Image Matching and 3D Reconstruction

DATS 6303
Deep Learning - Final Project
Individual Report - Sajan Kumar Kar

I. Introduction

1.1 Overview

Our final project focuses on the challenge of Structure from Motion, where we aim to reconstruct 3D representations from a collection of standard 2D images. The transition from a 2D image to a 3D representation entails several intricate steps:

1. **Keypoint and Descriptor Extraction:** First, we pinpoint specific areas in the images, termed keypoints, which possess unique characteristics unaffected by modifications like size, rotation, or lighting, angles and distnace. These keypoints are then described using numerical codes known as descriptors.
2. **Image Matching:** Subsequently, we establish correspondences between keypoints across different images, facilitating their alignment and matching.
3. **3D Reconstruction:** To transform these matched keypoints into a coherent 3D model, we employ specialized software such as Python's COLMAP library, which synthesizes the spatial arrangement of keypoints to generate a comprehensive 3D representation.

We found our data and inspiration from image matching kaggle competition. The dataset can be found here. The dataset has scenes from various different landmarks.

1.2 Shared Work

The entire project was a group effort of 3 of us. All of us had to learn so many newer concepts and help each other out at various circumstances. Majorly each of us focused more on one or two aspects of the problem.
Tanmay worked on the initial step of getting the image paris using DINOv2 model and also built and deployed the application in streamlit.
I worked on extracting the keypoints and descriptors and achieved it using the ALIKED model. Additionally, I also worked with also with pycolmap for going from 2D to 3D by performing 3D reconstruction.
Parv worked on the image matching and implemented LightGlue model to match keypoints and also helped Tanmay with the Streamlit setup. He also managed the project on GitHub.
All of us worked together in setting up the code structure.

II. Overview of Individual Work

My first task was to understand the general process of 3D reconstruction, the intermediate steps required, the why's and the different existing strategies. After everything was set up, I looked into methods that could work well in our case, architectures that would work well as a feature extractor. Once pairing of images was finished, I implemented the custom architecture for extracting keypoints and descriptors. Unfortunately they didn't provide results as expected. So, with further research, I switched to use the pretrained ALIKED model. This gave good results and after this I proceeded to perform sparse reconstruction.

III. Explanation

1. Research

A huge chunk of my time went into researching about how this problem is solved. Many questions arose in my mind; how do we go from a 2D view to a 3D view? Why do we need to extract keypoints? What methods exist to solve these problems? What methods work well? What can I build on top of these methods? After going through tons of Research papers, Wikipedia and Medium articles, I understood most of the concepts.

2. Custom model for keypoint extraction (Failed)

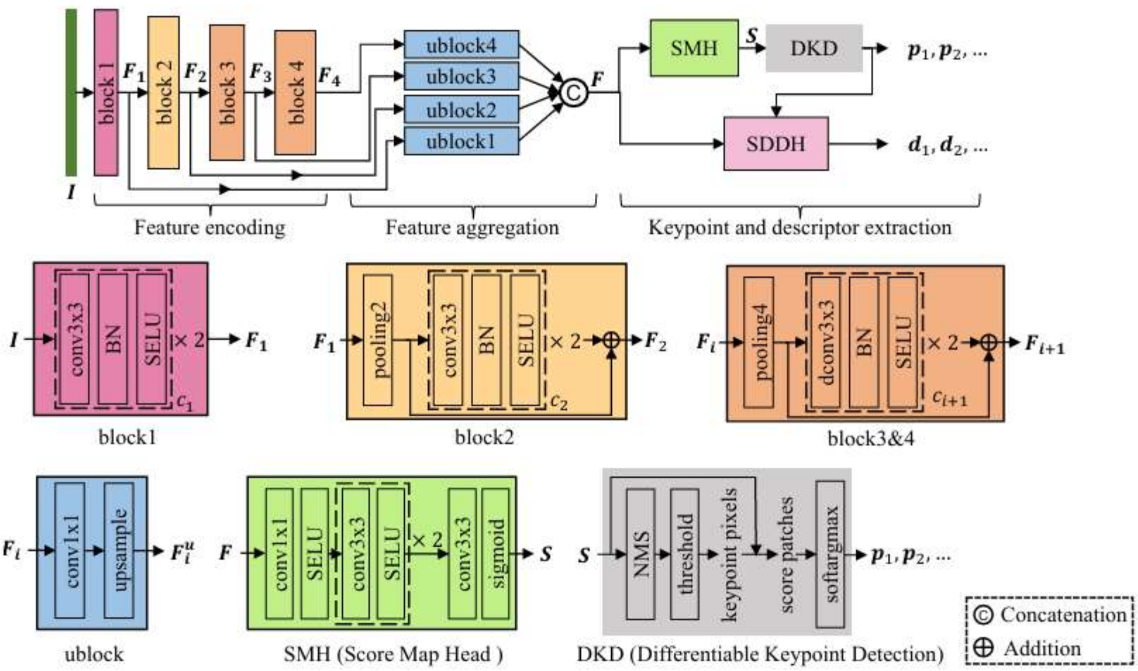
There according to my understanding, It seemed a convolutional feature extractor with simultaneous keypoint and descriptor extraction head should work well and it'll be able to extract keyoints. Below is the setup of the architecture that I designed initially:

```
def conv_block(in_channels, out_channels):
  return nn.Sequential(
      nn.Conv2d(in_channels, in_channels+16, kernel_size=3, padding=1),
      nn.ReLU(inplace=True),
      nn.Conv2d(in_channels+16, out_channels, kernel_size=3, padding=1),
      nn.ReLU(inplace=True),
      nn.MaxPool2d(kernel_size=2, stride=2)
  )

class KeypointDescriptorNet(nn.Module):
  def __init__(self, in_channels, num_keypoints, descriptor_dim):
    super(KeypointDescriptorNet, self).__init__()
    # Feature extraction using the defined conv_block function
    self.block1 = conv_block(in_channels, 64)
    self.block2 = conv_block(64, 128)
    self.block3 = conv_block(128, 196)

    # Separate branches for keypoint and descriptor prediction
    self.keypoint_branch = nn.Sequential(
        nn.Conv2d(196, 64, kernel_size=3, padding=1),
        nn.ReLU(inplace=True),
        nn.Conv2d(64, num_keypoints, kernel_size=1)
    )

    self.descriptor_branch = nn.Sequential(
        nn.Conv2d(196, 196, kernel_size=3, padding=1),
        nn.ReLU(inplace=True),
        nn.Conv2d(196, descriptor_dim, kernel_size=1)
    )
```

The results of keypoints were extremely poor. Initially, I was using the 'wall' images from Heritage dataset and I thought this was working. Then I tried it with the 'bike' images and there's an image of the bike from the back view with a good portion of the image covering the basket. There were rarely any keypoints on the basket which is supposed to be the primary object in the image. I tried increasing the number of blocks in the model till 9, tried adding dropouts and switching Max pooling with Average pooling but nothing seemed to work.
With the time constraint, I decided to use a pretrained model.

3. ***ALIKED***

So initially, I tried to use the superpoint model because my idea in the architecture where at the end we simultaneously calculate the keypoints and the descriptors was something I had learnt after reading the SuperPoint paper. Unfortunately, I was unable to implement Superpoint and then I had to research further where I learned about deformable convolutions.
This made me implement ALIKED. Here is a brief overview of the ALIKED architecture:



The input image I is initially encoded into multi-scale features $\{F_1,F_2,F_3,F_4\}$ with encoding block1 to block4, and the number of channels of F_i is c_i. Then, the multi-scale features are aggregated with upsample blocks (ublock4 to ublock1), and the output features F^u_i are concatenated to obtain the final image feature F. The Score Map Head (SMH) extracts the score map S with F followed by a Differentiable Keypoint Detection (DKD) module [10] to detect the keypoints p_1 p_2. The SDDH then efficiently extracts deformable invariant descriptors at the detected keypoints. "BN", "poolingN", and "DCN3x3" denote batch normalization, NxN average pooling, and 3x3 deformable convolution, respectively.
Then it uses a Sparse Deformable Descriptor Head (SDDH) to learn deformable positions of supporting features for each keypoint and constructs deformable descriptors.

This model gave amazing results, it worked well with everything. Extracted keypoints properly from all the scenes. I tried changing the confidence threshold for this and at 0.01, it gave the best results. Here are some examples of images with their keypoints extracted using the ALIKED model.

4. *Sparse Reconstruction*

After image matching was finished, I looked at the various ways to perform SfM. I founded 3 tools to choose from Colmap, OpenMVG and Kaolin. Both Kaolin and Open MVG had their own methods while colmap had a python integration. So I tried to use pycolmap's Incremental Pipeline which uses triangulation to perform sparse 3D reconstruction. In brief, triangulation refers to the process of determining a point in

3D space given its projections onto two, or more, images. In order to solve this problem it is necessary to know the parameters of the camera projection function from 3D to 2D for the cameras involved.
My initial results for the sparse reconstruction weren't quite up to par, with the Mean Average Accuracy (MAA) for the bike image hovering around 0.001, which appeared disappointingly low despite the reconstruction looking somewhat acceptable. To tackle this issue, I decided to implement RANSAC to mitigate the effects of noisy data and outliers inherent in the keypoint correspondences calculated using feature descriptors. This led to a notable improvement in the MAA, ranging around 0.01, and significantly enhanced the quality of the sparse reconstruction. I also conducted experiments with various hyperparameters, ultimately settling on min_model_size = 3 and max_num_model = 2, while keeping the rest at default settings, as they yielded the best results.
However, my testing was limited to scenes from the Haiper, Heritage, and Urban datasets due to the rotational and translational threshold specifications provided for these scenes only. Despite achieving observable improvements in the reconstruction quality, the persistently low MAA values remained surprising to me, considering the decently comprehensible nature of the sparse reconstructions. Below are the values for 3 different scenes from 3 datasets:

Dataset Type	Scene	MAA
Urban	Kyiv Puppet Theatre	0.002
Haiper	Bike	0.01
Heritage	Dioscuri	0.0019

I tried to improve the reconstruction process, so I went ahead to implement dense reconstruction. Unfortunately, at the Stereo Fusion stage I was receiving CUDA integration errors and was not able to figure out what was going wrong. Hence I had to size it down and stop at Sparse reconstruction. (Link to view the 3D sparse reconstruction results)

5. *Summary and Conclusion*

Throughout the entire process I learnt a lot about 3D vision problem and how they differ from and offer an additional set of challenges to 2D vision problems. These include the limitations of traditional convolution, solutions for that, deformable convolutions. By reading about superpoint, I learnt how self-supervised learning can be implemented to generate labels, the concept of joint training of interest points and descriptors. I also learnt that outliers in the data could lead to a massive underperformance.
A lot more could be done with this to improve the models. Different algorithms like SuperPoint instead of ALIKED or even SIFT could be tried. I could spend time to think about what can be done to preprocess the data. For instance, one thing we noticed is that some of the images in a particular scene are rotated. Maybe orienting them to have the same rotational axis would help the model improve. There exists different algorithms for image reconstruction. I could try modifying or tuning the exisiting SfM

algorithms to yield better results. Dense reconstruction through MVS could be implemented to get a better 3D reconstruction.

IV. Summary and Conclusion:

Throughout the entire process I learnt a lot about 3D vision problem and how they differ from and offer an additional set of challenges to 2D vision problems. These include the limitations of traditional convolution, solutions for that, deformable convolutions. By reading about superpoint, I learnt how self-supervised learning can be implemented to generate labels, the concept of joint training of interest points and descriptors. I also learnt that outliers in the data could lead to a massive underperformance.
A lot more could be done with this to improve the models. Different algorithms like SuperPoint instead of ALIKED or even SIFT could be tried. I could spend time to think about what can be done to preprocess the data. For instance, one thing we noticed is that some of the images in a particular scene are rotated. Maybe orienting them to have the same rotational axis would help the model improve. There exist different algorithms for image reconstruction. I could try modifying or tuning the exisiting SfM algorithms to yield better results. Dense reconstruction through MVS could be implemented to get a better 3D reconstruction.

Overall, this has been the most interesting and challenging project I have ever worked with. There is a lot more to learn and try and I have yet to properly understand the nuances of 3D vision which I plan to explore furthur.

VI. Percentage Code: 56.6%

VII. References

1. P. Lindenberger, P. -E. Sarlin and M. Pollefeys, "LightGlue: Local Feature Matching at Light Speed," 2023 IEEE/CVF International Conference on Computer Vision (ICCV), Paris, France, 2023, pp. 17581-17592, doi: 10.1109/ICCV51070.2023.01616.
(https://ieeexplore.ieee.org/document/10377620)
2. Oquab, Maxime & Darcet, Timothée & Moutakanni, Théo & Vo, Huy & Szafraniec, Marc & Khalidov, Vasil & Fernandez, Pierre & Haziza, Daniel & Massa, Francisco & El-Nouby, Alaaeldin & Assran, Mahmoud & Ballas, Nicolas & Galuba, Wojciech & Howes, Russell & Huang, Po-Yao & Li, Shang-Wen & Misra, Ishan & Rabbat, Michael & Sharma, Vasu & Bojanowski, Piotr. (2023). DINOv2: Learning Robust Visual Features without Supervision.
(https://www.researchgate.net/publication/370058767_DINOv2_Learning_Robust_Visual_Features_without_Supervision)

3. J. Dai et al., "Deformable Convolutional Networks," 2017 IEEE International Conference on Computer Vision (ICCV), Venice, Italy, 2017, pp. 764-773, doi: 10.1109/ICCV.2017.89.

keywords: {Convolution;Kernel;Object detection;Standards;Feature extraction;Two dimensional displays},
(https://ieeexplore.ieee.org/document/8237351)

4. Zhao, Xiaoming & Wu, Xingming & Chen, Weihai & Chen, Peter C. Y. & Xu, Qingsong & Li, Z.G.. (2023). ALIKED: A Lighter Keypoint and Descriptor Extraction Network via Deformable Transformation.
(https://arxiv.org/abs/2304.03608)

5. (https://www.kaggle.com/code/asarvazyan/imc-understanding-the-baseline#Sparse-Reconstruction)
6. Paul-Edouard Sarlin, Daniel DeTone, Tomasz Malisiewicz, and Andrew Rabinovich. SuperGlue: Learning feature match ing with graph neural networks. In CVPR, 2020. 1, 2, 6, 8, 11, 12, 14, 15
(https://arxiv.org/abs/1911.11763)