MyAutoPano

Ajith Kumar Jayamoorthy
MS in Robotics Engineering
Worcester Polytechnic Institute
ajayamoorthy@wpi.edu

Shiva Kumar Tekumatla
MS in Robotics Engineering
Worcester Polytechnic Institute
stekumatla@wpi.edu

***Abstract*—In this project, we presented two different ways of stitching two or more images to create a seamless panorama image. In this work, we considered images with repeated local features. One of the two ways is classical computer vision method and the other one is deep-learning method.**

I. Phase 1: Traditional Approach

A traditional method that can be used for panorama stitching is shown in figure 1.

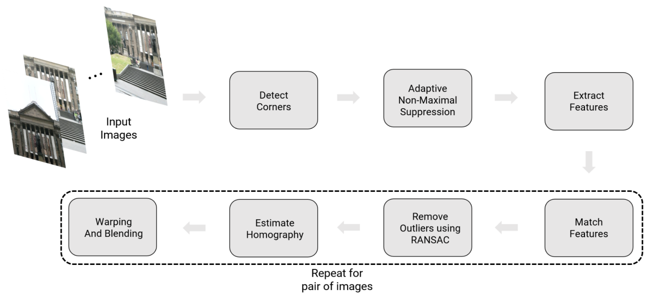


Fig. 1. Overview of panorama stitching using traditional method

In this approach, initially the corners are detected for each image that needs to be stitched together. Once the corners are detected, adaptive non-maximal suppression method is method is used to find local maxima of corners. Features are extracted from each image and then matched.Matched features clubbed with random sample consensus, and the outliers are removed. Homography is estimated and then the images are blended together. Three sample images and their expected panorama is shown in figure 2.

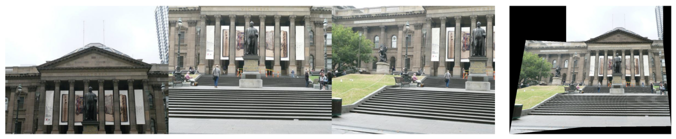


Fig. 2. First three images: Input to the panorama stitching algorithm, last image: output of the panorama stitching algorithm

A. Corner Detection

To detect corners in the given image, we used Harris Corners method. This functionality is achieved from cv2.cornerHarris method.We used a threshold of 0.01 , block-size of 2 and Sobel size of 3. The outputs for different images are given by figures 3, 4, and 5.

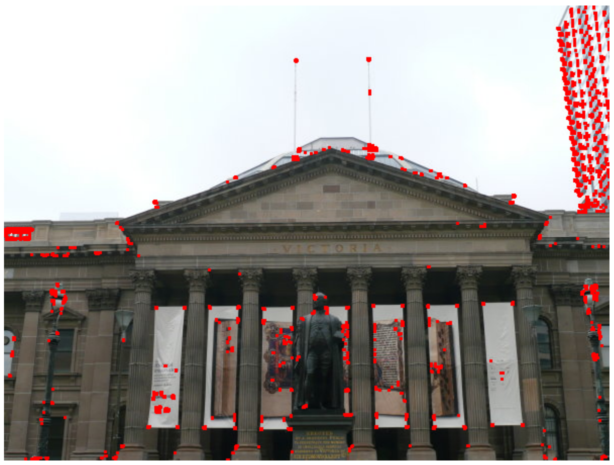


Fig. 3. Detected corners using Harris Corners method for input 1

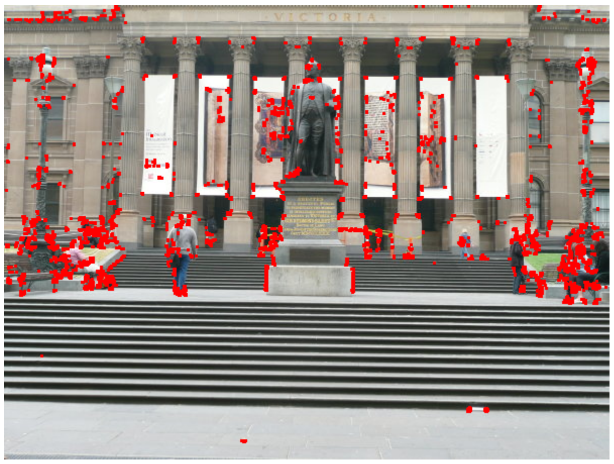


Fig. 4. Detected corners using Harris Corners method for input 2

But the corners detected here are too dense . If these corners are used directly for the feature matching , because of too many corners , computation can be expensive. These corners can be reduced using adaptive non-maximal suppression method.

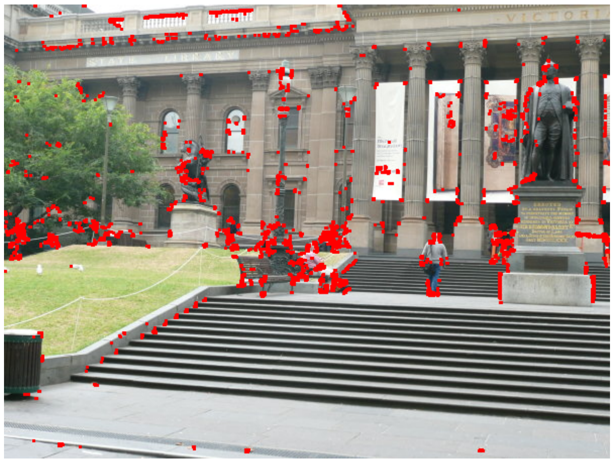


Fig. 5. Detected corners using Harris Corners method for input 3

B. Adaptive Non-Maximal Suppression (ANMS)

Main motive of this step is to reduce the corners such a way that they are equally distributed throughout the image. This method can make sure that the final stitched image will not have weird warping.

In a real image, a corner is never perfectly sharp, each corner might get a lot of hits out of the N strong corners - we want to choose only the N best corners after ANMS.The algorithm for implementing ANMS is given by figure 6.

```
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
Sort $r_i$ in descending order and pick top $N_{best}$ points
```

Fig. 6. Algorithm to implement ANMS

Output of ANMS for each input images is shown by figures 7 , 8, and 9.

C. Feature Descriptor

In this step, each feature corner also called as feature point is converted into feature vector. For this conversion, we Took a patch of size 40×40 centered around the key-point/feature point , and applied Gaussian blur. We used cv2.GaussianBlur command with default parameters. After that , blurred output is sub-sampled to 8×8. This is reshaped to obtain a 64×1 vector. This vector is standardized to have zero mean and variance of

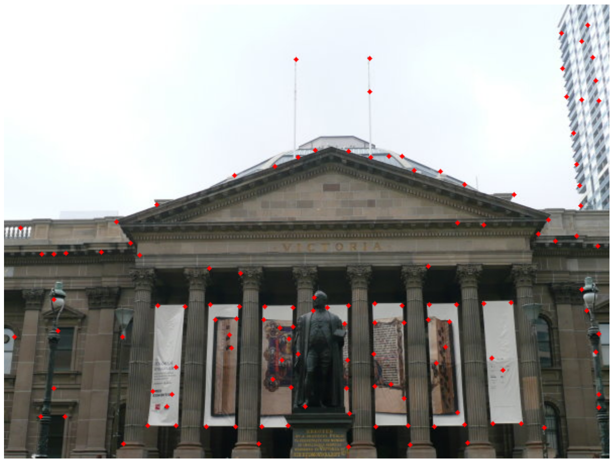


Fig. 7. ANMS output for input 1

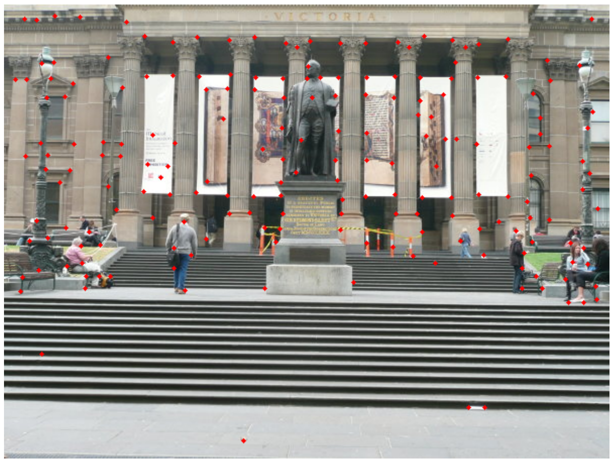


Fig. 8. ANMS output for input 2

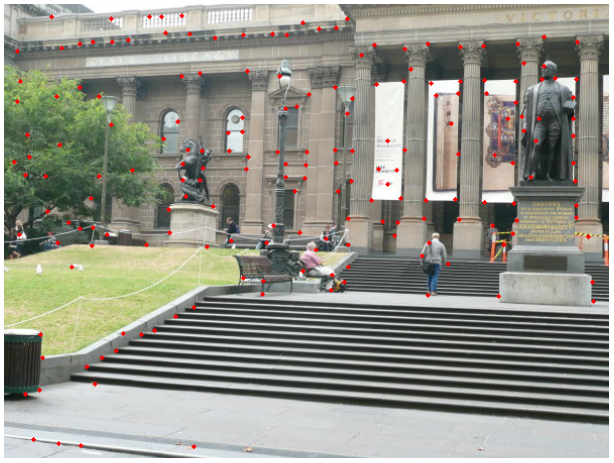


Fig. 9. ANMS output for input 3

1.Here, Standardization helps to remove bias and to achieve some amount of illumination invariance.

D. Feature Matching

Each previously encoded 64×1 feature vectors from each image is matched using feature correspondence. This can be computed by the following method: 1)Pick a point in image 1, compute sum of square differences between all points in image 2. 2) Take the ratio of best match (lowest distance) to the second best match (second lowest distance) and if this is below some ratio keep the matched pair or reject it. 3) Repeat this for all points in image 1. 4)You will be left with only the confident feature correspondences and these points will be used to estimate the transformation between the 2 images, also called as Homography. Output of feature matching for each pair is shown by figures 10 , 11 , and 12.

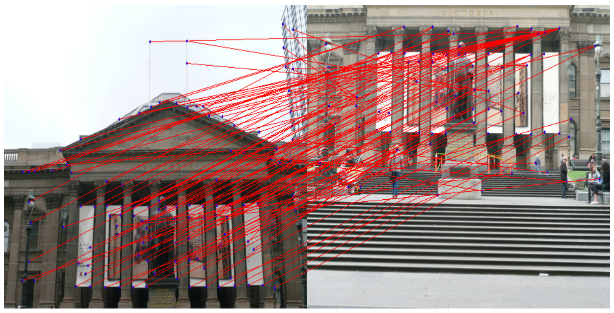


Fig. 10. Output of Feature Matching for input 1 and 2

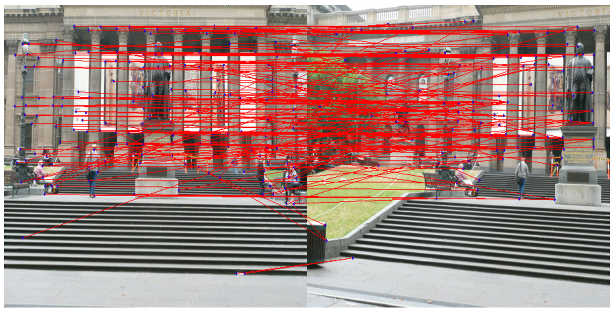


Fig. 11. Output of Feature Matching for input 2 and 3

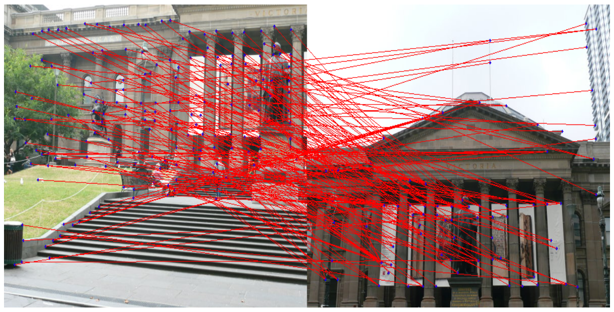


Fig. 12. Output of Feature Matching for input 3 and 1

In this step , we can clearly observe the wrong matches. This can be solved by using random samples consensus method, also known as RANSAC.

E. RANSAC for outlier rejection and to estimate Robust Homography

We now have matched all the features correspondences but not all matches are right. To remove incorrect matches, we used a robust method called Random Sample Concensus or RANSAC to compute homography.

The RANSAC steps are:

1) Select four feature pairs (at random) from both images
2) Compute homography H between the previously picked point pairs
3) Compute inliers where SSD is lower than some threshold
4) Repeat the last three steps until you have exhausted Nmax number of iterations (specified by user) or you found more than some percentage of inliers
5) Keep largest set of inliers

The outputs after eliminating the outliers are shown by figures 13 , 14 , and **??**

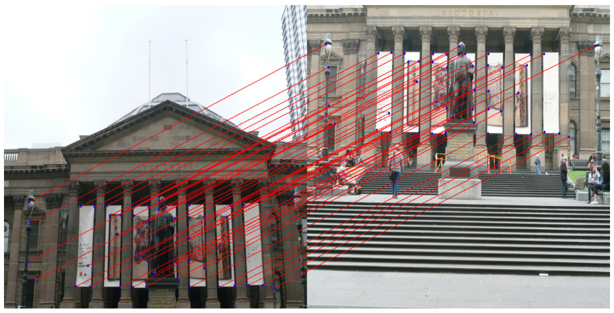


Fig. 13. Feature matches after outliers have been removed using RANSAC for input 1 and 2

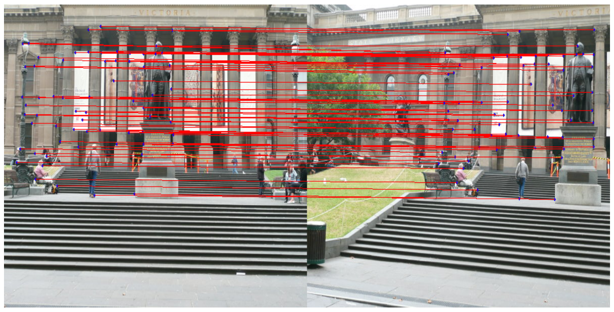


Fig. 14. Feature matches after outliers have been removed using RANSAC for input 2 and 3

F. Blending Imges

Panorama can be produced by overlaying the pairwise aligned images to create the final output image. For this we have used pairs of given input images. Initially input image 1 and 2 are stitched together , later input images 1 and 3. Finally the result of these two are stitched together. The results for these are shown by figures 16 , 17 , and 18.

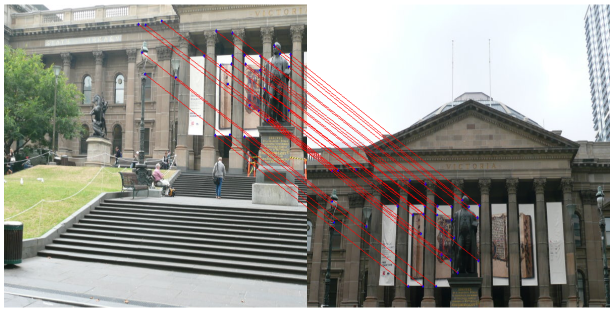


Fig. 15. Feature matches after outliers have been removed using RANSAC for input 3 and 1

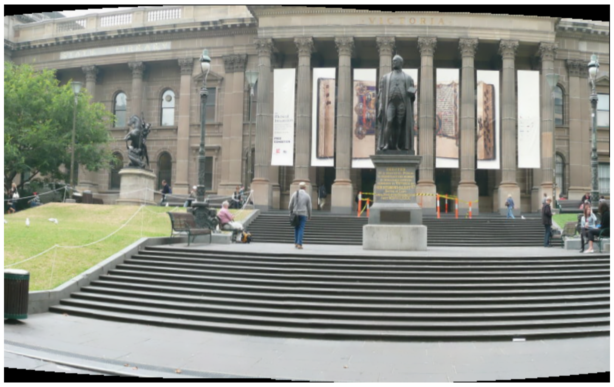


Fig. 16. Stitched output for input 1 and 2

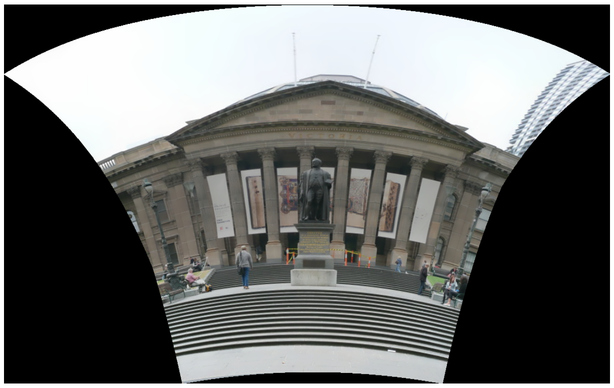


Fig. 17. Stitched output for input 1 and 3

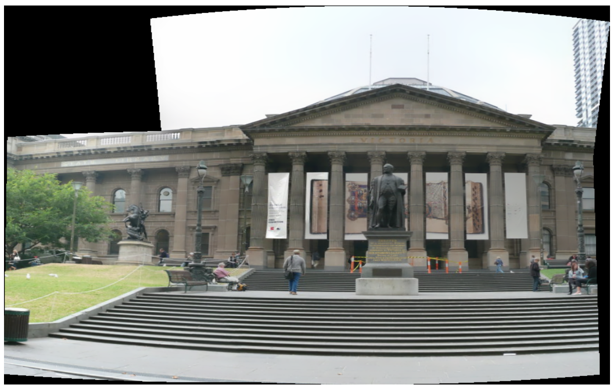


Fig. 18. Final Panorama

II. PHASE II - DEEP LEARNING APPROACH

In this phase, we have implemented the deep learning approach to perform homography estimation between two images, one being the original image and the second being the warped duplicate of the original image. Also, we have build two type of network, namely, supervised as well as unsupervised networks.

A. Data Generation

The primary source of data is taken from the **MSCOCO** dataset consisting of color images. Due to the huge size of the dataset, a small sample has been taken from it and it has been divided into Train, Validation and Test Dataset. The Train, Validation and Test datasets consist of 5000, 1000 and 1000 images, respectively.

The original images are provided in a zip-file. The images are extracted from the zip-file, converted to gray-scale and then stored into a folder named TrainOrgImgs. Then a small patch of size 64 x 64 pixels is considered at random from the gray-scale image and a perturbation of value (ρ = 16) is applied. The original patch and the resultant patch are stacked together and stored into another folder named Train. The difference between the original corners and the perturbed corners is calculated as H4Pt. This values is then stored as Label for our data. Similarly, the process is repeated for Validation and Test dataset.

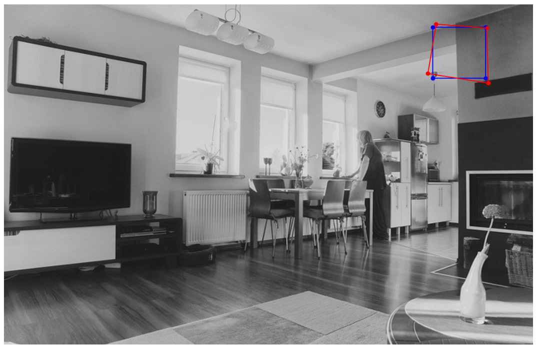


Fig. 19. Sample Training Image (Blue: Patch-A; Red: Patch-B)

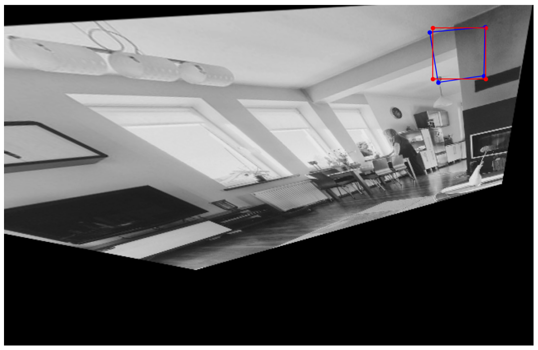


Fig. 20. Warping of image from B to A

Fig. 21. Sample Training patches; Left: Patch-A, Right: Patch-B

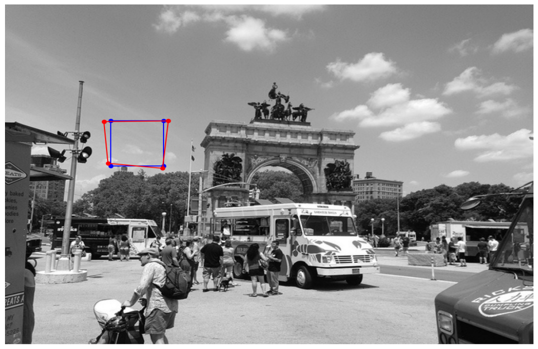


Fig. 22. Sample Validation Image (Blue: Patch-A; Red: Patch-B)

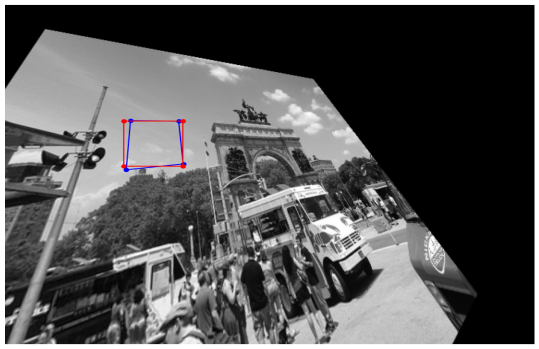


Fig. 23. Warping of image from B to A

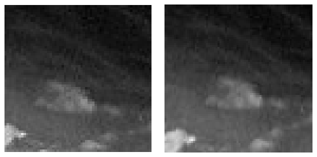


Fig. 24. Sample Validation patches; Left: Patch-A, Right: Patch-B

B. Network Architecture

1) Supervised Learning: The base homography architecture has been provided as shown by figure 25. The input image size

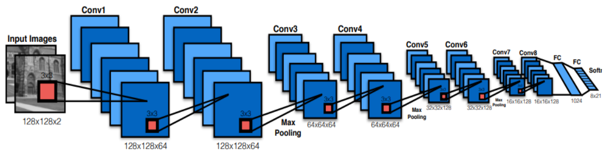


Fig. 25. Supervised Homography Architecture

considered in our scenario is a 64x64 patch. The two patches A and B are stacked one behind another and are stored as numpy arrays. The label for the particular patch pair has been stored in a text document as described in the Data preparation Step. The overall flow of the code can be represented by the diagram in Figure 26. The output of the architecture is an

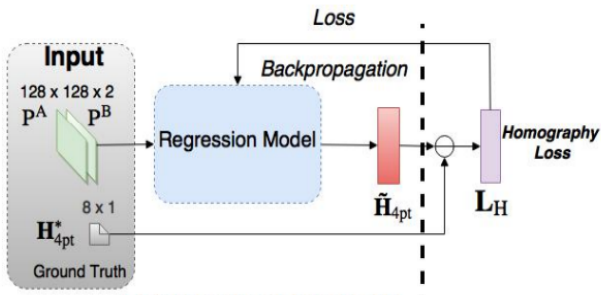
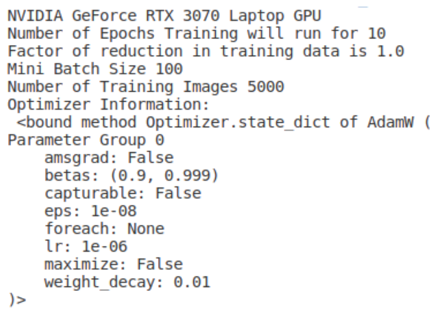


Fig. 26. Supervised Homography Architecture Diagram

1x8 matrix which represents the perturbation (H4Pt) in each corner of the patch A corresponding to the patch B. The model trains to learn the Homography between Patch A and Patch B and displays the H4Pt values as the output. The GenerateBatch function was divided into two function: SupGenerateBatch and UnSupGenerateBatch, to satisfy accordingly the two networks. A random sample function has been implemented in the GenerateBatch function to guarantee the randomness of data during training and to avoid order bias during training. In the Network file, two different classes have been created, namely SupHomographyModel and UnSupHomographyModel to cater the needs to two different architectures. Based on the input of the model-type while running the code, the appropriate model is initiated and the the respective training process is carried over in the training code. In case of the network optimizer both SGD as well as AdamW has been used to train the data. The Figure 27 and Figure 28 shows the optimizer used and Model architecture, respectively.

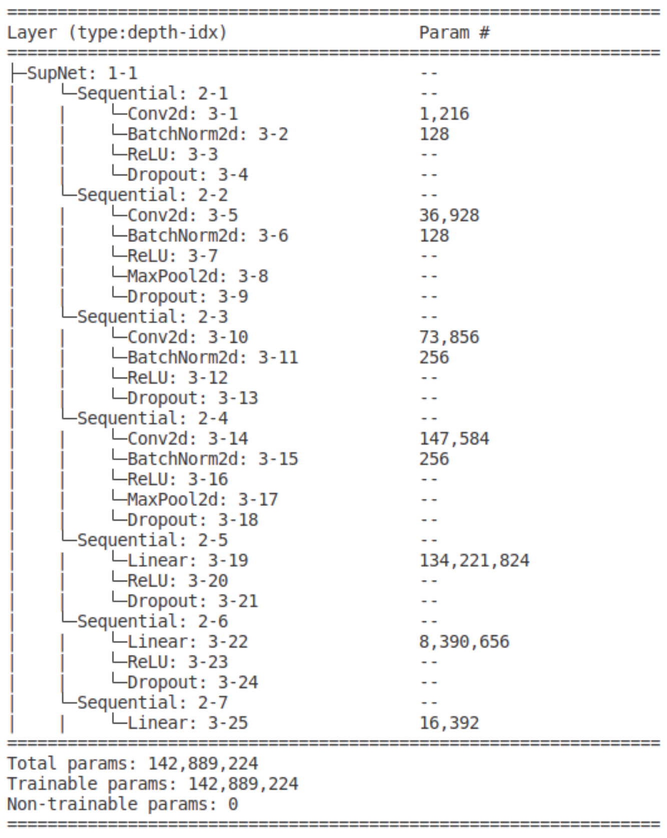


```
NVIDIA GeForce RTX 3070 Laptop GPU
Number of Epochs Training will run for 10
Factor of reduction in training data is 1.0
Mini Batch Size 100
Number of Training Images 5000
Optimizer Information:
 <bound method Optimizer.state_dict of AdamW (
Parameter Group 0
    amsgrad: False
    betas: (0.9, 0.999)
    capturable: False
    eps: 1e-08
    foreach: None
    lr: 1e-06
    maximize: False
    weight_decay: 0.01
)>
```

Fig. 27. Details of data split and optimizer used in the Supervised learning

2) Unsupervised Learning: The base homography architecture has been provided as shown by figure 29.

The initial CNN network of the Unsupervised model is similar to the supervised learning part. In addition we have two more components called the Direct Linear Transform (DLT) and Spatial Transformer Network (STN). The flow of output in the unsupervised model is as show in Figure 30



```
=================================================================
Layer (type:depth-idx)                   Param #
=================================================================
├─SupNet: 1-1                            --
|    └─Sequential: 2-1                   --
|    |    └─Conv2d: 3-1                  1,216
|    |    └─BatchNorm2d: 3-2             128
|    |    └─ReLU: 3-3                    --
|    |    └─Dropout: 3-4                 --
|    └─Sequential: 2-2                   --
|    |    └─Conv2d: 3-5                  36,928
|    |    └─BatchNorm2d: 3-6             128
|    |    └─ReLU: 3-7                    --
|    |    └─MaxPool2d: 3-8               --
|    |    └─Dropout: 3-9                 --
|    └─Sequential: 2-3                   --
|    |    └─Conv2d: 3-10                 73,856
|    |    └─BatchNorm2d: 3-11            256
|    |    └─ReLU: 3-12                   --
|    |    └─Dropout: 3-13                --
|    └─Sequential: 2-4                   --
|    |    └─Conv2d: 3-14                 147,584
|    |    └─BatchNorm2d: 3-15            256
|    |    └─ReLU: 3-16                   --
|    |    └─MaxPool2d: 3-17              --
|    |    └─Dropout: 3-18                --
|    └─Sequential: 2-5                   --
|    |    └─Linear: 3-19                 134,221,824
|    |    └─ReLU: 3-20                   --
|    |    └─Dropout: 3-21                --
|    └─Sequential: 2-6                   --
|    |    └─Linear: 3-22                 8,390,656
|    |    └─ReLU: 3-23                   --
|    |    └─Dropout: 3-24                --
|    └─Sequential: 2-7                   --
|    |    └─Linear: 3-25                 16,392
=================================================================
Total params: 142,889,224
Trainable params: 142,889,224
Non-trainable params: 0
=================================================================
```

Fig. 28. Summary of the Network Architecture used in the Supervised learning

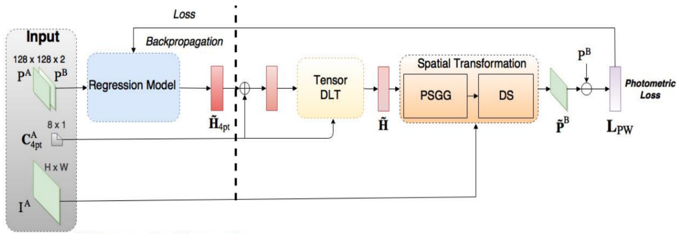


Fig. 29. Unsupervised Homography Architecture Diagram

3) Unsupervised Learning: The base homography architecture has been provided as shown by figure 29.

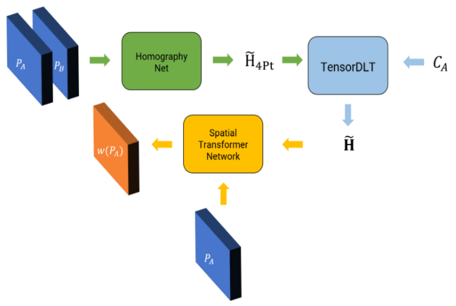


Fig. 30. Flow of Unsupervised Network model

4) Direct Linear Transform (DLT): DLT takes in the input from H4Pt obtained from the output of the initial CNN network and the Corner co-ordinates of Patch A. Using this information we calculate the predicted corners of Patch-B. This Patch-A corner and Patch-B corner information is further used to evaluate the Homography matrix of size (3x3). In the paper, the authors have discussed a method to calculate the equation.In our implementation we have considered each element in a batch tensor by tensor and have evaluated the Ai matrix accordingly. From that we have obtained using the equation Ai * h = bi, we have calculated the value of h which is a 1x8 matrix.

5) Spatial Transformer Network (STN): The STN uses the output of the DLT, the 3x3 homography matrix and the patch A to predict the patch B. Once warped, the photo-metric loss is computed by comparing the predicted patch and the actual patch.

C. Results

The supervised learning model has been trained and tested and the loss in pixel are as shown in the table below:

Model-name	Val-EPE	Test-EPE
Sup+Static	32.25 px	90.96 px

It can be observed that the training loss is less but the testing loss is very high. This might be due to either overfit or underfitting of the model with the training data. The figure 31 show the Loss-vs-Iteration plot for the validation data during the training process.

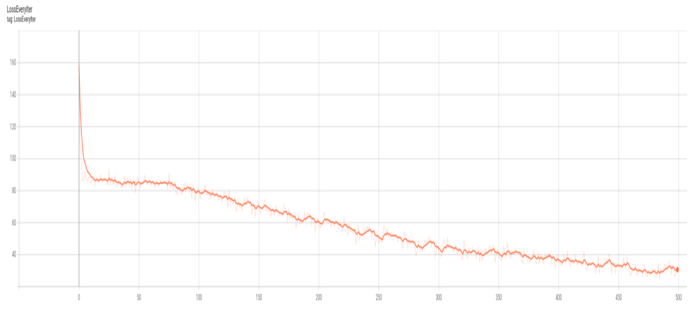


Fig. 31. Validation Loss-vs-Iteration plot

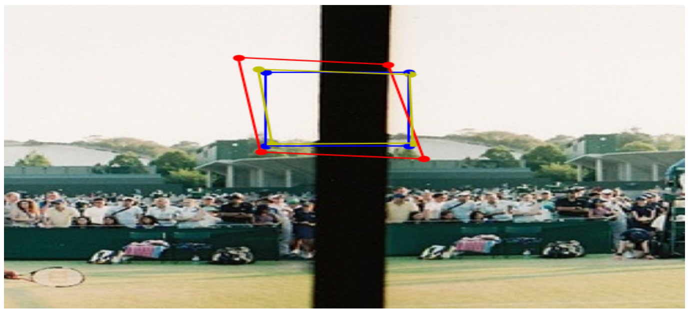


Fig. 32. The following is the visualization of the output of the Supervised Network: Blue - Patch A; Red - Patch B and Yellow - Predicted Patch B

The model is run on the a few testing set and the output of the model along with the ground truth are visualized as shown in the Figure 32. We can observe that the network has predicted the patch B co-ordinates very different from the ground truth. This shows that the network has not learnt properly and might be over-fitting the training data leading to huge testing loss.

References

[1] https://rbe549.github.io/fall2022/proj/p1/

[2] DeTone, D., Malisiewicz, T., & Rabinovich, A. (2016, June 13). Deep image homography estimation. arXiv.org. Retrieved September 17, 2022, from https://arxiv.org/abs/1606.03798

[3] Unsupervised deep homography: A fast and robust homography estimation model. (n.d.). Retrieved September 18, 2022, from https://ieeexplore.ieee.org/document/8302515