Compute Vision Final Project

410551012

June 21, 2023

Abstract

Use deep architecture and a training strategy to learn a local feature pipeline from scratch, using collections of images without the need for human supervision. Refer to the method of scale invariant network proposed in [OTFY18] to extract feature points, and learn the descriptor according to the Siamese Network in [YVFL16]

1 Goal

The goal of this final project is to enhance the accuracy and robustness of image stitching. We implemented image stitching in out Assignment 2. However, image stitching still has some limitations. For example, even slight changes in viewpoints can significantly affect the stitching result. Changes in viewpoints can lead to poor matching of keypoints, resulting in incorrect or fewer matches. The image below demonstrates a simple test . If I carefully move the camera horizontally and capture images, using SIFT keypoints can achieve good results. However, when I make rough movements, the stitching becomes bad. As you can see, the image on the topside shows a well-stitched when the camera movement is smooth and controlled. On the other hand, the image on the button shows a poorly stitched result due to rough camera movements. Therefore, I propose to optimize image stitching from the perspective of feature matching. By improving the feature matching process, we can enhance the accuracy and robustness of image stitching.

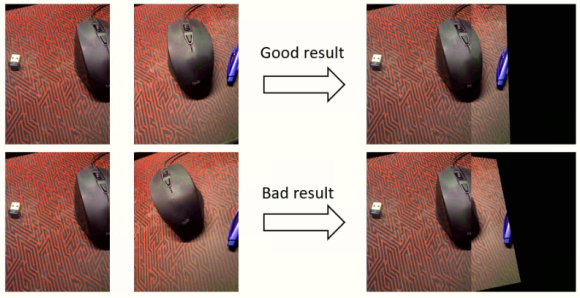

Figure 1: Limitation of image stitching based on SIFT.

2 Methodology

2.1 Self-supervise learning

Since it is self-supervised learning without labels, we need to generate the supervision signal for the model we want to update. Therefore, we create two identical detectors and descriptors. The signal-generating model is denoted as j, while the training model is denoted as i. Based on the rotation invariance, I aim to guide the model's learning and find optimization directions in the absence of labels. Self-supervised learning precisely leverages prior information or specific properties within the data to construct auxiliary tasks that provide training objectives for the model. By rotating the input images and computing the difference between $score_map_i$ and $score_map_j$, I utilize this difference as the loss function.

2.2 Architecture

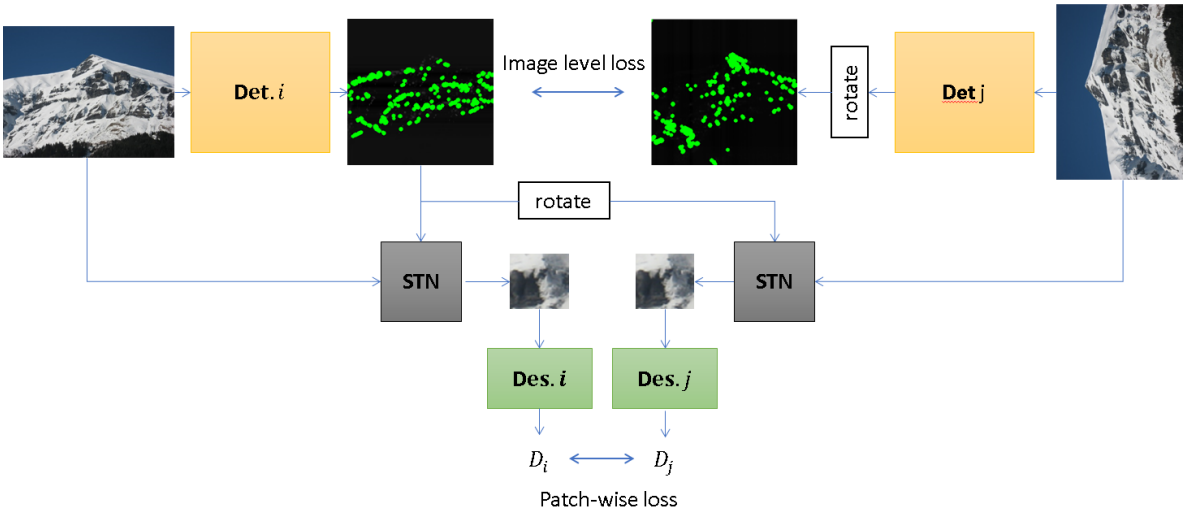

Figure 2: Training pipeline.

The input to model i is the original image, while the input to model j is the rotated version of the input image from model i. After passing through the detector model (both for model i and model j), a score map is obtained. The k keypoints with the highest scores are selected as the keypoints. For model j, since it operates on the rotated image, the predicted score map needs to be rotated back to the original orientation before calculating the loss with $score_map_i$. Once the coordinates of the keypoints are obtained, the original image is cropped based on these coordinates to generate 32×32 patch images. The patches with orientation information are input to the descriptor model, resulting in descriptors for each patch. Let's denote the descriptor for a patch as D. Patch-wise loss is calculated between Di and Dj. After that, only the parameters of model i are updated, while the parameters of model j are copied from the previous iteration's model i weights.

2.3 Feature Extractor

The design of the feature extractor adopts a simple architecture. There are only three convolutional layers in total. After each convolution, leaky relu is used. The final output will be 5 channel, h*w feature maps.

Layer	Output Size
Conv2d	(in_channels, H, W) $\rightarrow$ (32, H, W)
LeakyReLU	(32, H, W) $\rightarrow$ (32, H, W)
Conv2d	(32, H, W) $\rightarrow$ (64, H, W)
LeakyReLU	(64, H, W) $\rightarrow$ (64, H, W)
Conv2d	(64, H, W) $\rightarrow$ (out_channels, H, W)
LeakyReLU	(out_channels, H, W) $\rightarrow$ (out_channels, H, W)

Table 1: Architecture of the feature extractor.

2.4 Scale-Invariant Network

The purpose of this architecture is to achieve scale invariance, by using multi-scale convolutional layers and softmax operations after convolution operations to process input feature maps and make the output have consistent feature representations at different scales. This helps the model to achieve better robustness and generalization ability on inputs of different scales.

Figure 3: Feature maps extracted by feature extractor.

Layer	Output Size
Conv2d	$(1,\ H,\ W) \rightarrow (1,\ H,\ W)$
Conv2d	$(1,\ H,\ W) \rightarrow (1,\ H,\ W)$
Conv2d	$(1,\ H,\ W) \rightarrow (1,\ H,\ W)$
$\vdots$	$\vdots$
Conv2d	$(1,\ H,\ W) \rightarrow (1,\ H,\ W)$

Table 2: Architecture of the ScaleInvariant module.

To generate a scale-space response, we resize it N times, at uniform intervals between $\frac{1}{R}$ and R, where $N = 5$ and $R = \sqrt{2}$ in our experiments. Merge all the score maps into a final scale-space score map, S, with a softmax-like operation. Define it as :

$$S = \sum_{n} \bar{h}^n \odot \text{softmax}_n(\bar{h}^n) \tag{1}$$

To increase the saliency of keypoints, we perform a differentiable form of non-maximum suppression by applying a softmax operator over 15×15 windows in a convolutional manner, which results in N sharper score maps.

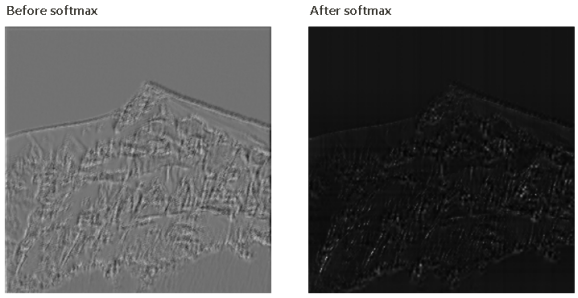

Figure 4: Result of softmax operation.

Then select the k highest score as the result of the keypoint for comparison. We can find that in the picture on the left, the key points are easily selected on the edge of the picture, which will cause some problems during the training process.

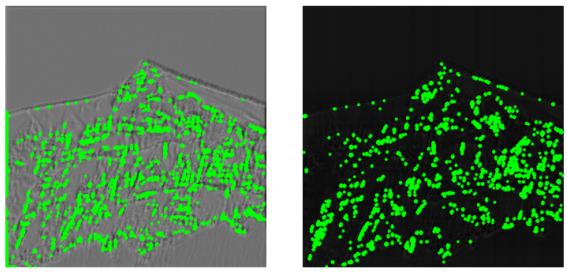

Figure 5: Result of keypoints location.

2.5 Orientation Estimation

Apply a single 5×5 convolution on o which outputs two values for each pixel. They are taken to be sine and cosine of the orientation and used to compute a dense orientation map θ using the arctan function. Train a CNN $\hat{f}w(.)$ to predict two values, which can be seen as a scale cosine and sine, and

Layer	Output Size
Conv2d	$(5,\ H,\ W) \rightarrow (2,\ H,\ W)$

Table 3: Architecture of the OrientationConv module.

compute an angle by taking :

$$fw(p_i^*) = \arctan 2(\hat{f}w^{(1)}(p_i^*), \hat{f}w^{(2)}(p_i^*)) \tag{2}$$

where $\hat{f}w^{(1)}(p_i^*), \hat{f}w^{(2)}(p_i^*)$ are the two values return by the CNN for patch p_i^*, and the $\arctan 2(y, x)$ is the fourquandrant invese tangent function.

2.6 STN

Image patches around the chosen keypoints are cropped with a differentiable sampler (STN) and fed to the descriptor network, which generates a descriptor for each patch.

2.7 Image-level Loss

Although the input of mode j is only the input of rotate model i, Intuitively, after the extraction of model j, the obtained feature only needs to be rotated back to the original angle, and the output of model i can be obtained. However, it can be seen from fig that even if feature j is rotated, the same result as feature i will not be obtained. Therefore, I plan to aim at the minimize distance, and expect that even if the image is rotated, the extracted feature should have rotation invariance.

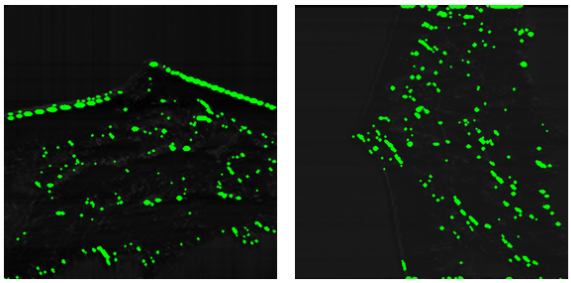

Figure 6: Comparison of feature i and rotated feature j.

$$L_{\text{im}}(S_i, S_j) = |S_i - \text{rotate}(S_j)|^2 \tag{3}$$

2.8 Patch-wise Loss

We then extract descriptors at these corresponding regions p_{k_i} and $p_{\hat{k}_j}$. If a keypoint falls on occluded regions after warping, we drop it from the optimization process. With these corresponding regions and their associated descriptors D_{k_i} and $\hat{D_{k_j}}$, we form L_{pair}, which is used to train the detector network, i.e., the keypoint, orientation, and scale components. Mathematically, we write:

$$L_{\text{pair}}(D_k^i, \hat{D}k^j) = \frac{1}{k} \sum_k |D_k^i - \hat{D}_k^j|^2 \tag{4}$$

2.9 GHH Activation(Generalized Hinging Hyperplane Activation)

To achieve state-of-the-art results with CNNs. This activation function is a generalization of the popular ReLU, maxout , and the recent PReLU activation functions based on Generalized Hinging Hyperplanes (GHH), which is a general form for continuous piece-wise linear functions. As GHH activation function is more general, it has less restrictions in shape, and allows for more flexibility in what a single layer can learn. This activation function plays one of the key roles in our method for obtaining good orientations. For a given layer output $y = [y_{1,1}, y_{1,2}, ..., y_{2,1}, ..., y_{S,M}]$ before activation, we consider the following activation function:

$$o(y) = \sum_{s \in 1,2,...,S} \delta_s \max_{m \in 1,2,...,M} y_{s,m} \tag{5}$$

where,

$$\delta_s = \begin{cases} 1, & \text{if } s \text{ is odd} \\ -1, & \text{otherwise} \end{cases} \tag{6}$$

S and M are meta-parameters controlling the number of planar segments and thus the complexity of the function.

- Maxout activation: $S = 1$
- ReLU activation: $M = 2$ with $y_{s,1} = 0$
- PReLU: $S = 2$, $M = 2$, $y_{s,1} = 0$, and $y_{1,m} = -\alpha_m y_{2,m}$, where α_m is a scalar variable

3 Libraries I Used

I use pytorch to build my deep learning framework, and all the implementations are based on the content of the paper, without referring to the source code. In feature matching and image stitching, for the convenience of experiments, I directly use the opencv library.

- torchvision: Version 0.13.0+cu113
- opencv-python: Version 4.6.0.66

4 Result and Comparison

4.1 Keypoints Detection

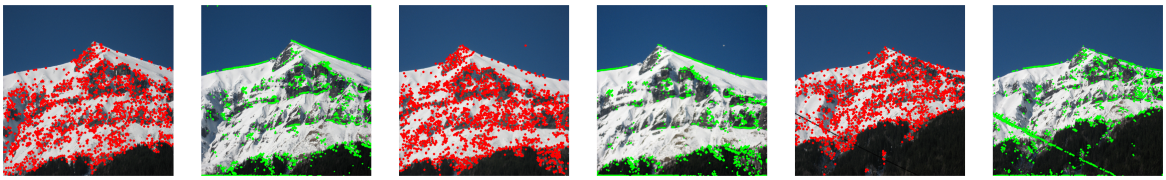

Figure 7: The red points are the feature points obtained by opencv SIFT, and the green points are the feature points obtained by my method.

4.2 Feature Matching

It can be observed that the matching lines of opencv are almost horizontal, mainly because the two images are roughly only horizontally displaced, so the parallel lines appear the result is good. And my matching lines often cross and diagonally, which probably means that there is a problem with the calculation of matching points.

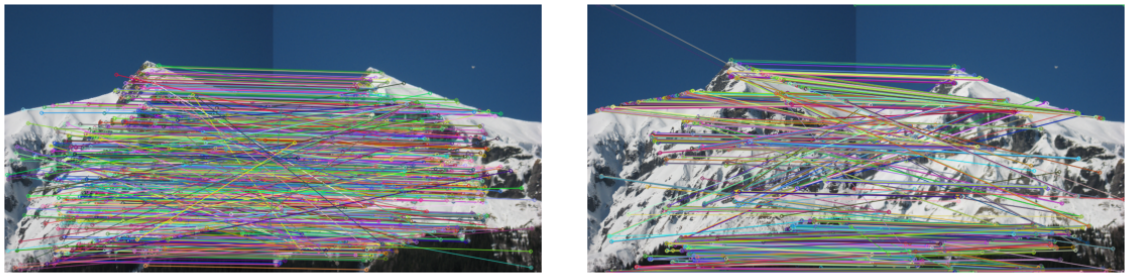

Figure 8: The left side is the feature matching result of opencv; the right side is mine.

5 Conclusion

Using a self-supervised learning method to find key points and descriptors, this method does not require labels, and uses optimization to find the best solution. This sounds like a very good way, but it is a pity that I have not been able to fully reproduce the references method. I think the problem locate in the calculation of orientation. I think the location of the key points found by the detector is reasonable, but in the orientation estimation, there should be something wrong, resulting in the final matching, the input descriptor D is too poor, and then produce bad matching results.

References

[OTFY18] Yuki Ono, Eduard Trulls, Pascal Fua, and Kwang Moo Yi. Lf-net: Learning local features from images. *arXiv preprint arXiv:1805.09662*, 2018. Submitted on 24 May 2018 (v1), last revised 22 Nov 2018 (this version, v2).

[YVFL16] Kwang Moo Yi, Yannick Verdie, Pascal Fua, and Vincent Lepetit. Learning to assign orientations to feature points. *arXiv preprint arXiv:1511.04273*, 2016.