Group – 2: 3D Reconstruction

I. Introduction:

In today's digital age, the accessibility of photography has been greatly expanded, with many individuals wielding powerful cameras right in their pockets. This makes it easier for individuals to capture snapshots of landmarks and share them with others. However, these two-dimensional images offer limited perspectives, typically reflecting only the viewpoint of the photographer. Yet, with the vast repository of images available online, there lies a tantalizing opportunity to harness these collective visual data to construct comprehensive, three-dimensional models of various subjects. This endeavor, known as Structure from Motion (SfM), entails synthesizing 3D representations from a multitude of diverse images. Our project aims to tackle the challenge of reconstructing 3D models from a set of 2D images of a scene consisting of images from different angles, distances, noise, etc.

The process of 3D image reconstruction follows 3 major steps in our project. Once the images of a particular scene are paired against each other, keypoints are extracted from all the images. After that, corresponding keypoints are matched between pairs of images. This is known as image matching. Then the process of reconstruction starts where we go from 2D images to a 3D view of the image. This is known as Structure from Motion (SfM) or sparse reconstruction.

For quick reference, section II ahead explains our data. Section III talks about the pre-implementation ideas and plans and tried experiments. Section IV talks about the different results of the ideas we were about to implement. Section V summarizes the results obtained. The last section included references to the sources used for this project.

II. Dataset Description:

The data has been sourced from Kaggle from the IMC2023 competition. It consists of about 12GB of images divided by sub-datasets and then by various scenes, with the number of images per scene varying from less than 10 to around 250. The various images in a scene include several images from different angles, different distances, with different viewpoints with noise as well. There is a train_labels.csv file with the following info:

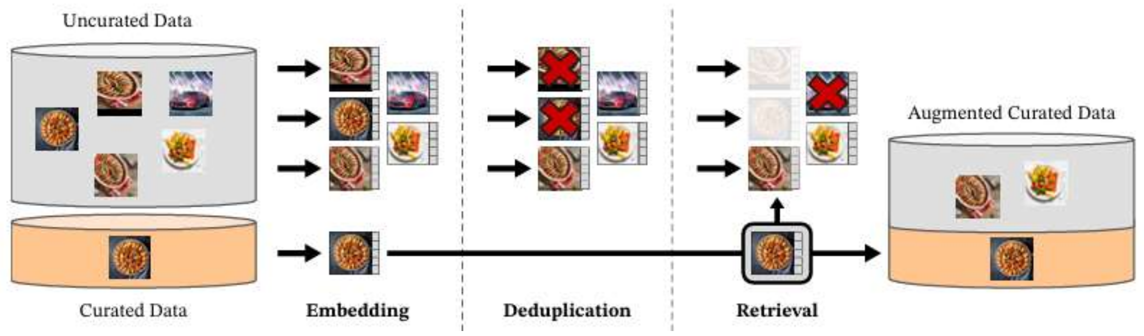
- dataset: The unique identifier for the dataset.
- scene: The unique identifier for the scene.
- image_path: The image filename, including the path.
- rotation_matrix: The first target column. A 3×33×3 matrix, flattened into a vector in row-major convection, with values separated by ;.
- translation_vector: The second target column. A 3-D dimensional vector, with values separated by ;.

There is also a sfm folder which contains the ground truth 3D reconstruction for each of the scenes in each of the dataset.

III. Algorithms used:

Three major algorithms/models were used in our project. In the order of usage they are:

DinoV2: It is based on a self-supervised learning approach which means it needs no labels. It learns directly from the image. Thus, can capture better features.



DINOv2's Vision Transformer Architecture is as follows:

1. The input image is first divided into fixed-size patches, which are then linearly embedded. Positional encodings are added to these embeddings to preserve information about the relative positions of the patches.

2. The embeddings pass through multiple layers of the transformer encoder, each consisting of multi-headed self-attention and position-wise fully connected layers.

3. DINO uses a teacher-student setup where both models are identical in architecture but differ in their parameter update dynamics. The student model tries to predict the output of the teacher.

Then it matches the images to form pairs. The process is as follows:

1. Extract dense embeddings from images using the pretrained Vision Transformer.

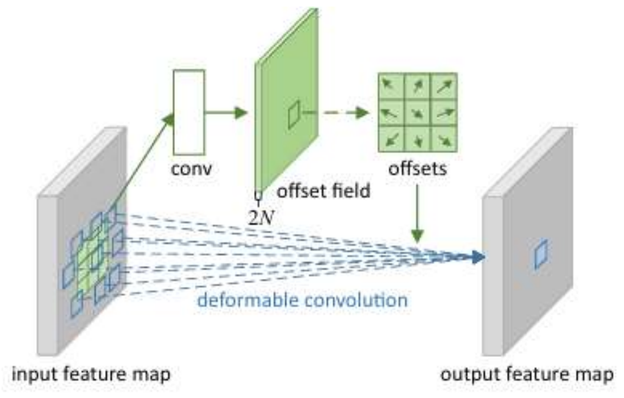
2. Feature Normalization: To ensure embeddings have consistent scale, crucial for reliable similarity measurements.

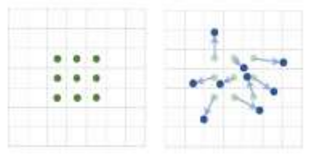
3. Efficient Pairwise Distances: Calculating distances between embeddings, facilitating the identification of similar images.

4. Select image pairs based on a similarity threshold.

ALIKED: A Lighter Keypoint and Descriptor Extraction Network via Deformable Transformation: After we receive the image pairs it is time to extract the keypoints and descriptors. For this we use the ALIKED model.

Deformable Convolution – A modified convolution that adds 2D offsets to the regular grid sampling locations of a standard convolution.





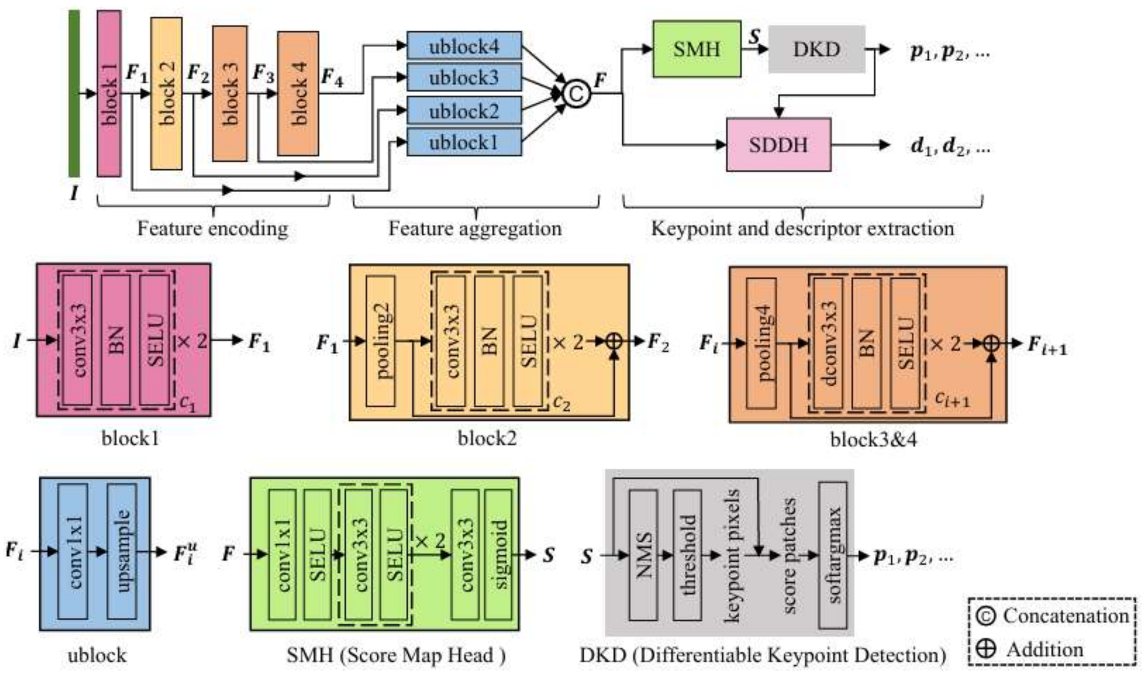
The 2D convolution consists of two steps:

1. sampling using a regular grid over the input feature map
2. Summation of sampled values weighted by W.

In deformable convolution the regular grid is augmented with offsets.

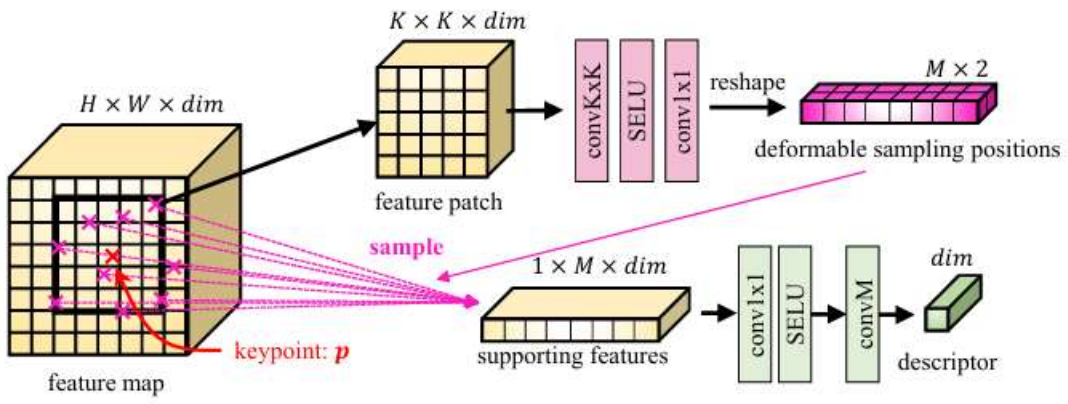
Why deformable convolutions?

Traditionally, DNNs were used to extract descriptors of image patches at predefined keypoints. The conventional convolution operations lack to provide the geometric invariance required for the descriptor. The deformable convolutional network can model any geometric transformation by adjusting the offset for each pixel in the convolution.



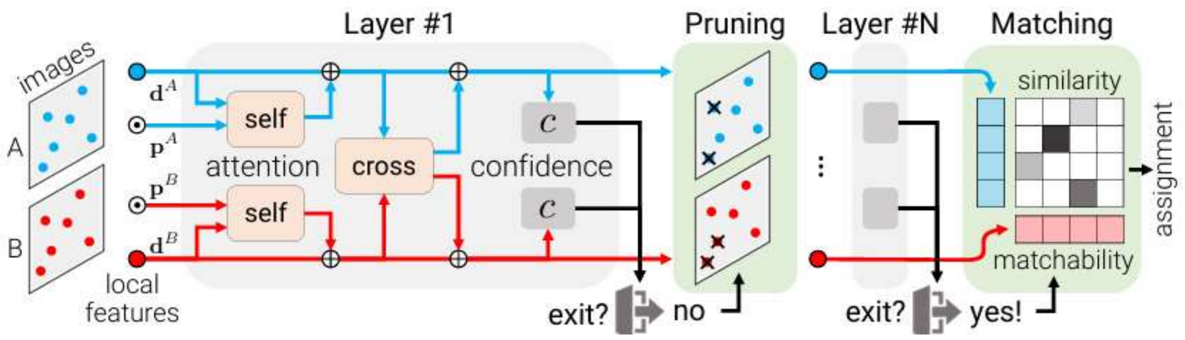
ALIKED used a Sparse Deformable Descriptor Head (SDDH) to learn deformable positions of supporting features for each keypoint and constructs deformable descriptors.

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.



The SDDH estimates M deformable sample positions on KxK keypoint feature patches (K=5 in this example), samples M supporting features on the feature map based on the deformable sample positions, encodes the supporting features, and aggregates them with convM for descriptor extraction.

LightGlue: Based on the SuperGlue architecture LightGlue helps in matching the keypoints across the set of images and builds upon Superglue to provide efficient and faster results.



The model begins by receiving inputs consisting of two sets of local features from two images (referred to as image A and image B). Each local feature in these sets comprises:

1. 2D point location (`p_i`): The coordinates (x, y) of the feature in its respective image, normalized by the image dimensions to fall between 0 and 1. (The keypoints)
2. Visual descriptor (`d_i`): A high-dimensional vector extracted using a feature detector and descriptor (ALIKED in our case), which encodes the appearance around the feature point, allowing for a robust comparison across different views. (The descriptors)

Each feature's visual descriptor initializes its corresponding state vector in the model. This state vector (`x_i`) is what the Transformer architecture will manipulate through its layers to refine and compare feature information between the two images.

Then comes the transformer backbone, which is where the primary computation of LightGlue occurs, involving layers of self-attention and cross-attention:

Self-Attention Mechanism: Goal - To refine each feature's representation by aggregating contextual information from all other features within the same image.

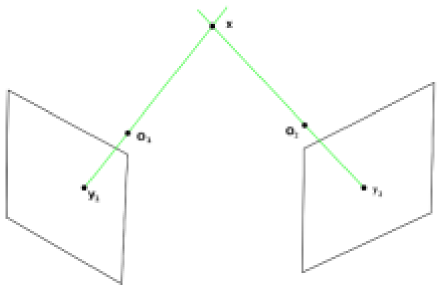
Operation - Each feature generates query (q) and key (k) vectors through trainable transformations of its state vector. The attention scores between all pairs of features within the image are calculated based on the dot product of queries and keys. The feature states are updated by aggregating (via a weighted sum) the states of all other features in the image, weighted by the softmax-normalized attention scores.

Cross-Attention Mechanism: Goal - To enhance the feature states by incorporating relevant information from corresponding features in the opposite image, essentially aligning features across images.

Operation - Each feature in image A generates a key, and each feature in image B uses these keys to calculate cross-image attention scores, and vice versa. Similar to self-attention, the states are updated based on these cross-attention scores, allowing features in one image to pull information from the other.

Then it employs adaptive depth and point pruning strategies to enhance efficiency. After the Transformer layers process the features: The model computes a pairwise score for each pair of features across the two images, based on the similarity of their updated state vectors (Assignment Scores). Each feature receives a score indicating its likelihood of having a match in the opposite image, which is used to weigh the pairwise scores (Matchability Scores). The final output is a soft partial assignment matrix, where each element represents the probability of a feature in image A matching with a feature in image B. High values in this matrix indicate likely matches, from which correspondences are selected.

After all this, we have matched image pairs for the different images in a scene. Now to go from 2D to 3D a concept of triangulation comes into picture.



How do we go from 2D to 3D?

- An image is a projection of a 3D space onto a 2D plane
- Each point in an image, hence, corresponds to a line in 3D space.
- All points on the line in 3D are projected to that singular point in the image.
- If a pair of corresponding points in two, or more images, can be found it must be the case that they are the projection of a common 3D point **x.**
- The set of lines generated by the image points must intersect at **x.**

IV. Experimental setup:

The experimental setup phase of our project was crucial, as it involved laying the foundation for our 3D reconstruction endeavour. Much time and effort were dedicated to comprehensively understanding the problem statement and the intricate process of reconstructing a three-dimensional representation from a given set of images.

Our initial focus was on dissecting the key intermediate steps involved in this process. Central to this understanding was the extraction of keypoints from the images, which serve as pivotal reference points for subsequent image matching and reconstruction tasks. We delved into various methodologies for keypoint extraction. The 1st idea was to try implementing a custom architecture of a feature extractor to get the keypoints and the feature descriptors.

Subsequently, our strategy encompassed leveraging existing algorithms or models for image matching, with a keen emphasis on fine-tuning these approaches to suit our project objectives.

Finally, armed with the extracted keypoints and refined image matching techniques, we aimed to proceed with the 3D reconstruction phase. Our plan involved exploring established libraries and frameworks such as Colmap, OpenMVG, or NVIDIA's Kaolin, to facilitate the generation of a sparse 3D representation from the input image data.

Throughout this experimental setup phase, our focus remained on understanding the idea behind 3D reconstruction, research, experimental planning, and adaptation to ensure that our approach achieves our ultimate goal of three-dimensional reconstruction from image data in a manner where one can use their own set of images to reconstruct a 3D view.

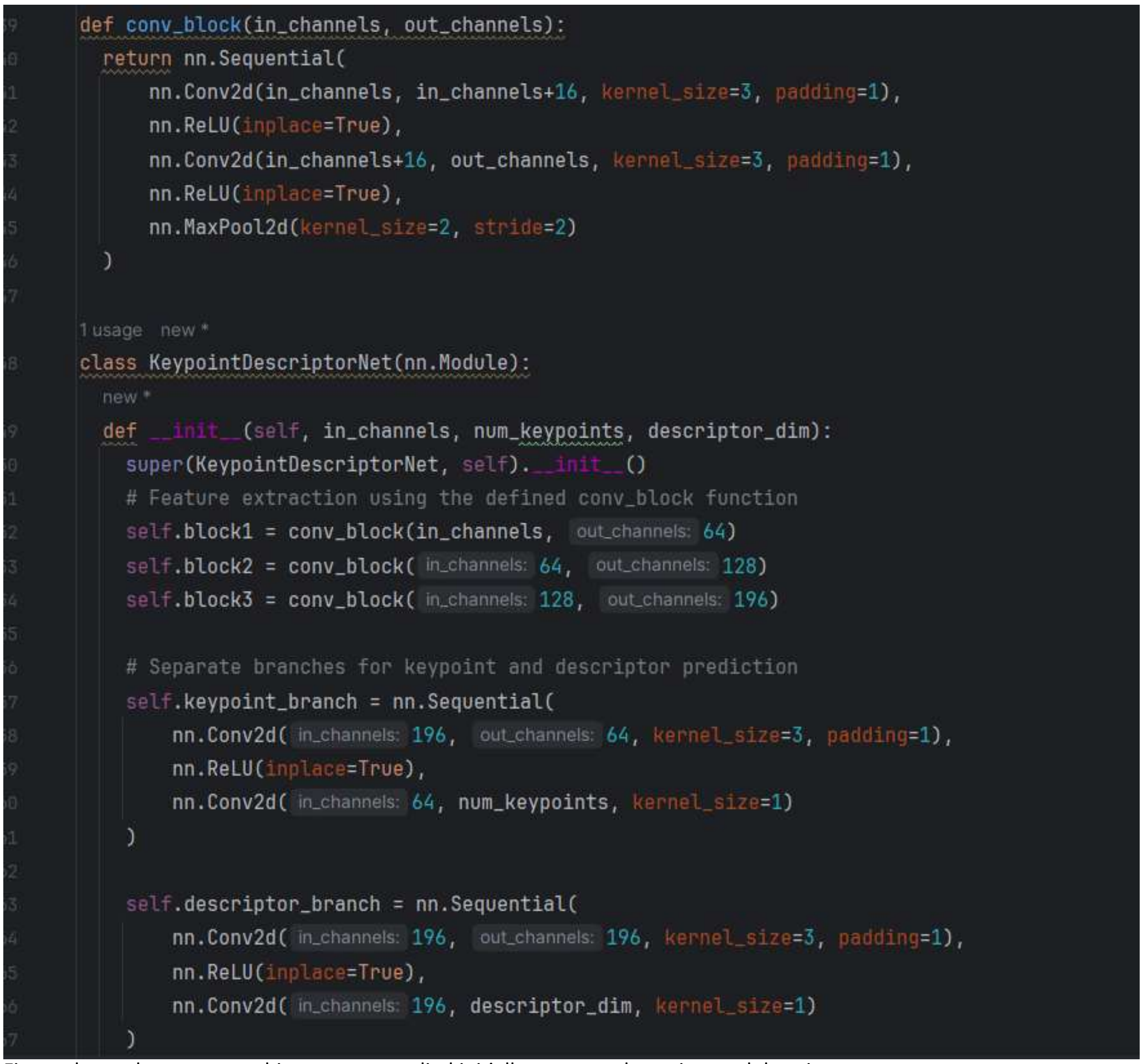
V. Results:

In the initial phase of our project, a critical task was to establish a robust method for obtaining pairs of matching images. This step is pivotal as the accuracy of reconstructing landmarks or scenes hinges on analyzing images that exhibit visual similarity. For instance, when reconstructing the Lincoln Memorial, it's imperative to extract keypoints and descriptors exclusively from images featuring the Lincoln Memorial itself, rather than inadvertently including unrelated images like those of Mars, which could compromise the precision of the reconstruction process.

To tackle this challenge, we did extensive exploration of various resources, including Kaggle Notebooks and research papers, to investigate potential methodologies for image matching. Among the techniques that surfaced prominently were SIFT (Scale-Invariant Feature Transform) and DELF (DEep Local Features). These established methods provided valuable insights into traditional approaches for matching images based on distinctive features.

Furthermore, we delved into the realm of deep learning by studying Meta's Vision Transformer DINO and closely examining their research paper. DINO introduces a novel approach to obtain normalized image embeddings using a transformer architecture. Notably, DINO employs a student-teacher network paradigm, wherein the student model endeavors to emulate the output of the teacher model, thereby facilitating the extraction of meaningful image representations. This is what we ended up using.

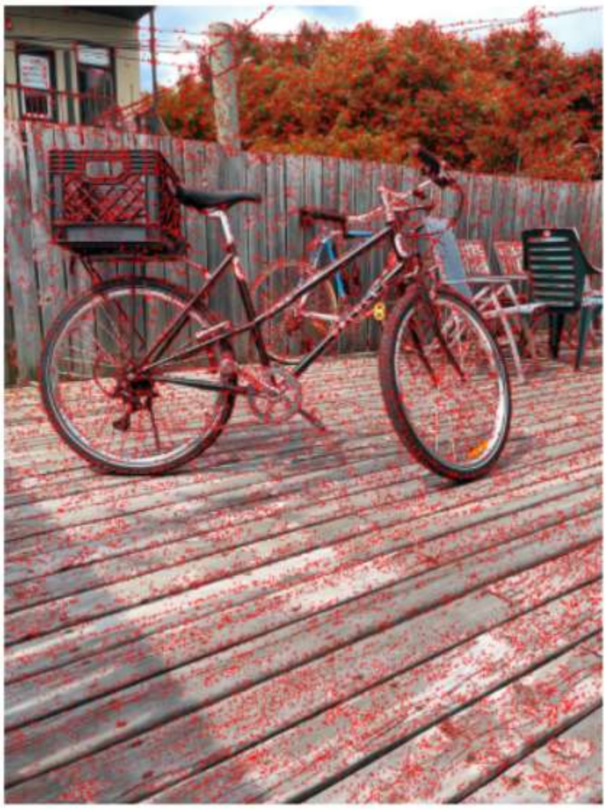
After meticulously designing and experimenting with a custom model architecture for keypoint and descriptor extraction, we encountered significant challenges with the performance of our designed architecture. Despite our initial optimism, the results yielded by our custom model were far from satisfactory. We noticed that the keypoints that it extracted were completely outlandish. For example, in the bike images, there were no keypoints detected for the model of the bike except for the basket. Initially when we were trying on the wall image, we were getting good results but after trying on a few other images we realized that our architecture isn't working at all. We tried increasing the complexity of our architecture by adding more layers but it still wouldn't provide any good results.



```
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)
  )

1 usage   new *
class KeypointDescriptorNet(nn.Module):
  new *
  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,  out_channels: 64)
    self.block2 = conv_block( in_channels: 64,  out_channels: 128)
    self.block3 = conv_block( in_channels: 128,  out_channels: 196)

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

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

Figure shows the custom architecture we applied initially to extract keypoints and descriptors.

Recognizing the limitations of our approach, we made the decision to pivot towards leveraging pre-trained models to expedite our progress. While our initial attempts with models like Superpoint encountered implementation challenges, our exploration of ALIKED proved to be remarkably fruitful. We tried different confidence thresholds for the model and a threshold of 0.01 provided us with the best results.



The above image shows the output we obtained from our ALIKED model. As can be observed, it is able to extract the keypoints accurately, not only from the main object (the bike) but also from the surrounding areas, which would help in reconstructing the surrounding down the lane. Some more results of keypoint extraction are provided below.

Post this we had to match the keypoints that we extracted using ALIKED. After some more research we decided to use Lightglue for this and play around with the different parameters like width confidence and depth confidence of the model to get the best matches. But the pre-stated parameters gave us the best results. Below are some examples of the results we got for image matching.

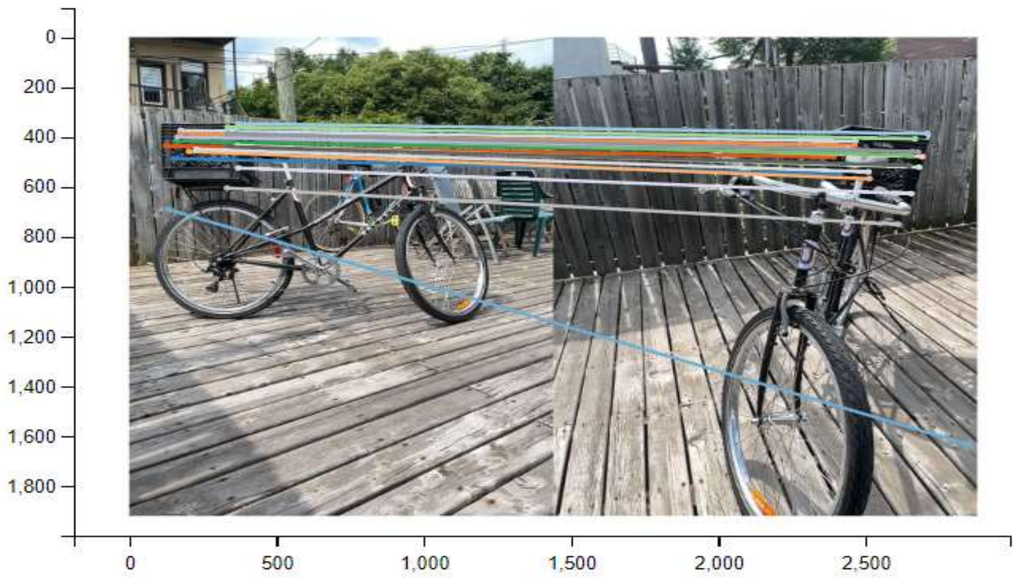
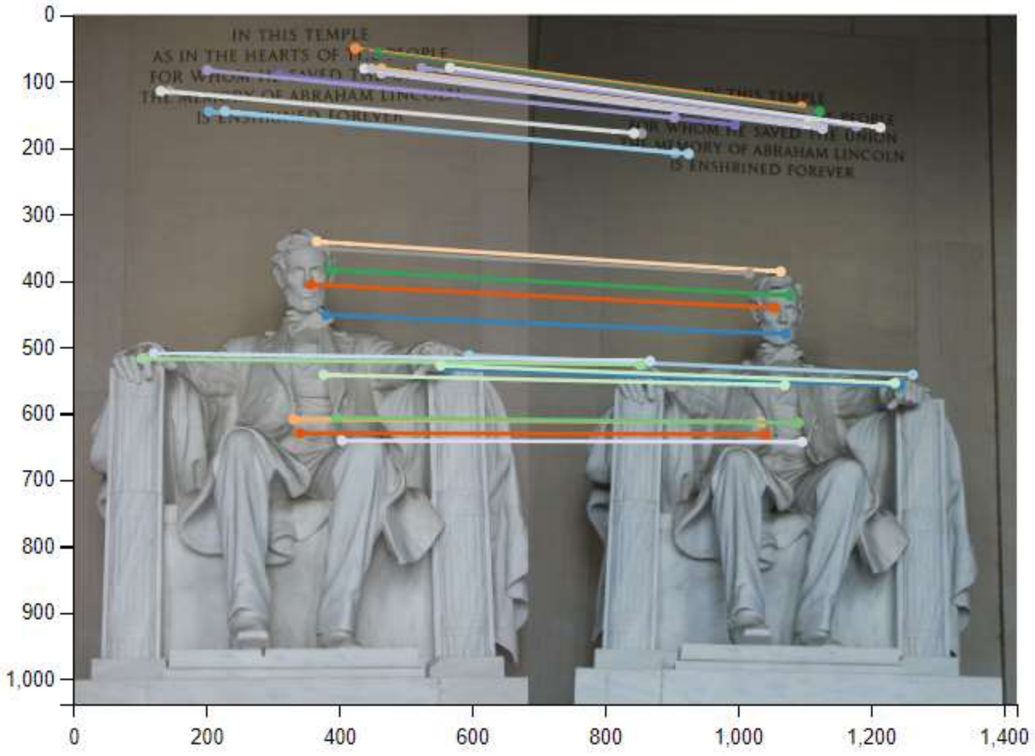
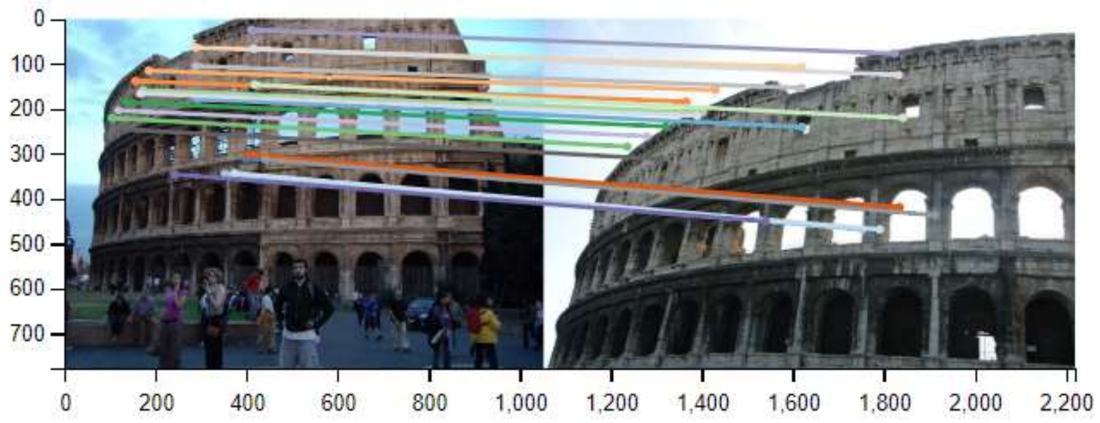


Image matching of bike. (These represent matching of only 20 keypoints for simplicity of the visual. Originially, there are matches of all the 4096 keypoints we have extracted.)

Here are some more results of the image matching pairs for Lincoln Memorial and Colloseum.





After image matching, we looked at what could be a better way to perform SfM. We had 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 we tried to use pycolmap's Incremental Pipeline which uses triangulation to perform sparse 3D reconstruction.

Link to view the 3D sparse reconstruction results

Initial results were not bad but not very good either. All these sparse reconstruction results were measured in MAA (Mean Average accuracy). For the bike image, we got an initial MAA of 0.001 which seems extremely poor but the reconstructed view wasn't as bad. To improve the accuracy, we implemented RANSAC. Since the keypoint correspondences are computed using feature descriptors, the data is bound to be noisy and (in general) contains several outliers. Thus, to remove these outliers we use RANSAC which resulted in an improved MAA in the range of 0.01 and even the sparse reconstruction was considerably better. We also experimented with various hyperparameters such as min_model_size, max_num_models, ba_global_function_tolerance, ba_global_max_refinements, ba_refine_focal_length but ultimately the best results we got were with min_model_size =3 and max_num_model = 2 and the rest at default settings.

Also we could only test the dataset against scenes of haiper, heritage and urban because the dataset provided rotational and translational threshold for these 3 scenes only.

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

We were surprised by these low MAA values considering the sparse reconstructions looked decently comprehensible. After getting decently observable results at least, we proceeded to try dense reconstruction. However, we were unable to achieve it. The dense reconstruction involves a couple of steps - Undistortion, Stereo Fusion, Poisson and Delauney reconstruction. Unfortunately, at the Stereo Fusion stage we were receiving CUDA integration errors and were not able to figure out what was going wrong. Hence we had to size it down and stop at Sparse reconstruction.

VI. Summary:

Using a combination of DinoV2, ALIKED and LightGlue we obtained 3D reconstructions of images that were at least presentable. In terms of quantifiable metrics, the results were extremely poor. We realized 3D reconstruction is a problem that requires a lot more time and effort.

Throughout the entire process, we have learnt numerous new concepts. Moving from regular 2D vision problems to 3D vision problems introduced us with additional set of challenges with newer concepts and algorithms. We noticed that the outliers present in the image set clearly makes the model underperform. We learnt the limitations of traditional convolution, learnt about deformable convolutions, geometric transformations and their role in 2D and 3D vision.

A lot more could be done with this to improve the models. We could try using different algorithms like SuperPoint instead of ALIKED or even SIFT. We could spend time to think about what we could do to preprocess the data. For instance, one thing we noticed is the 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. We could try modifying or tuning the exisiting SfM algorithms to yield better results.

We were unable to figure out a way to perform dense reconstruction from the sparse reconstruction we acquired. The project could extend to perform MVS (Multi-view stereo) on the sparse reconstruction.

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)