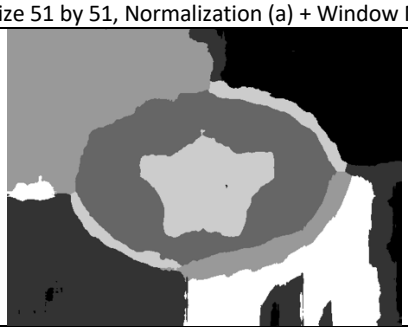
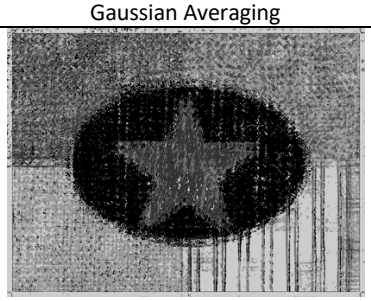
Improved approach (iii) (Normalization (b) + Global Mean + 3D features)

Window Size 7 by 7	Window Size 21 by 21	Window Size 41 by 41
		

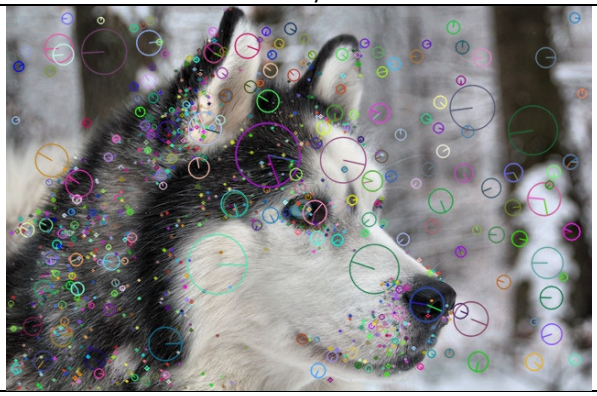
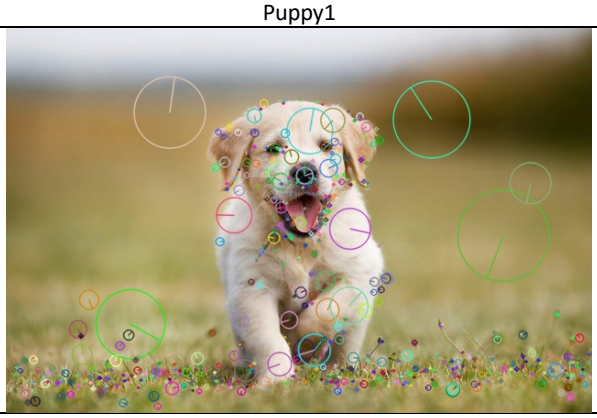
Improved approach (iv) (Normalization (b) + Window Mean + 3D features)

Window Size 7 by 7	Window Size 21 by 21	Window Size 41 by 41
		

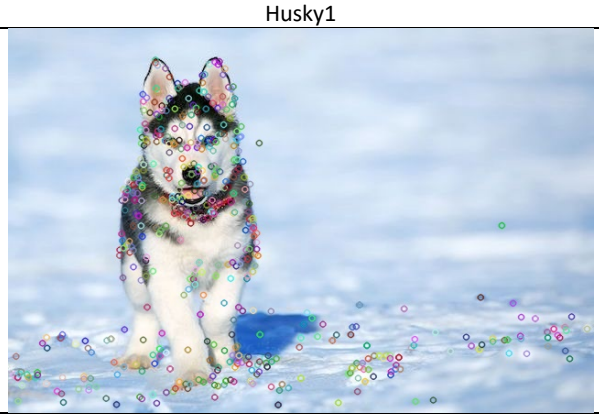
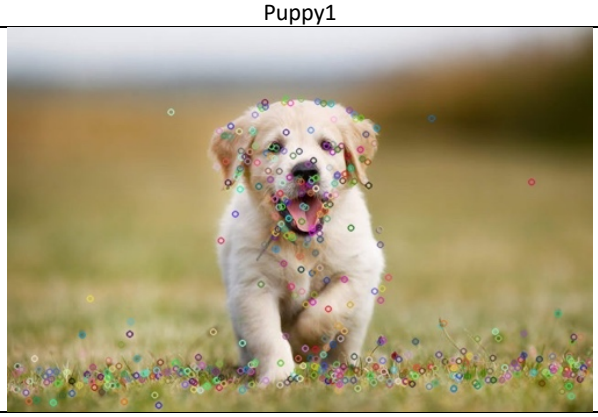
Other Trials:

Window Size 51 by 51, Normalization (a) + Window Mean + 3D	Gaussian Averaging
	

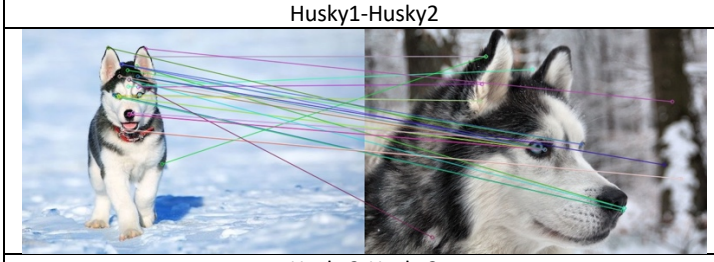
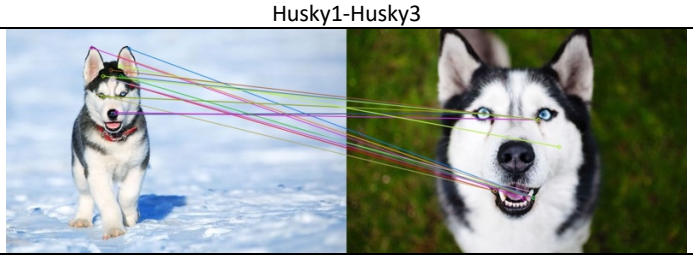
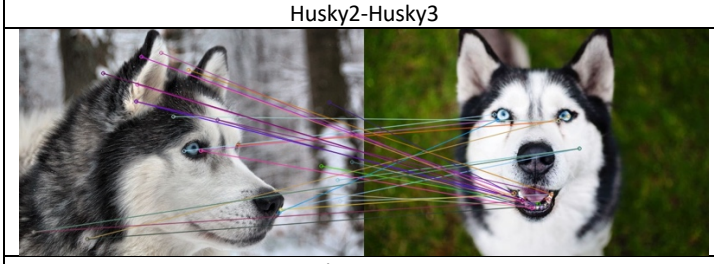
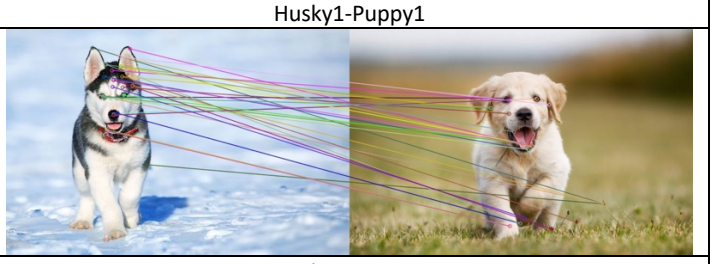
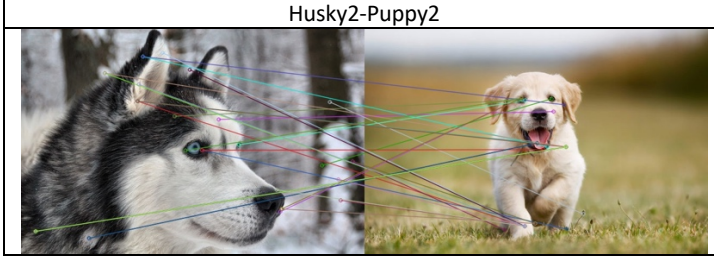
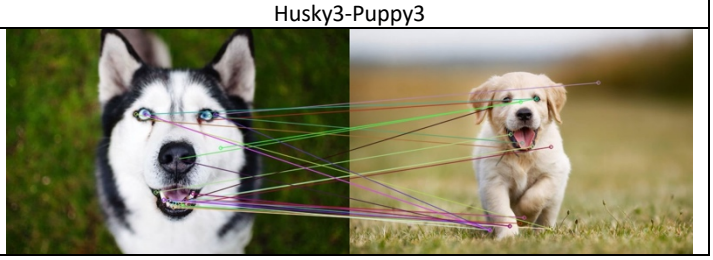
Feature Extraction with Unlimited Feature Numbers (Rich Points):

Husky1	Husky2
	
Husky3	Puppy1
	

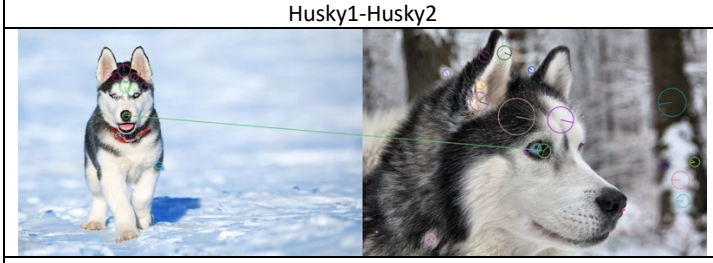
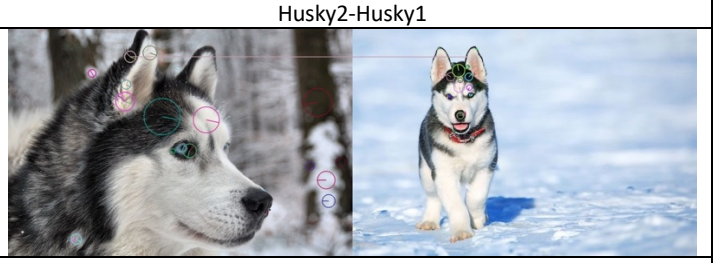
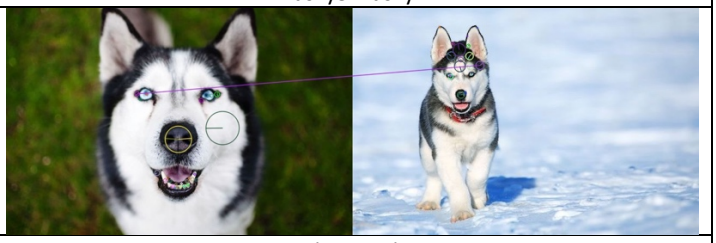
Feature Extraction with Unlimited Feature Numbers (Only Points):

Husky1	Husky2
	
Husky3	Puppy1
	

Feature Matching with Limited Feature Numbers (30)

Husky1-Husky2	Husky1-Husky3
	
Husky2-Husky3	**Husky1-Puppy1**
	
Husky2-Puppy2	**Husky3-Puppy3**
	

Largest Scale Feature Matching (Query|left – Train|Right)
(The largest scale feature is the one in the query / left image)

Husky1-Husky2	Husky2-Husky1
	
Husky1-Husky3	**Husky3-Husky1**
	
Husky2-Husky3	**Husky3-Husky2**