Examine SIFT Descriptor on Cat Image Matching

Team member:
Mengxi Zhou (.2656)
Jincheng Li (.7004)
Yun Jiang (.1982)

1. Idea and Problem definition

In this project, we experiment with the SIFT descriptor to see how it is able to measure the similarity between cat Images. Our inital thought was SIFT might find images where the cats had similar postures. Thus, the problem could be defined as: given a dataset consisting of cat images and an unseen input cat image, find the top 5 images in the dataset that are most similar to the input image in terms of the cat posture. This method contained several computer vision techniques: interest points finding, image segmentation, difference of Gaussian, SIFT descriptor and the matching for SIFT descriptors.

The dataset is taken from Kaggle: https://www.kaggle.com/crawford/cat-dataset . The original dataset contains ~10K cat images and we take 1K from it to form our dataset. We randomly pick several images from the remaining 9K as new unseen images.

2. Proposed method

A. Pipeline

1) Preprocess the dataset:

For each image in the dataset, do the following:

I. Find interest points on the image.
II. Do image segmentation or edge detection on the image and filter out those interest points that are not around the cat.
III. Do Difference-of-Gaussian to find proper regions for the interest points.
IV. Build SIFT descriptors for the interest 'regions'.
V. Save the SIFT descriptors for further matching use.

2) Process incoming new image:

For the incoming new image, do the following:

I. Do steps I - IV as above. Denote this generated SIFT descriptors as new_SIFT.
II. For each candidate image in the dataset, get the saved SIFT descriptors (old_SIFT) for the image. Do a matching between new_SIFT and old_SIFT and generate an overall matching score for this candidate image.
III. Take the best 5 matching candidates and show the matchings.

B. Interest point detection

We tried FAST and Harris detector to find interest points in the images. And we decided to go with Harris detector for the reason that: 1) It is faster than the FAST algorithm, especially when we are dealing with large images; 2) The R value generated by Harris detector is more comparable between interest points. Thus, we can select the best N interest points for each image to save computation time.

We only retain the top 100 interest points in terms of their R value to save computation time.

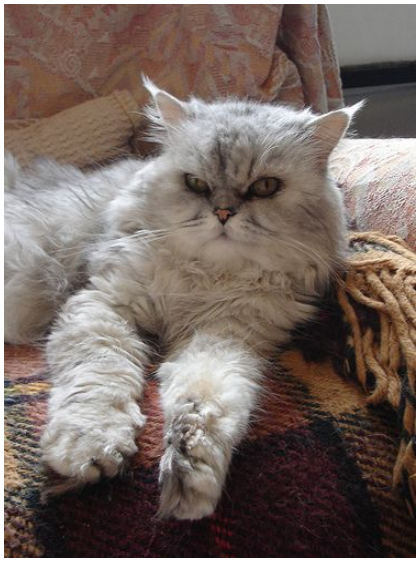
C. Image segmentation

Since our major task is to compare the cat's posture between images, we are more interested in those interest points around the 'cat' area. This prevents us from matching two interest points on the background materials.

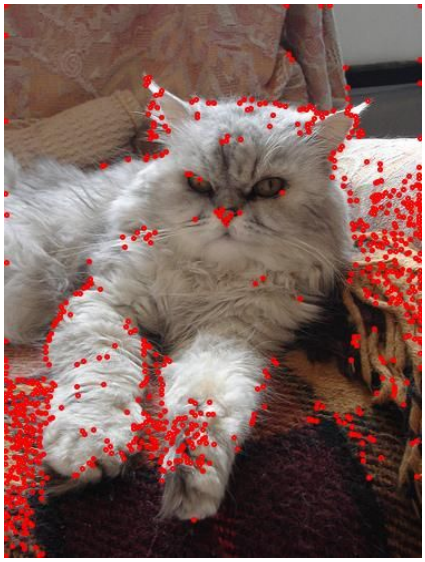
Thus, we tried several approaches to take the 'cat' area out of the original image. More detailed descriptions about this part will be provided in the next section.

D. Difference-of-Gaussian

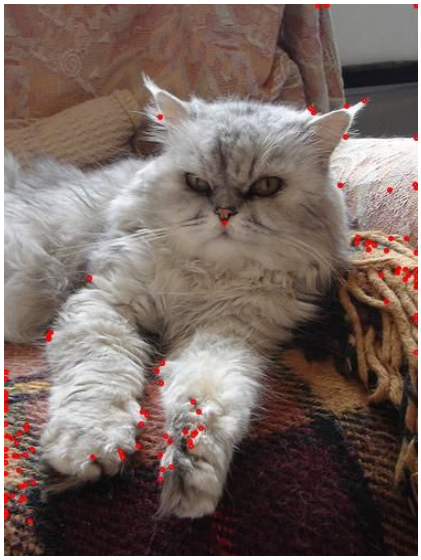
To get scale-invariant interest regions, we implemented the Difference-of Gaussian method. We use 5 levels of 'Octave' and within each 'Octave' level 5 images are generated to compute DoG.



Original Image



Interest Points (All)



Interest Points (Top 100)

Interest Regions (Top 100)

The resulting DoG scales along with the interest point locations are passed to the next step to generate SIFT descriptors.

E. SIFT descriptor

This algorithm was invented by Dr. Davis Lowe in 2004 [2]. The method extracted keypoints and then computed their descriptors. For the descriptor, the algorithm divided a patch into 16 cells and computed Gaussian weighted histogram of gradient directions for all pixels inside the cells. The resulting descriptor had 128 element feature vector for each keypoint.

F. Matching Method

To match a new input image to the closest (most similar) image in the dataset, we did the following:

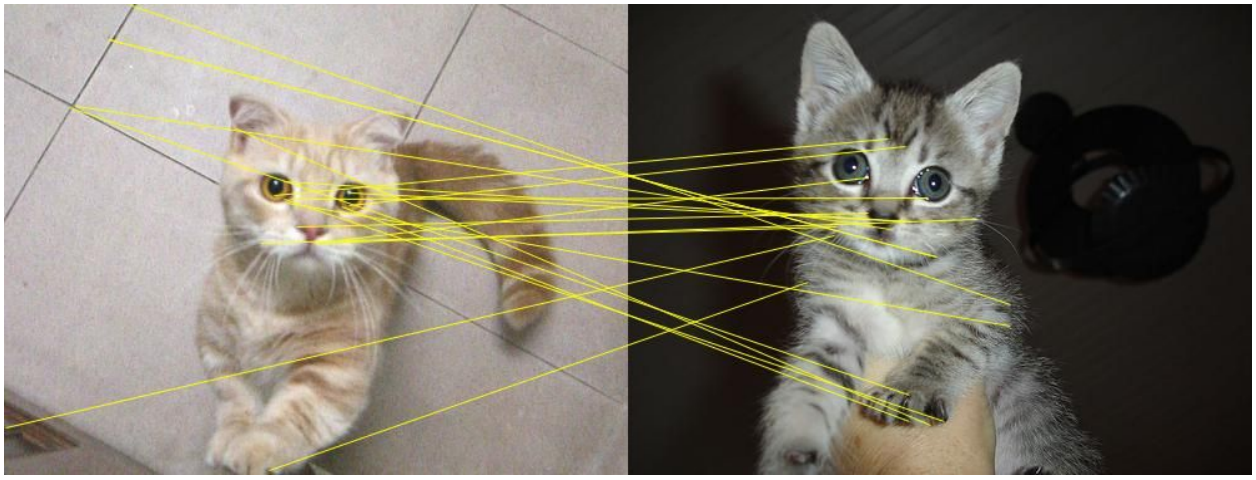
I. Take the SIFT descriptors set for the new image (A), mark it as SIFT_new_set.
II. For each image (B) in the original image dataset:
    A. Take the SIFT descriptors set for this image, mark it as SIFT_old_set.
    B. For each SIFT descriptor in SIFT_new_set, find the closest SIFT descriptor in SIFT_old_set in terms of the Euclidean space.
    C. After we have matchings for all SIFT descriptors in SIFT_new_set. Take the best 20 descriptor matchings and sum the distances as a distance from A to B. (If the size of SIFT_old_set is smaller than 20, then take all descriptors in SIFT_old_set into consideration.)
III. Find the closest 5 image Bs in terms of its distance to A and show the matchings.

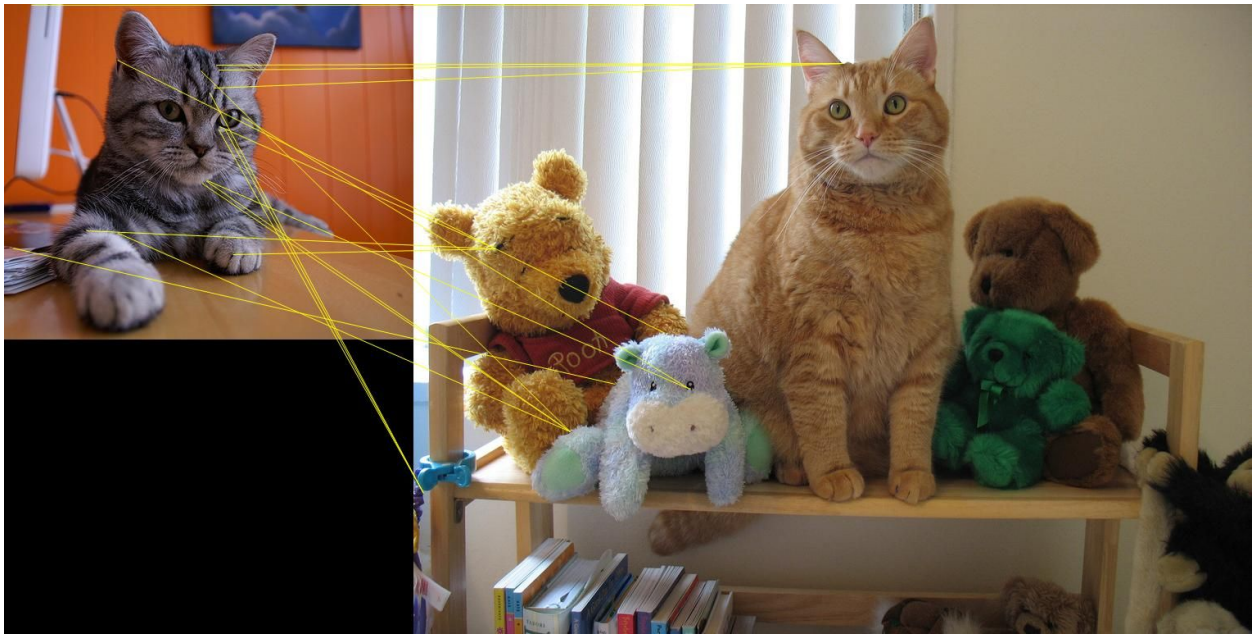
G. Draw Matched Points

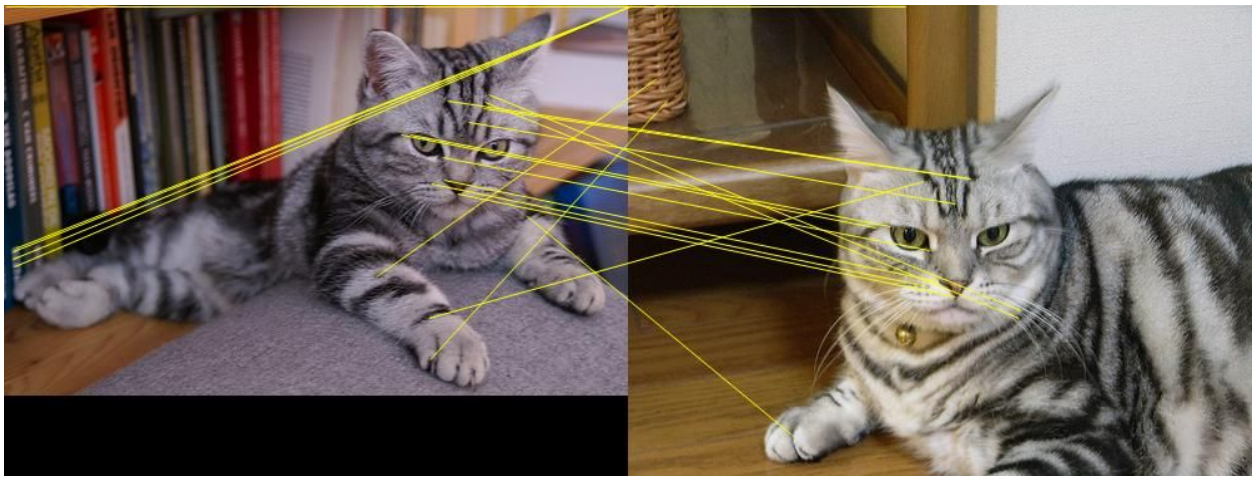
We first made the two images that have the matched points the same height by inserting empty rows to the one which has fewer rows and then appended the images. After that, we drew lines joining matches using different colors.

Here we show some example matching image pairs without applying image segmentation. The new unseen image is on the left side and the image from the dataset is on the right side. The yellow lines indicate the matching interest points correspondingly.

As you can see from the examples, since we haven't applied image segmentation, a lot of interest points on the background are included in the matching, which creates some undesired effects. Thus, we further investigated several methods to segment the image and to only take those interest points around the cat into consideration.







3. Image Segmentation

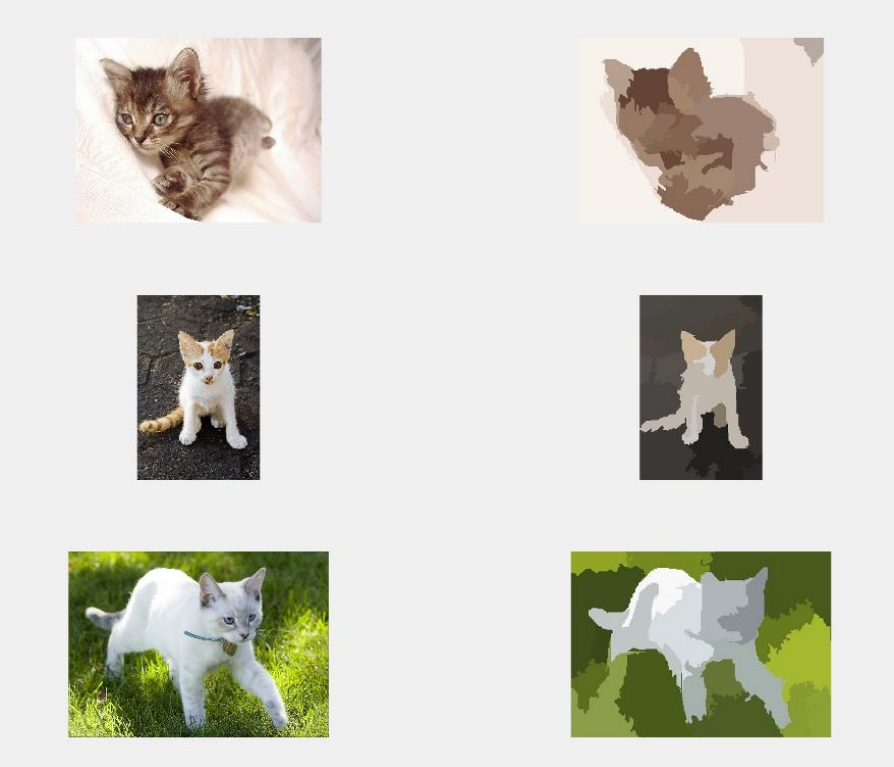
Preprocessing

Some images in the dataset have large sizes, so we resized them without changing the resolution to reduce the computation. Additionally, we did a Gaussian smoothing on the image because the color of the cat is not monotone.

Segmenting

We implemented Efficient Graph-Based Image Segmentation(Felzenszwalb and Huttenlocher, 2004)[1] because this algorithm has good performance with low computation cost. Figure 1 shows the result of the image segmentation(with 20 segments). It took 2~3 seconds to segment an image. This algorithm is based on Kruskal's MST algorithm on an undirected weighted graph which represents the image. Each pixel is a vertex, and it has at most 8 edges connecting with its neighbors at 8 directions. Edge weight is defined by weighted Euclidean distance between the two pixels in RGB space. Here are the steps of this algorithm.

1. Initialize the segmentation pixel-wisely which means each pixel is a segment.
2. Compute the weight of edges and sort them ascendingly.
3. For each edge that connects pixels from different segments, if its weight is less than or equal to some threshold T that T is a function of the internal differences and the size of the two segments, we will merge these two segments. T is relatively large when the segment size is small, so it is harder to merge when the segments grow bigger during the process.



The time complexity is O(|E|log|E|) where |E| is the number of edges. Since |E|<=8|V| where |V| is the number of vertices, we can also say that it is O(|V|log|V|).

In practical, we encountered some problems. We found that the number of segments is more than our expectation after running the algorithm above. We addressed the problem by KNN assigning and neighbor merging. Here are the steps of the algorithm.

1. Sort the segments by their size descendingly and remove segments one by one until there was less than 20% of the image remaining. Define the remaining segments as minor segments and those removed as major segments.
2. Find the k-nearest neighbor(k=5) for each pixel in minor segments and decide which major segment to be assigned to.

Statistically, there were 30~120 segments left, depends on the image. Then, we continue to merge small segments with their neighbors. Here are the steps of the algorithm.

1. Compute the periphery of the small segment we want to merge by dilation and subtraction.
2. Compute the average color distances(the same method as before) between this segment and all neighboring segments which own at least 15% of the periphery.
3. Merge the small segment with the most similar neighboring segment(the one that has the shortest distance).

After neighbor merging, we were able to reduce the number of segments to a very small number. We decided to stop this algorithm when 20 segments left because some cat segments would merge with some background segments if we continued merging to a lower number.

4. Segment Classification

By image segmentation we were able to create a 20-segmented image based on its color. However we wanted to classify each segment(pixel) into 2 classes(0-background, 1-cat).

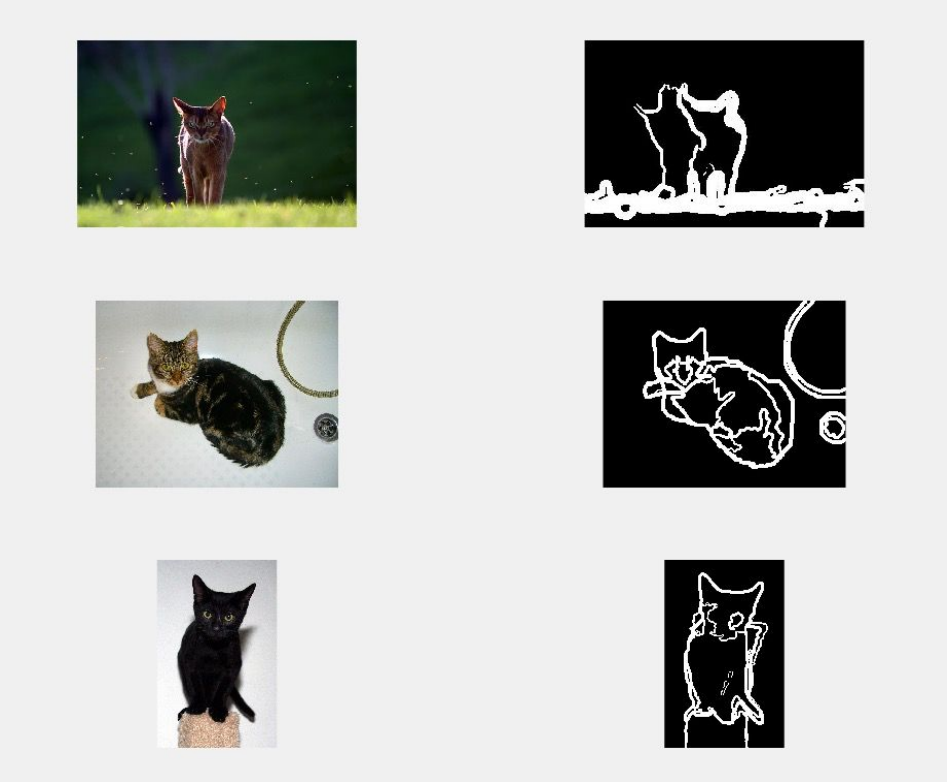
There was a lot of hardship when generating a binary image. We used 3 different methods.

K-means

We tried K-means(K=2) first, but the stochastic initialization would get different results when running multiple times. Additionally, background segments were merged with the cat segments in many images.

Edge detection

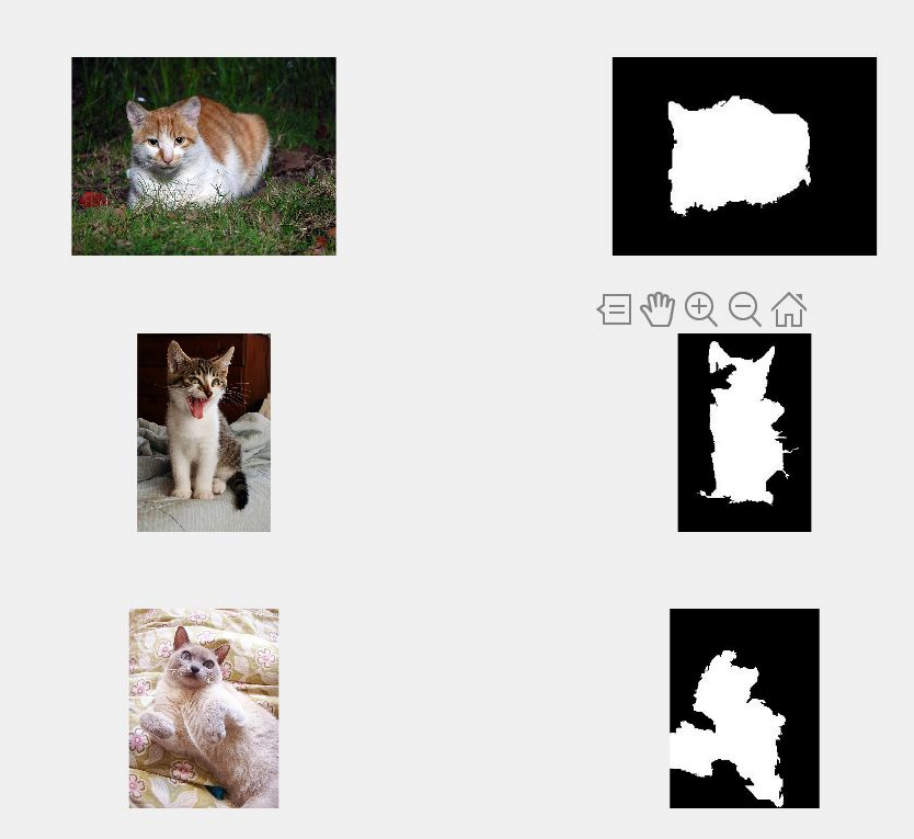
Then we tried another technique, edge detection. In this case, 1 represents edges and 0 represents internal areas. We ran an edge detection by Gaussian derivative on the 20-segment image. Here are some results of edge detection.



As shown in the results, some edges of background also involved in the image, so this would definitely have some impact on the posture recognition.

Location-based classifying

Based on the assumption that the cat would probably appear at the central part of the image, location-based classifying was a reasonable method. In this method, we computed the mean of each segment with respect to its standardized x,y coordinates. Then we classified the segments by a threshold T that the segment whose mean was lower than the threshold would be classified as cat, otherwise it was background. After experimentation, T was set to 0.6. Here are some results of location-based classifying.

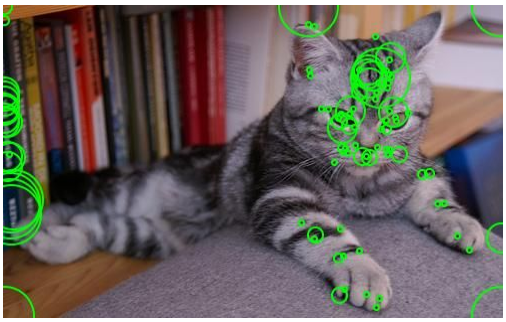


We also thought that T should be flexible because the proportion of the cat varied in images. However, we did not find a good solution to compute T from the image itself.

5. Results

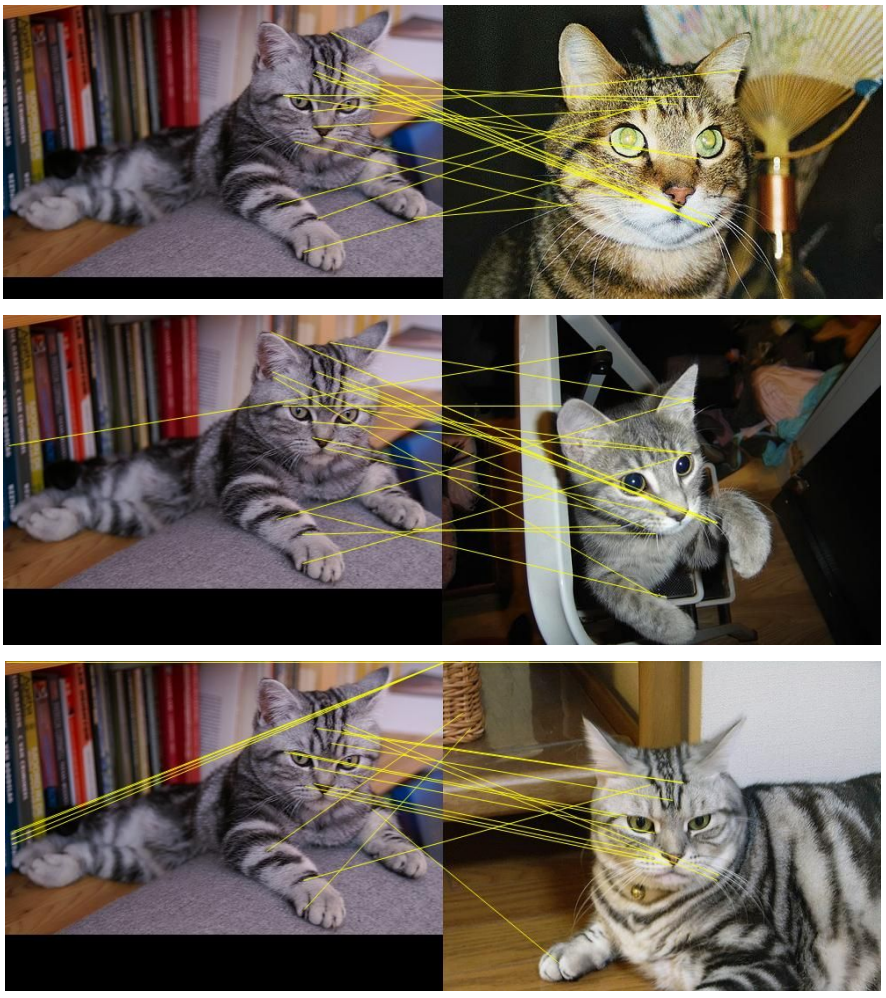
a. Without Image segmentation

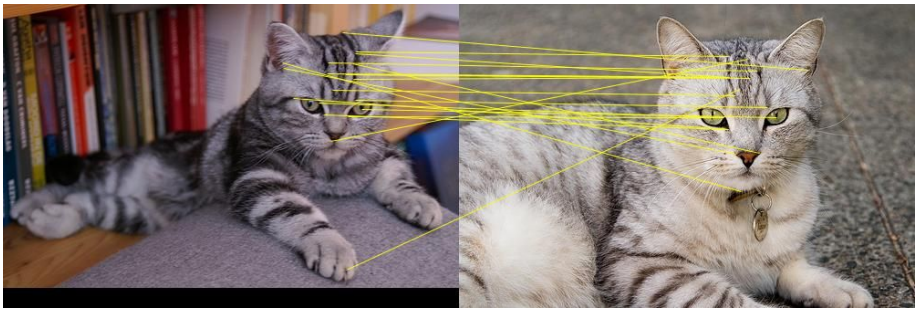
Original Unseen Image | Interest Points Image

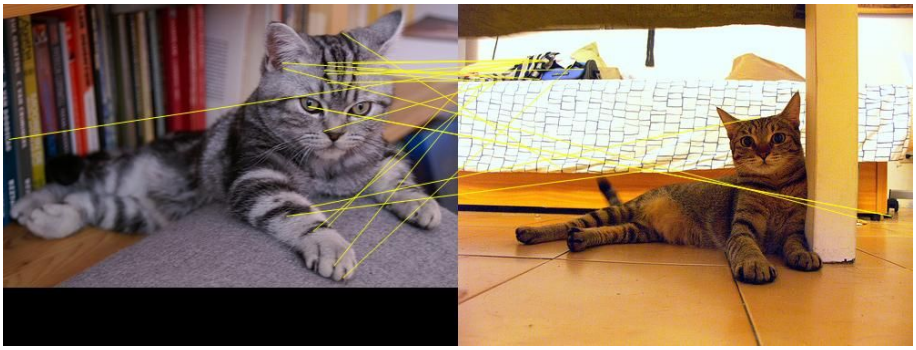


Interest Region Image

Best 5 matches from the dataset:





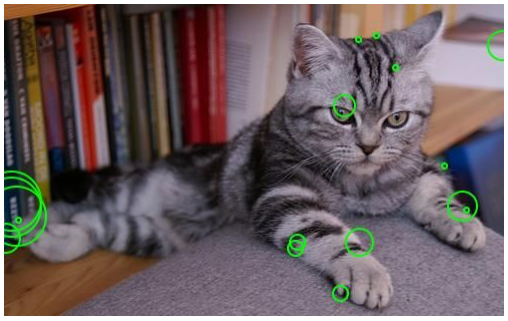


b. With edge detection

Original Unseen Image

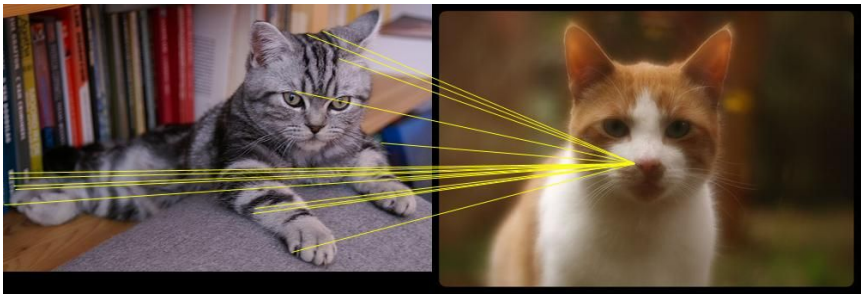
Edge Filter Image

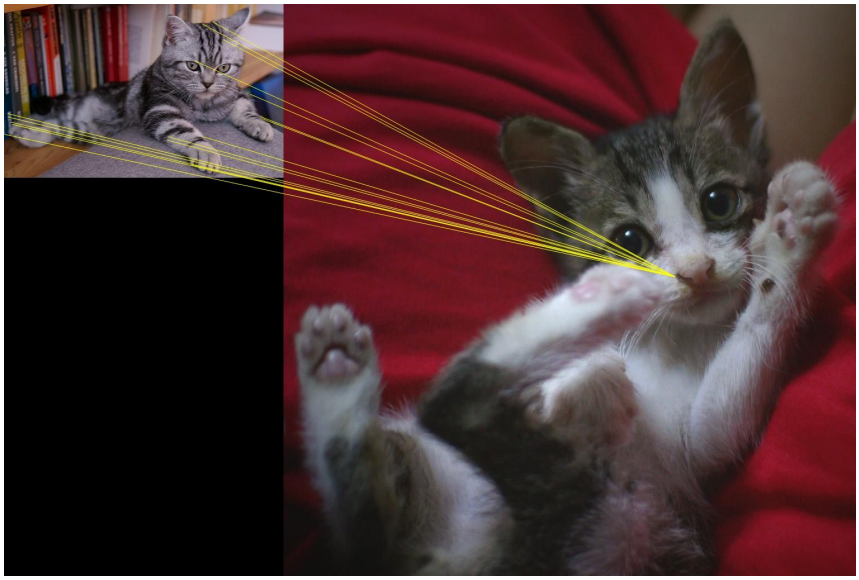
Interest Point Image

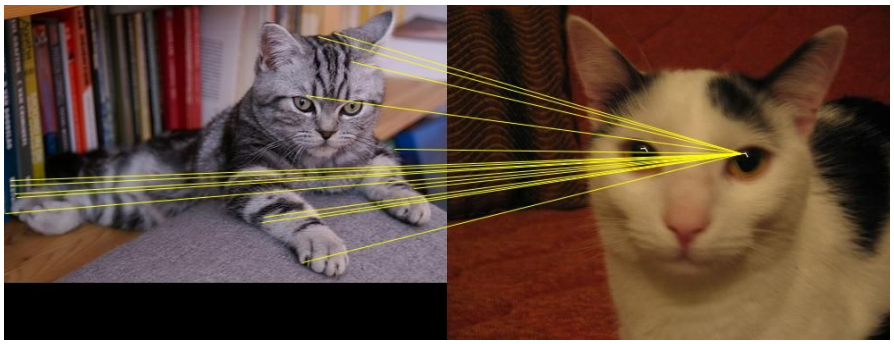


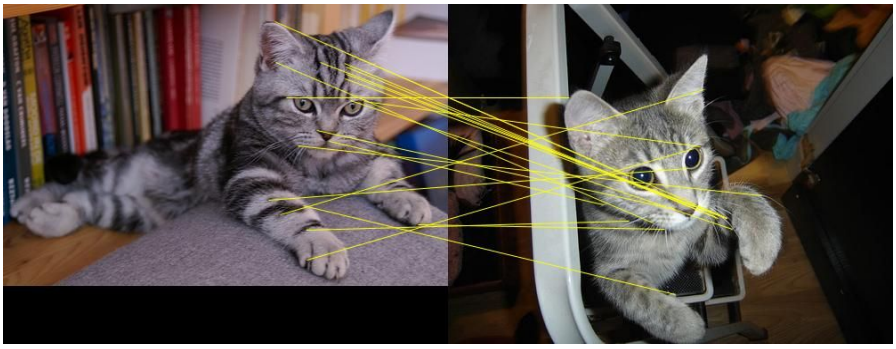
Interest Region Image

Best 5 matches from the dataset:









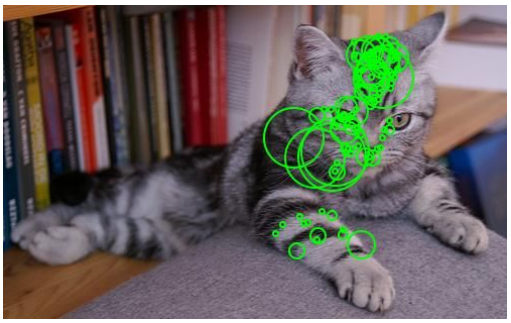
As we can see here, taking only the interest points on the edges does not help our process. A lot of interest points are lost and the remaining interest points are still not guaranteed to be around the cat. Also, a lot of interest points on the unseen image side find the same interest point on the seen image to be their best matches, which produces meaningless results.

c. With image segmentation

Original Unseen Image

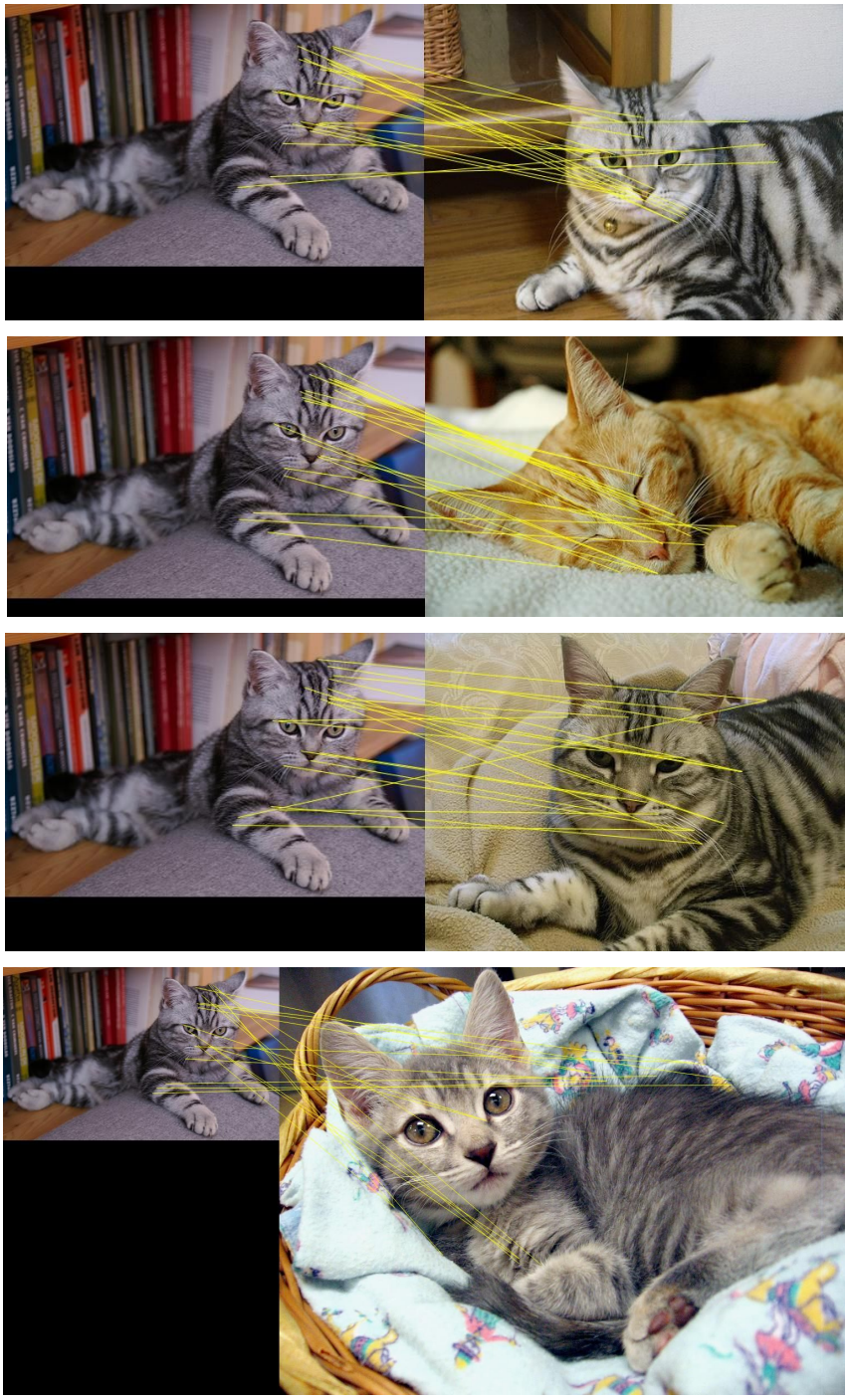
Segmentation Filter Image

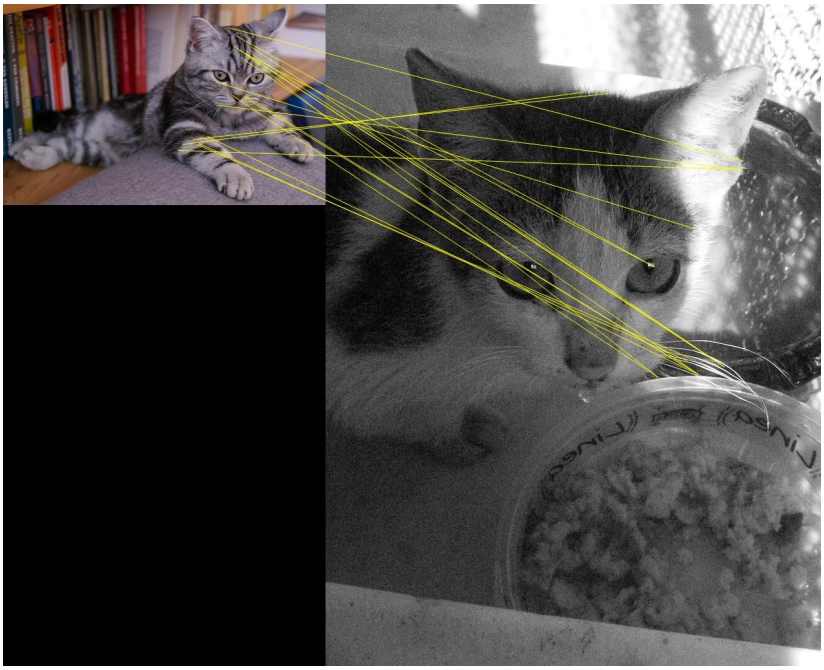
Interest Point Image



Interest Region Image

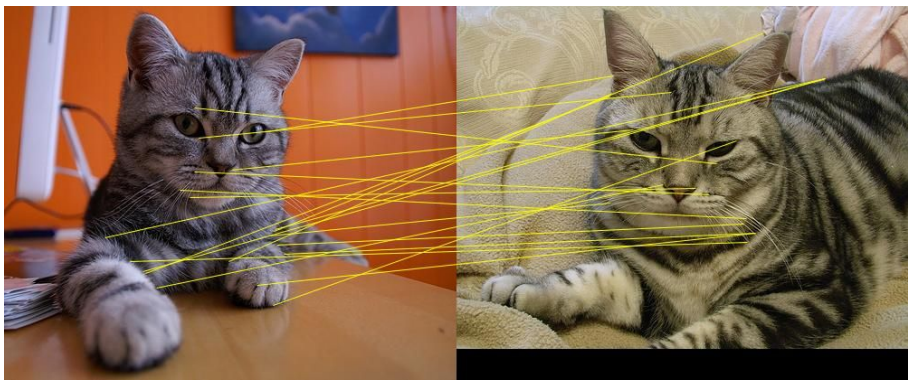
Best 5 matches from the dataset:

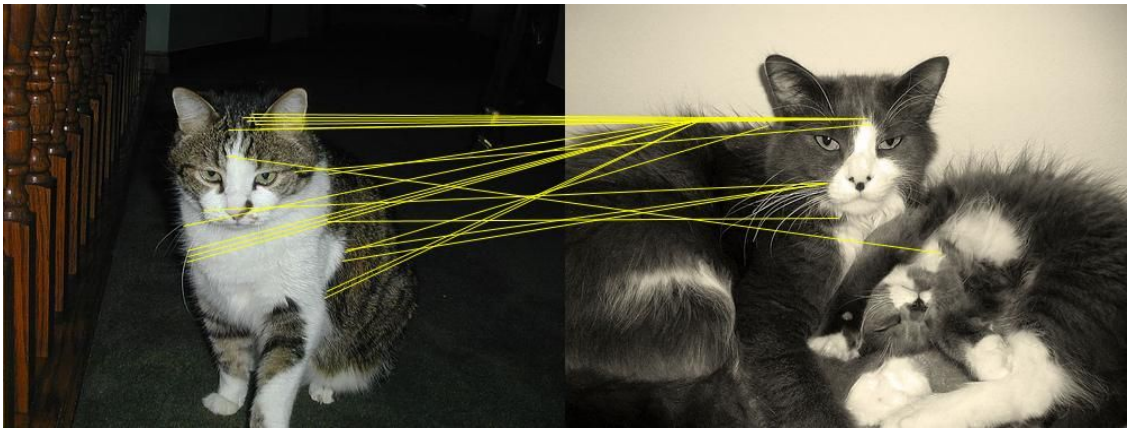


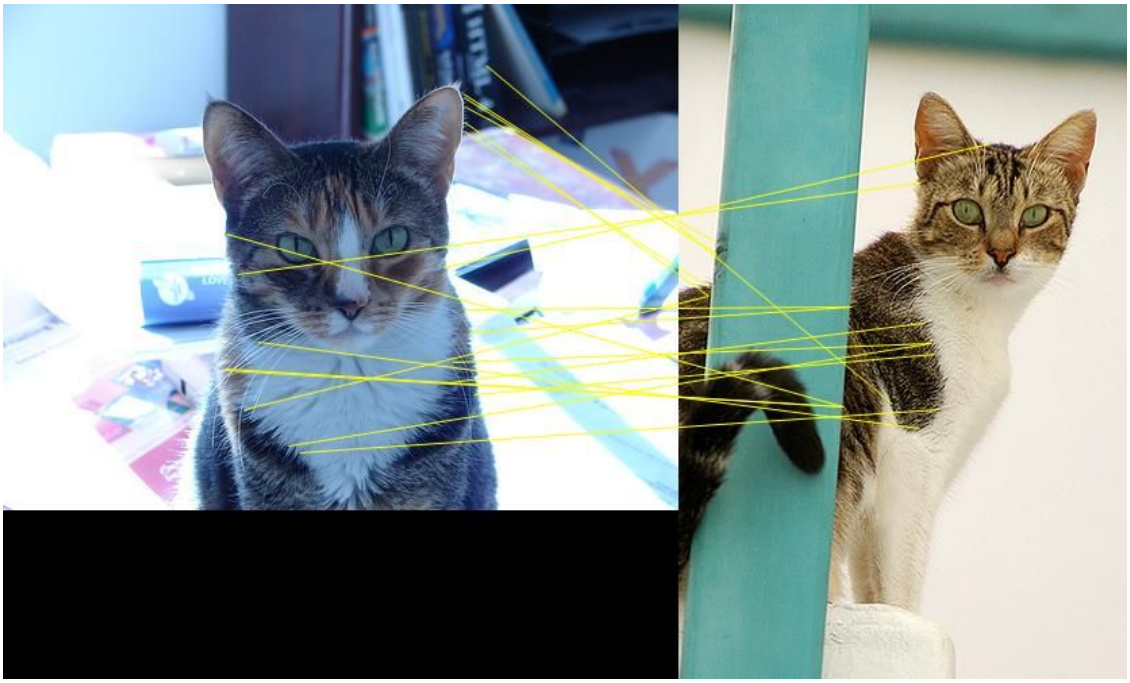


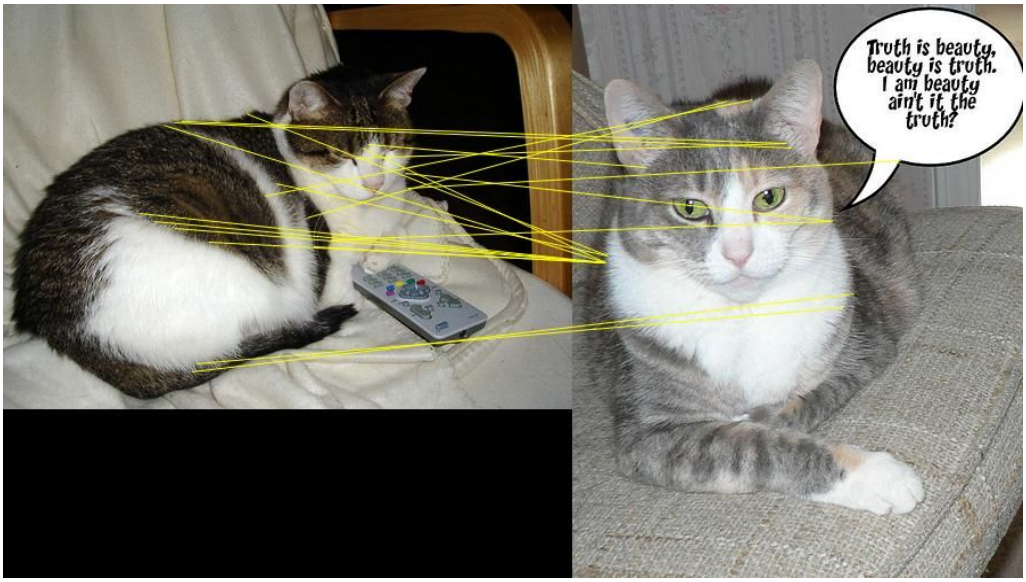
As we can see from the top matchings. SIFT descriptors are not able to find cats with similar postures. Instead, it is more likely to find cats with **similar textures**.

More matching results are shown below:








Truth is beauty,
beauty is truth.
I am beauty
ain't it the
truth?

6. Lessons Learned

a. SIFT descriptors may not be proper features for general image similarity comparison. It is more suitable to find 'exact matchings', i.e. for two images that contain one common object, match the object from one image to another.
b. Hard to get a consistent threshold on Harris detector's R value for all images. One threshold may find interest points pretty good on one image but fails on another.
c. Do sufficient research before starting work.
d. The diversity of the color is more than what we expected. There are some other color spaces that could be better than the normal RGB one.
e. The computer needs to know what is a cat. It cannot recognize a cat by given only some colored segments. Thus some machine learning techniques have to be done to train the computer. The cat can appear in any location of the image, with any size.
f. Think of how humans do these and make computers imitate humans.

7. Allocation of work

Mengxi: I majorly did Interest Points Detectors, Difference-of-Gaussian, Matching algorithm and the overall pipeline programs. I also experimented with the k-means algorithm and a pre-trained semantic segmentation neural network model (they trained the model on ADE20K dataset) to do the image segmentation, but the results are worse than Yun's.

Jincheng: Analyzed the requirements and program specifications. Built SIFT descriptors and the functions to concatenate two images and then draw the matched points and use lines to join the matches. (the last version was also capable of drawing lines using different colors and used a different algorithm, this one is simplified but fits our program) Implemented complete SIFT algorithms (previous versions, using VLFeat in the final version). Experimented with VLFeat library to discover better ways to get SIFT descriptors. Tested the scripts and the whole program.

Yun: Image segmentation and segment classification.

8. Future work

a. For image segmentation, the algorithm we implemented is good enough but it is too hard to recognize which segment is the cat without training. In the future, we will exploit a neural network to train a model that tells us which segment is a cat.
b. A lot of parameters are involved in this project. We didn't have enough time to experiment with different parameter settings. A better parameter set may have a better result.

9. References

[1] Pedro F. Felzenszwalb and Daniel P. Huttenlocher. Efficient Graph-Based Image Segmentation. IJCV 59 (2004)

[2] Lowe, David G. “Distinctive Image Features from Scale-Invariant Keypoints.” *International Journal of Computer Vision*, vol. 60, no. 2, 2004, pp. 91–110., doi:10.1023/b:visi.0000029664.99615.94.