Image Matching and 3D Reconstruction

DATS 6303

Deep Learning-Final Project **Individual Report**

- **Parv Bhargava**

I. Introduction

1.1 Overview

For our final project on image matching, we worked on creating 3D reconstruction from multiple 2D photos. This is useful in areas like augmented reality and 3D modeling. The process involves several steps:

1. **Keypoint and Descriptor Extraction:** Find specific spots in the image that are unique and don't change with image modifications like size, rotation, or lighting. Describe these spots with a numerical code.
2. **Keypoint Matching:** Connect these spots across different images to match them up.
3. **3D Reconstruction:** Use software, like Python's COLMAP library, to turn these matched points into a 3D model.

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. All image files of 12.3 GB.

1.2 Shared Work

Tanmay worked on **DINOv2** model for image creating image pairs and **streamlit application development.**

Sajan worked on **ALIKED** model for **extracting keypoints** and also with pycolmap for reconstruction.

I **(Parv)** worked on the **LightGLUE** model to match keypoints and also helped Tanmay with the Streamlit setup. Additionally, I managed the project on GitHub and wrote bash scripts to set up EC2, ensuring our environment could be recreated. I also refactored the code.

II. Overview of Individual Work

I started by reviewing code from Kaggle competition notebooks and researching how deep learning could be applied to converting 2D images into 3D. I also looked into how this was done without using deep learning. After my research, I focused on image matching, where I received keypoints and descriptors from Sajan to perform the matching. During my research, I encountered various complex model architectures that had been developed over many years. Initially, we considered creating our own models, but due to the complexity and time constraints, we decided to use pre-trained models instead.

For matching image pairs, I did a thorough review of the literature to determine the best models for our needs, and we chose ALIKED and LightGLUE. LightGLUE, which builds upon SuperGLUE, is more computationally efficient, offering a balance between accuracy and processing speed. I found that the initial settings for the model worked best after testing with different settings (Here I would like to add I tested the thresholds on data which limited as it was taking a lot of time to generate rotational matrix and translation vector).

I also worked on a Streamlit application, focusing on the backend to provide the necessary functions for various operations.

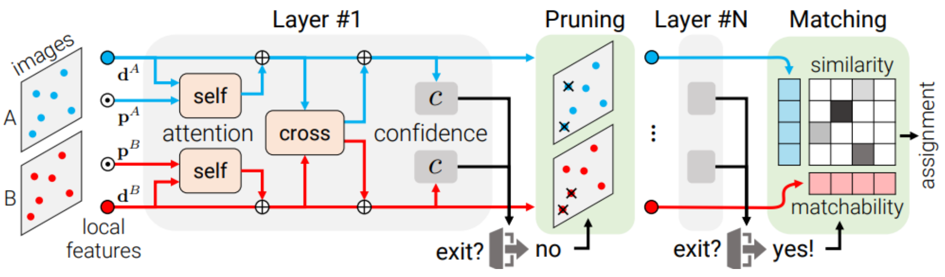
Additionally, I managed the project, wrote bash scripts to set up the environment properly (like loading model weights in PyTorch), and ensured that everyone could test their models without issues. (I learned a lot about loading pre-trained models weights into pytorch)

III. Explanation

1. *Litrature review*

I researched the 2023 Image Matching Competition and discovered two deep learning approaches: SuperGLUE and LightGLUE. After reviewing both models, I concluded that LightGLUE performs better and is less computationally expensive compared to SuperGLUE. I will explain the light glue model now. I have not included the explanation for superGLUE as we did not used that model.

2. *LightGlue Model*



- **Input**
  LightGlue receives features from two images (image A and image B). Each feature has:

  Keypoints (p_i): Normalized (x, y) coordinates of the feature in the image.
  Visual descriptor (d_i): A vector that describes the feature's appearance.

- **Initial State**
  The features' visual descriptors set the initial state for processing. This state is used to compare and refine the features through a series of steps.

- **Transformer Backbone**
  The main processing in LightGlue happens through a series of self-attention and cross-attention mechanisms:

Self-Attention: Refines each feature by considering all other features in the same image.
Cross-Attention: Updates feature information by incorporating relevant data from the other image, helping align features across both images.

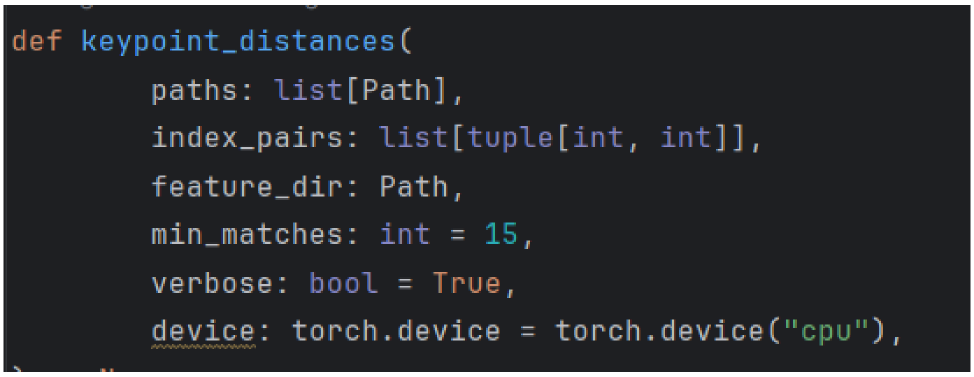
- **Adaptive Mechanisms**
  To improve efficiency, LightGlue uses:

  Adaptive Depth: The process stops early if the confidence level is high enough.
  Point Pruning: Removes features unlikely to match, reducing computational load.

- **Matching Features**
  Assignment Scores: Calculates how likely features from both images match.
  Matchability Scores: Estimates if a feature is likely to find a match in the other image.
- **Output**
  Produces a partial assignment matrix: By combining the assignment scores with the matchability scores, the matrix is created. Each element in this matrix represents the probability of a feature from image A matching with a feature from image B. These are the matches we need for 3D reconstruction.

3. ***Streamlit and Enviroment Setup***
   I developed backend code for running the model in the Streamlit app My work includes creating functions for inferencing from models and also integrating the backend together to make sure everything runs smoothly and helped Tanmay with refactoring the code. Additionally, I created bash scripts for getting data from kaggle and setting up the environment, ensuring all model weights are loaded correctly.

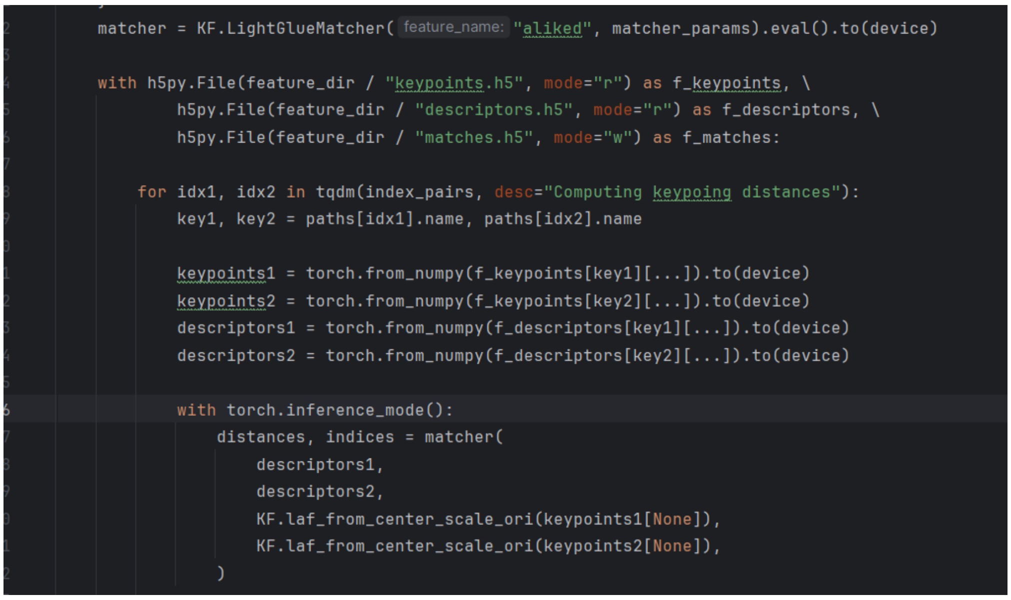
4. ***Code Explanation for LightGLUE:***
   The function keypoint_distances calculate and stores the distances between keypoints of image pairs. It takes lists of image paths and index pairs, specifying which images to compare, and uses directories for the received data. (min_matches here is minimum number of matches required to consider the comparison valid)



```
def keypoint_distances(
        paths: list[Path],
        index_pairs: list[tuple[int, int]],
        feature_dir: Path,
        min_matches: int = 15,
        verbose: bool = True,
        device: torch.device = torch.device("cpu"),
```

Fig 1: The keypoint_distance function

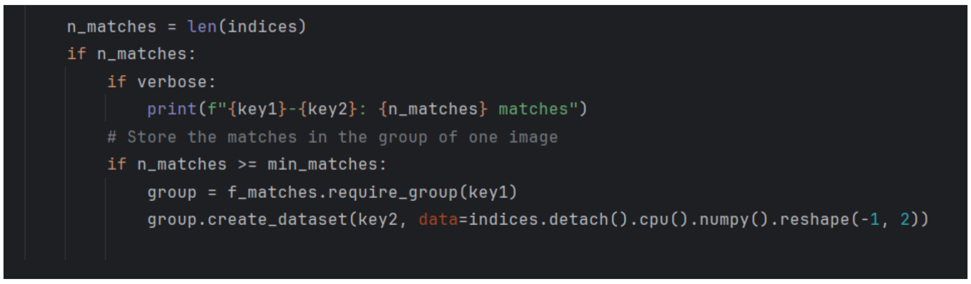
The function uses a LightGlue as matcher to compare keypoints and descriptors, which are then stored matches.h5.



```
matcher = KF.LightGlueMatcher( feature_name: "aliked", matcher_params).eval().to(device)

with h5py.File(feature_dir / "keypoints.h5", mode="r") as f_keypoints, \
        h5py.File(feature_dir / "descriptors.h5", mode="r") as f_descriptors, \
        h5py.File(feature_dir / "matches.h5", mode="w") as f_matches:

    for idx1, idx2 in tqdm(index_pairs, desc="Computing keypoing distances"):
        key1, key2 = paths[idx1].name, paths[idx2].name

        keypoints1 = torch.from_numpy(f_keypoints[key1][...]).to(device)
        keypoints2 = torch.from_numpy(f_keypoints[key2][...]).to(device)
        descriptors1 = torch.from_numpy(f_descriptors[key1][...]).to(device)
        descriptors2 = torch.from_numpy(f_descriptors[key2][...]).to(device)

        with torch.inference_mode():
            distances, indices = matcher(
                descriptors1,
                descriptors2,
                KF.laf_from_center_scale_ori(keypoints1[None]),
                KF.laf_from_center_scale_ori(keypoints2[None]),
            )
```

Fig 1: LightGLUE implemenatation for getting matches

If the number of matches between two images exceeds a minimum threshold, these matches are saved in an output file.



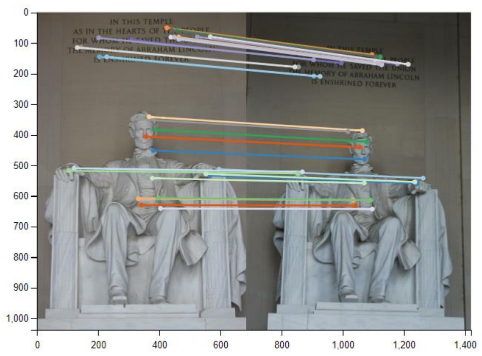
```
n_matches = len(indices)
if n_matches:
    if verbose:
        print(f"{key1}-{key2}: {n_matches} matches")
    # Store the matches in the group of one image
    if n_matches >= min_matches:
        group = f_matches.require_group(key1)
        group.create_dataset(key2, data=indices.detach().cpu().numpy().reshape(-1, 2))
```

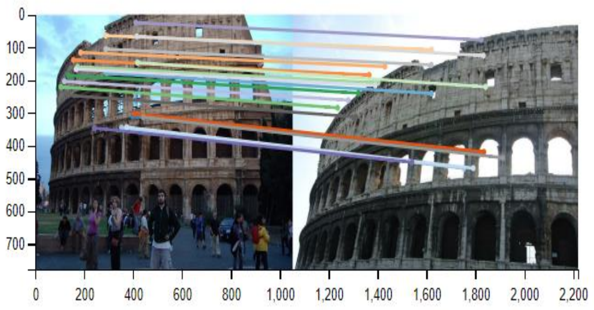
Fig 1: Thresholding the output.

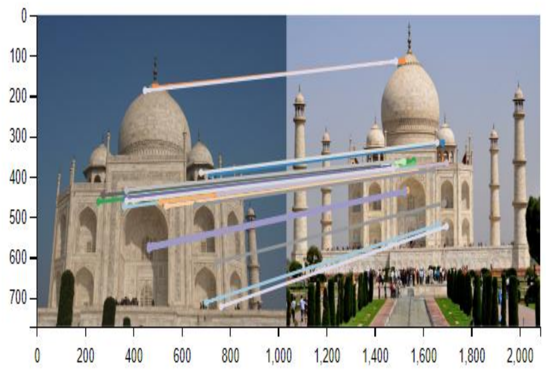
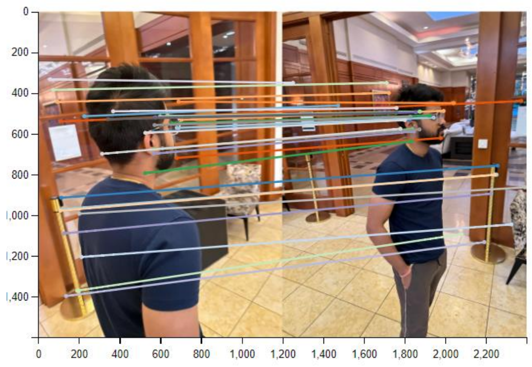
IV. Results

I was successfully able to match the images using LightGlue model as the matcher and from those matches, we were able to make a 3D reconstruction.

These are some results after performing image matching.





We also played around a bit and found out if we gave different images (example of different people or scenes) there were still matches found. We know that was going to happen as model always tries to find the pattern. This could be a point for furture research on how to make matcher diffrentitate between two different scenes. To tackle this we can go for the clustering approach during preprocessing which we implement in image classification problems to remove such images.

For measuring performace sajan wrote a script to compare mAA values for rotational matrix and translation vector to check how close we are to mimic classical sfm performace from the data we downloaded.

Also towards the end we were able to get final reconstructions from 2D images, Though we used pre-trained models it was a great learninig experience.

V. Summary and Conclusion:

I learned a lot from tackling the tough challenge of recreating a 3D image from a 2D image. Initially, we felt overwhelmed, but we managed to develop a functional app. This project was exciting and encouraged me to delve into various research papers for model architectures.

One standout concept was from the LightGLUE paper was, which introduced a confidence classifier using an MLP (Multilayer Perceptron). It was a clever approach to employ MLP as a gating mechanism with confidence assessment, which was new to me and gave me so many ideas to use this in networks to make an network more computationally efficient. This is the equation for confidence classifier I was talking about from LightGLUE paper:

$$c_i = \mathrm{Sigmoid}\left(\mathrm{MLP}(\mathbf{x}_i)\right) \in [0,1] \ .$$

We mainly fine-tune the parameters and run inference on pre-trained models. We considered designing our own model architecture and initially tried and got extremely poor results which was not able to detect keypoints properly. We realized it would require more time to develop an sound architecture for this problem statement, so we sized it down to use pre-trained model instead of training our own. But we were able to learn a lot from this process as we read through papers for different architectures and implementations.

I also learned how to perform inference on pre-trained models in pytorch and to load weights, for coding part I referred multiple kaggle notebooks and also took some utils scripts for database from internet. I have included the licence for scripts in the codes.

For the future, I'm thinking of doing a more thorough read of the papers again to get a better understanding of these models. I also want to perform dense reconstruction, which we tried towards the end but were not able to achieve. Additionally, I plan to start creating my own architecture, which might not be as advanced as these models but will be a good start.

Overall, I thoroughly enjoyed the project and am eager to continue working on this project.

VI. Percentage Code: 71.6%

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. Link

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. Link

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. Link

4. Sarlin, P.-E., DeTone, D., Malisiewicz, T., & Rabinovich, A. (2020). SuperGlue: Learning Feature Matching with Graph Neural Networks. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). Link

5. https://github.com/cvg/LightGlue

6. https://www.kaggle.com/code/talhareyaz/3d-reconstruction-sfm-colmap

7. https://www.kaggle.com/code/asarvazyan/imc-understanding-the-baseline#Sparse-Reconstruction

8. https://www.kaggle.com/code/ruslanlvovich/image-reconstruction-pytorch?rvi=1

9. https://www.kaggle.com/code/mukit0/superpoint-imc2023-2nd-position#Setup-SuperPoint-API-and-Configuration