RBE 549 Project 1: MyAutoPano

UdayGirish Maradana
Robotics Engineering (MS)
Worcester Polytechnic Institute
Worcester, MA 01609
Email: umaradana@wpi.edu
Using 2 Late Days

Pradnya Sushil Shinde
Robotics Engineering (MS)
Worcester Polytechnic Institute
Worcester, MA 01609
Email: pshinde1@wpi.edu
Using 2 Late Days

***Abstract*—The following report consists of a detailed analysis of a classical algorithm and a Deep Learning approach for performing image stitching on a set of three or more images. The work is presented as a part of RBE 549 Project 1: MyAutoPano.**

I. Introduction

The project consists of two phases, Phase 1, which is an implementation of a traditional approach to image stitching, and Phase 2, which describes a Deep Learning approach. Phase 1 follows a procedural flow, to implement panorama stitching, which includes feature detection, feature descriptors, feature matching, eliminating outliers and finally blending and warping of images to form a well-stitched panorama. Phase 2 involves training a Convolutional Neural Network to estimate homography between a pair of images (we call this network a HomographyNet). The intricacies involved in both phases are described in the following sections.

II. Phase I: Traditional Approach

The overview of panorama stitching using the traditional approach can be found in Figure 1. To implement the algorithm for image stitching, we need to relate one image to the other by deciphering a correlation between the distinctive features that each of these images may portray. These features or key points can be defined as corners in the images. Once we obtain the corners, we can proceed with feature-matching to find the commonality between the two images. To achieve robustness in homography, algorithms such as RANSAC are used. At last a robust model that fits the data is computed, and we move forward with image warping and blending. The details regarding the intermediate steps of the algorithm are presented below.

A. Corner Detection

The first step in the traditional approach is to detect key points to find correspondences between images. Corners are good features for finding correspondence as they are distinct and repeatable. We have used the state-of-the-art Harris-Corner detection algorithm to detect corners in a set of images. To implement the corner detection algorithm we make use of *cornerHarris*(), imported from the OpenCV library, that takes a floating-point grayscale image as an input. The results of corner detection for the Train Set are given in Figure 2 and Figure 3.

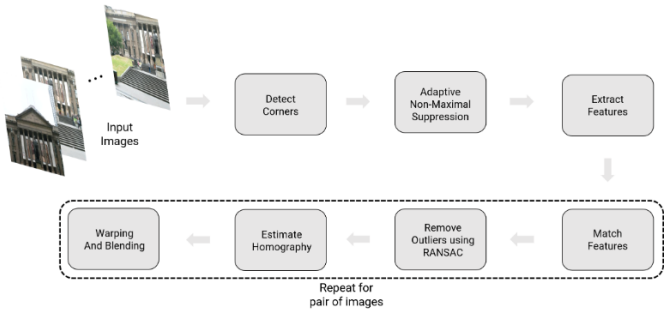

Fig. 1: Overview of Traditional Algorithm for Image Stitching

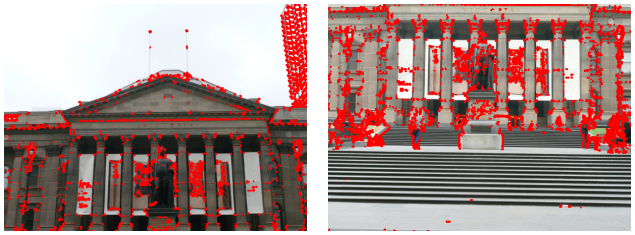

Fig. 2: Train Set 1: Corner Detection

B. Adaptive Non-Maximal Suppression (ANMS)

The corners obtained in the previous step need to be uniformly spread out to avoid random artifacts in warping. Adaptive Non-Maximal Suppression finds the N_{best} corners $N_{best} = 100$ that are equally distributed in the image and at least 2 pixels apart. The algorithm for implementing ANMS is shown in Figure 4. The corresponding results of the ANMS algorithm on the Train Set are shown in the figures below.

C. Feature Descriptor

After extracting N_{best} ANMS corners, we move forward to describing each feature point by a feature vector. We begin by centering a 64×64 patch around each feature point. Then, this patch is blurred using *GausssianBlur()* function of OpenCV and subsampled to the size of 8×8. Finally, we reshape

Fig. 3: Train Set 2: Corner Detection

Input : Corner score Image (C_{img} obtained using `cornermetric`), N_{best} (Number of best corners needed)
Output: (x_i, y_i) for $i = 1 : N_{best}$

Find all local maxima using `imregionalmax` on C_{img};
Find (x, y) co-ordinates of all local maxima;
$((x, y)$ for a local maxima are inverted row and column indices i.e., If we have local maxima at $[i, j]$ then $x = j$ and $y = i$ for that local maxima);
Initialize $r_i = \infty$ for $i = [1 : N_{strong}]$

```
for i = [1 : N_strong] do
    for j = [1 : N_strong] do
        if (C_img(y_j, x_j) > C_img(y_i, x_i)) then
            ED = (x_j − x_i)^2 + (y_j − y_i)^2
        end
        if ED < r_i then
            r_i = ED
        end
    end
end
```

Fig. 4: Adaptive Non-Maximal Suppression Algorithm

the subsampled patch as a 64×1 vector. To remove bias and achieve illumination invariance, we further standardize the vector by making its mean zero and variance equal to 1. A sample of feature descriptor is given in Figure 7.

D. Feature Matching

Once we have each feature point represented as a 64×1 vector, we can proceed with matching each of the feature points in image 1 to feature points in the corresponding image 2. Here, image 1 will be considered as the source image and needs to be compared with each image in the relevant set. The criterion for feature matching is that we calculate the squared distance between each of the corresponding vectors. We then take the ratio of the lowest distance to the second lowest distance and if this ratio is less than a certain pre-assigned factor $= 0.7$, we then consider that the features match. Figures 8 and 9 show feature-matching results for Set 1 and Set 2 respectively.

E. RANSAC for Outliers Rejection and Robust Homography

The feature matches obtained previously are bound to have outliers i.e. incorrect matches. We use the Random Sample Consensus or RANSAC algorithm to remove the outliers and

Fig. 5: Train Set 1: ANMS

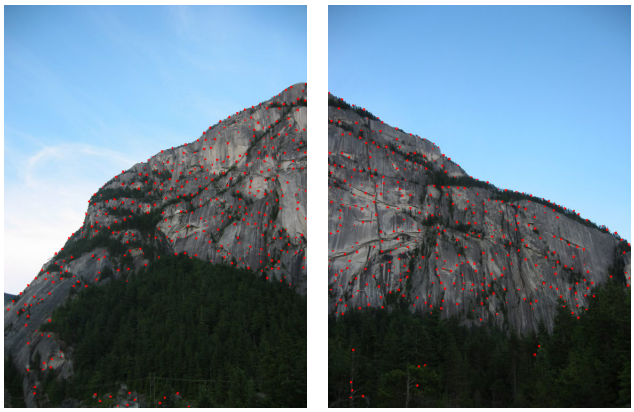

Fig. 6: Train Set 2: ANMS

estimate robust tomography. The structure of the RANSAC algorithm is as follows:

- Four pairs of matching features are selected at random from image 1, p_i and image 2, p'_i.
- Homography matrix H is calculated using getPerspectiveTransform() of OpenCV by passing the source (image 1) and destination (image 2) key points computed in step 1.
- We then proceed to computer inliers where we compare $SSD(\mathrm{p}'_i, H\mathrm{p}_i) < \tau$ where τ is chosen threshold $\tau = 0.5$ and SSD is sum of square difference function.
- Repeat the above steps until we reach the specific number of iterations or we find out the required number of inliers. Keep the best set of inliers.
- Recompute the least square estimate $\hat{H}$ on all of the inliers.

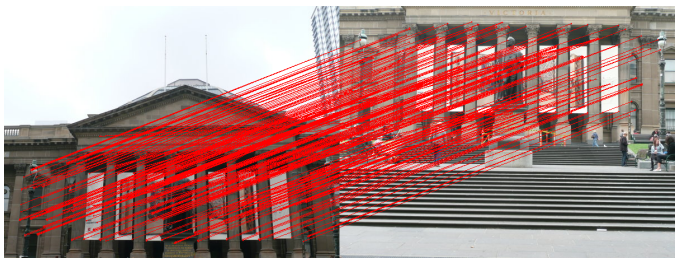

Fig. 7: Train Set1 : Feature Matching

F. Image Blending

Once we have with us the homography transformation matrix from Image 1 to Image 2, we will now move forward with warping the two images. We begin by retrieving the corner points from each image. To the corners of the second image, we apply the homography transformation. Finally, based on the transformed points, we obtain bounding box coordinates $((x_{min}, y_{min}), (x_{max}, y_{max}))$. We then define a translation matrix to adjust the position of the transformed image. To perform the warping operation of image 2 onto image 1 with the translated homography matrix, we will use the $warpPerspective()$ function by OpenCV. Finally, blend image 1 with the result of $warpPerspective()$. The panorama stitching of Train Set 1 and Train Set 2 is shown in Figure 10 and Figure 11.

Fig. 8: Train Set 1: Panorama

The panorama stitching of Test Set 2 and Test Set 3 is shown in Figure 12 and Figure 13.

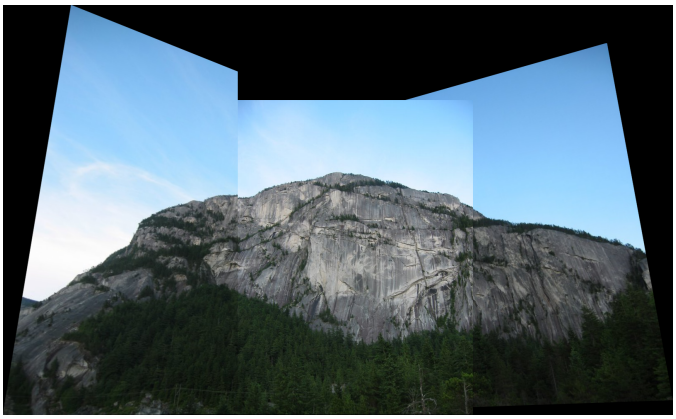

Fig. 9: Train Set 2: Panorma

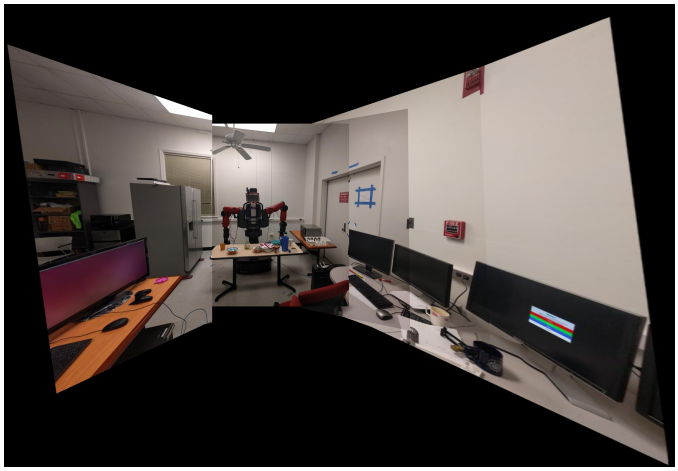

Fig. 10: Train Set 3: Panorama

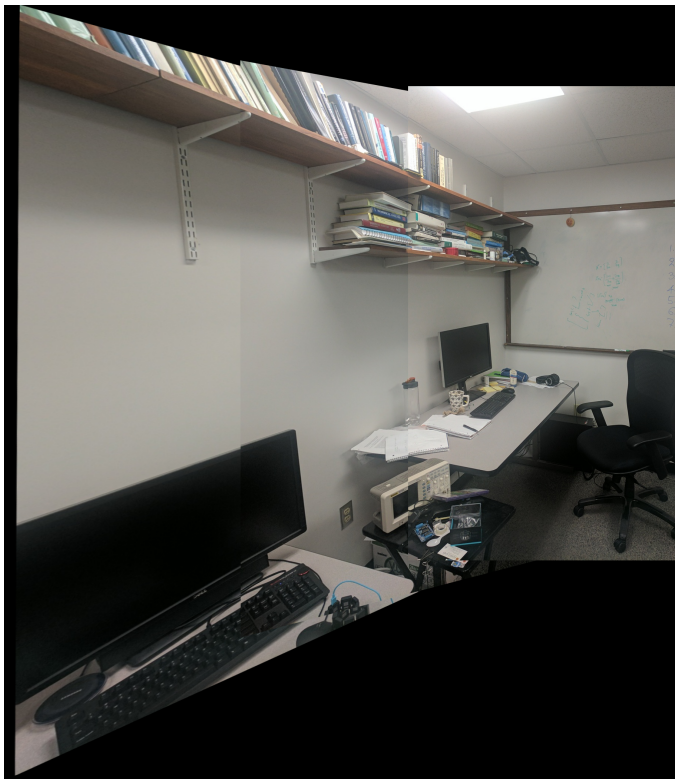

Fig. 11: Test Set 2: Panorama

The panorama stitching of Test Set 1 failed due to the similarity in features found. The extent of this similarity is high and thus it becomes hard to stitch the images. Refer to feature matching for Test Set 1 in Figure 14 and Figure 15.

III. Phase II: Deep Learning Approach

In Phase 2, we focus on training a neural network model with supervised as well as unsupervised methods. The deep learning approach combines corner detection, ANMS, feature extraction, feature matching, RANSAC, and estimate homography all into one. It inputs two grayscale images and provides

Fig. 12: Test Set 3: Panorma

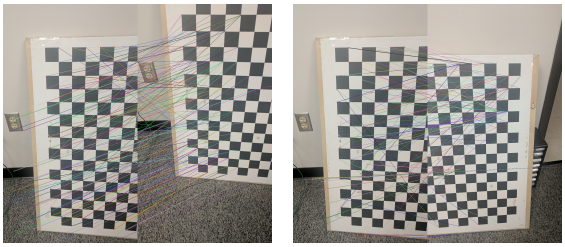

Fig. 13: Test Set 1: Feature Matching

position differences between four corner points of two images as output. From thereon, the homography matrix can be calculated. The overview of the Deep Learning approach is shown in the figure below.

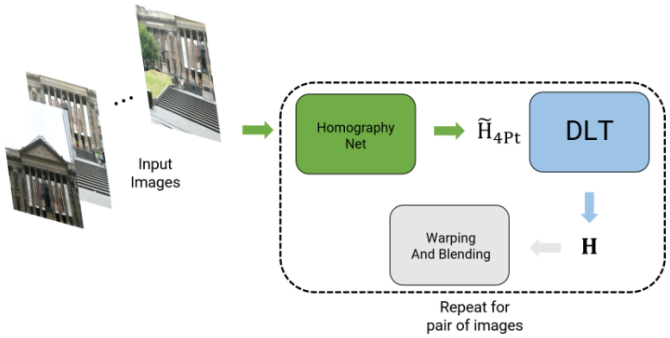

Fig. 14: Overview of Deep Learning Algorithm for Image Stitching

A. Data Generation

To approach homography between images, we need data that has some pre-defined known homography. This could be hard to achieve given the large size of datasets used. Therefore, the need to generate synthetic data becomes necessary. To stitch, we will be using the MSCOCO dataset to generate data. Data generation is done in the following ways:

- We need to generate synthetic image patches and their corresponding perturbed versions along with homography information. For that, first, we will take the input image A and define corner coordinates of Patch A that randomly describes an area in image A.
- The next step is to perform perturbation of Patch A to form Patch B such that it fits within a predefined pixel-shift range. In other words, we need to make sure that the perturbation does not exceed a certain range.
- Once we have Patch A and Patch B corner coordinates, we can compute inverse homography between them.
- We then proceed to warp image A with the corresponding inverse homography matrix, to generate image B.
- The final step is to extract image patches from image A and image B and calculate H_4pt as a difference between the perturbed corner coordinates of Patch B and coordinates of Patch A

We will further use H_4pt. (corner coordinates differences) as our labels to further train data to computer homography.

B. Supervised Appraoch

The supervised model consists of a total of eight convolutional layers. After every second convolutional layer, we define a MaxPooling layer. Towards the end, the final eight linear outputs are obtained which are eight predicted values of H_4pt. The loss function can then be defined as the difference between the predicted values and the ground truth values of H_4pt and is given by:

$$L2 - Loss = \|\mathbf{H}_{4Pt} - \tilde{\mathbf{H}}_{4Pt}\|_2^2$$

The model architecture of the supervised model is shown in the figure below. We have followed a similar architecture as defined in the original paper with almost the same settings for training.

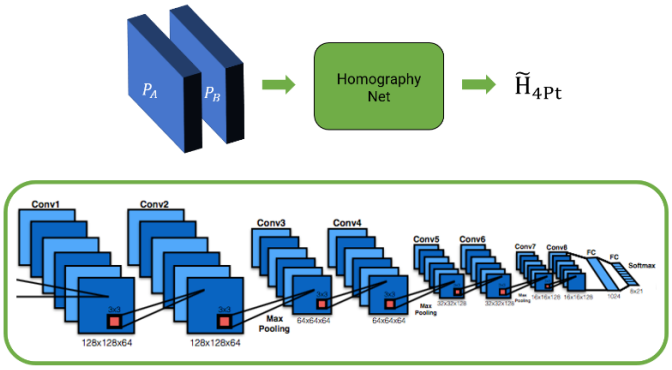

Fig. 15: Overview of Supervised Learning Model

Our Supervised Model architecture is shown in Fig 15,16

```
----------------------------------------------------------------
      Layer (type)               Output Shape         Param #
================================================================
          Conv2d-1         [-1, 64, 128, 128]           1,216
     BatchNorm2d-2         [-1, 64, 128, 128]             128
            ReLU-3         [-1, 64, 128, 128]               0
          Conv2d-4         [-1, 64, 128, 128]          36,928
     BatchNorm2d-5         [-1, 64, 128, 128]             128
            ReLU-6         [-1, 64, 128, 128]               0
       MaxPool2d-7           [-1, 64, 64, 64]               0
          Conv2d-8           [-1, 64, 64, 64]          36,928
            ReLU-9           [-1, 64, 64, 64]               0
         Conv2d-10           [-1, 64, 64, 64]          36,928
           ReLU-11           [-1, 64, 64, 64]               0
      MaxPool2d-12           [-1, 64, 32, 32]               0
         Conv2d-13          [-1, 128, 32, 32]          73,856
           ReLU-14          [-1, 128, 32, 32]               0
         Conv2d-15          [-1, 128, 32, 32]         147,584
           ReLU-16          [-1, 128, 32, 32]               0
      MaxPool2d-17          [-1, 128, 16, 16]               0
         Conv2d-18          [-1, 128, 16, 16]         147,584
           ReLU-19          [-1, 128, 16, 16]               0
      Dropout2d-20          [-1, 128, 16, 16]               0
        Flatten-21               [-1, 32768]               0
         Linear-22                [-1, 1024]      33,555,456
           ReLU-23                [-1, 1024]               0
         Linear-24                   [-1, 8]           8,200
================================================================
Total params: 34,044,936
Trainable params: 34,044,936
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.12
Forward/backward pass size (MB): 63.77
Params size (MB): 129.87
Estimated Total Size (MB): 193.76
----------------------------------------------------------------
```

Fig. 16: Enter Caption

TABLE I: Supervised Model

Performance	Train	Val	Test
EPE	14.48	30.775	25.61
RMSE	3.81	3.13	5.06

We have trained the Supervised model for more than 7 hours using 3.5 Lakh Training patches and 70k Validation Patches with a batch size of 64 and a learning rate of 0.0001 with SGD Optimizer. We have experimented with two variants of almost the same loss function, one time using nn.MSELoss() and the other training experiment using the square root of the nn.MSELoss(). In both of the cases, the loss converged. We also tried experimenting with normalizing both Image and corner differences and also not normalizing them. The nature of loss convergence is the same in both cases except the loss range is different which is expected.

Model Inference Time - **1.78 s** (Measured on RTX3080).

For the above training iteration, we have normalized the Images by dividing with 255 and also the corner differences $\mathbf{H}_{4Pt}$ by 32. This to ensure the smoothness in the learning.

The testing of model on Train, Test and Val datasets has been visualised as seen in the following figures.

C. Unsupervised Approach

The unsupervised model consists of the same Supervised Net in addition to that two new blocks named as TensorDLT and Spatial Transformer network are added. Here rather than taking the loss of the corner coordinate difference we generate a warped patch of P_A using differentiable DLT and Spatial

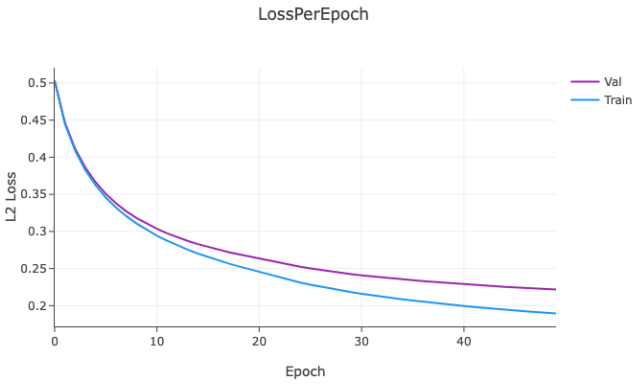

Fig. 17: Supervised Loss Curve

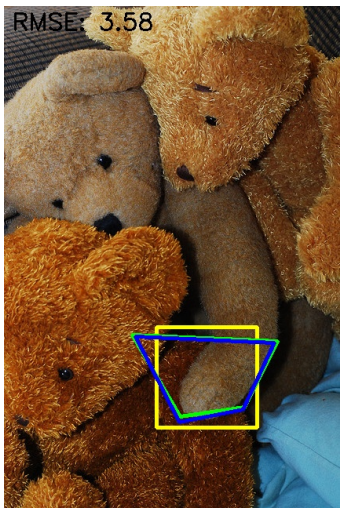

Fig. 18: Supervised Output: Train Image

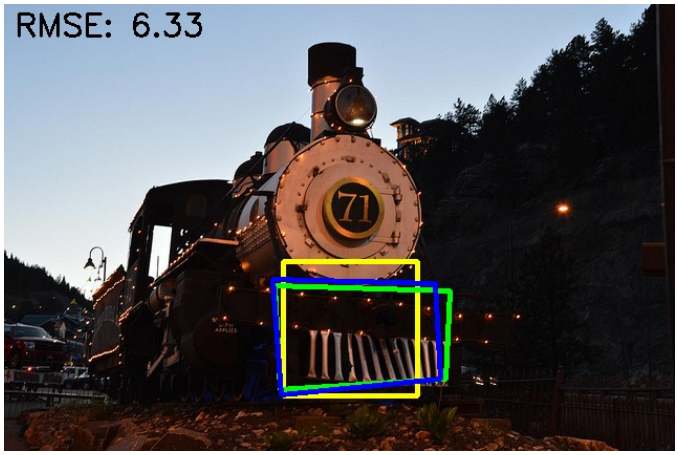

Fig. 19: Supervised Output: Val Image

Fig. 20: Supervised Output: Test Image

transformer methods. From this, we get the predicted patch B P_B.

The loss for this approach is defined as :

$$L1 - Loss = \|\tilde{\mathbf{P}}_B - \mathbf{P}_B\|$$

here Predicted patch is the warped patch A,

$$\tilde{\mathbf{P}}_B = \mathbf{w}(\mathbf{P}_A)$$

In this, the primary concept is to use TensorDLT (Differentiable Direct linear transform) and add a Spatial transformer network (Differentiable Warping function).

Tensor DLT: Let $\mathbf{X}$ and $\mathbf{X}'$ be the homogeneous coordinates of corresponding points in the two images. The relationship between them can be expressed as:

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \mathbf{H} \begin{bmatrix} x \\ y \\ w \end{bmatrix}$$

where $\mathbf{H}$ is the homography matrix. The DLT algorithm aims to estimate $\mathbf{H}$ from a set of corresponding points.

Consider n corresponding points, and let $\mathbf{X}_i$ and $\mathbf{X}'_i$ be the homogeneous coordinates of the i-th point in the two images. The relationship for each point can be written as:

$$\begin{bmatrix} x'_i \\ y'_i \\ w'_i \end{bmatrix} = \mathbf{H} \begin{bmatrix} x_i \\ y_i \\ w_i \end{bmatrix}$$

The DLT algorithm minimizes the reprojection error by solving the linear system of equations. The solution to this linear system provides the elements of the homography matrix $\mathbf{H}$.

Spatial Transformer: We tried to implement both of these using two different methods one is using Kornia (A differentiable Geometric Computer Vision library) and also writing the two methods from scratch using [2] [6]. One of the sources is the original Tensorflow implementation from the authors and one is a custom pytorch-based one. With Kornia-based implementation we got loss converged but the results were bad on the validation set. In the Custom implementation, we got the loss converged to some extent but the results were still not good.

Our L1 Loss converged to **44.4 approx** but the model was overfitting and we felt something wrong with the training as the results were not accurate. Then we tried training with the method using the transformer from the original implementation. The flow of the unsupervised network is as follows:

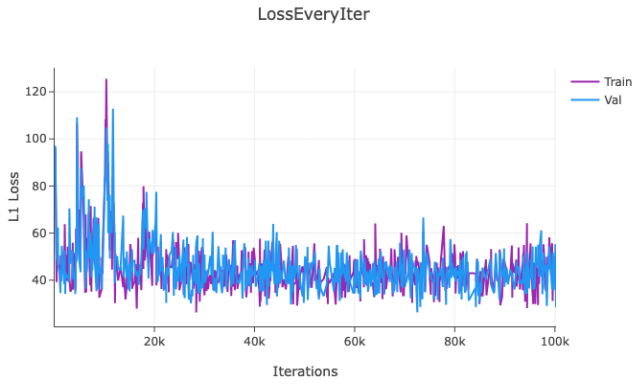

Fig. 21: Unsupervised Loss

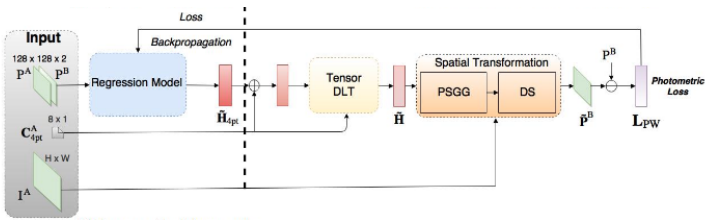

Fig. 22: UnSupervised Approach

We tried approaching the template matching but the results were not good.

D. Other Approaches - We explored (D2Net, SuperGlue, LOFTR)

Based on the experience, we found that even though the current Supervised and Unsupervised methods work, they still lack in adapting to scenarios very well and the input size we have given to the image is too small to generalize and generate good homography for various scenarios. Further reading on this topic, and then we thought why not to implement approaches that can give good descriptors rather than learning the corners differences or the homography? In this way, we can leverage both traditional matching like RANSAC and the deep learning features which are rich in terms of information. One of the famous works we found out about is D2 Net. This implements a framework for the Detection and description of Local features. This combined with traditional matching with threshold can give good results.

Then we researched more for networks with end-to-end capabilities like networks that have a form of RANSAC implemented internally, we found the works of detector-free matching, and surprisingly enough we found that the same authors who authored the Supervised Homography estimation coauthored a new paper in 2019, which is Superglue. This work leverages a neural network that takes two input images as before and matches a set of local features by jointly

finding correspondences and rejecting non-matchable points. This primarily leverages the concepts of Graph neural network which is kind of one of the most used approaches in Geometric computer vision. Further, this work is extended by a team of researchers, and in this work they replaced the past work with Transformers, especially leveraging the concept of multi-head attention. The primary disadvantage of these approaches is they need a lot of data to train. One of the primary challenges we faced with the traditional approach with features is scale and rotation. We have tested some of the results from Superglue and the results are truly great. One of the results is attached here. But even deep networks are quite prone to rotational variance.

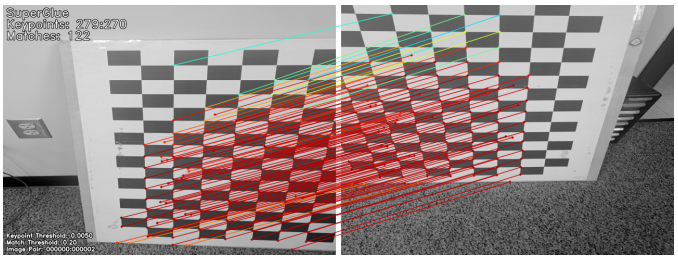

Fig. 23: SuperGlue Ouptut

IV. Discussion and Conclusion

This report presents both the Traditional and Deep learning-based approaches for homography estimation. The above pictures (Fig 18,19,20) show the results from supervised implementation. Each of the pictures describes the Patch A corners (Yellow), Patch B Corners (Green), and the predicted Homography or the Corners (Green). Even though we tried the Unsupervised approach and somehow the loss converged the outputs are very bad which were not plotted in some of the images. And further, the Unsupervised approach is a bit harder to train as it depends on specific parameters and it is more prone to getting stuck in local minima. The supervised approach is easy to train and also the results are accurate.

One of the primary confusions we had is the difference in the approaches listed in the Paper and the resource material (course website) we have regarding training the Unsupervised Network. Primarily regarding the difference between passing Image A in the original paper and patch A in the resource provided to the Spatial Transformer network to get Warp. This is something we want to understand further.

V. Further Explorations

To explore further Deep learning techniques we have tried to understand some of the approaches such as Superglue which are explained above. We equally tried to understand how to solve issues related to multiple issues such as rotation, scale, and repetition of features and understood that approaches such as BRIEF, SIFT, and ORB solve these issues.

In one of my previous works, we also explored LoFTR which is one of the state-of-the-art approaches in Feature learners for homography (End to End flow for getting feature matches). This was recently integrated into Kornia and the testing code we have placed in the submission.

References

[1] DeTone, D., Malisiewicz, T. & Rabinovich, A. Deep Image Homography Estimation. (2016)

[2] Nguyen, T., Chen, S., Shivakumar, S., Taylor, C. & Kumar, V. Unsupervised Deep Homography: A Fast and Robust Homography Estimation Model. (2018)

[3] Sarlin, P., DeTone, D., Malisiewicz, T. & Rabinovich, A. SuperGlue: Learning Feature Matching with Graph Neural Networks. (2020)

[4] Sun, J., Shen, Z., Wang, Y., Bao, H. & Zhou, X. LoFTR: Detector-Free Local Feature Matching with Transformers. (2021)

[5] Dusmanu, M., Rocco, I., Pajdla, T., Pollefeys, M., Sivic, J., Torii, A. & Sattler, T. D2-Net: A Trainable CNN for Joint Detection and Description of Local Features. (2019)

[6] Github- Unsupervised Implementation: https://github.com/breadcake/unsupervisedDeepHomography-pytorch/tree/main