HW1 - MyAutoPano

Mihir Deshmukh
Robotics Engineering
Worcester Polytechnic Institute
Email: mpdeshmukh@wpi.edu

Ashwin Disa
Robotics Engineering
Worcester Polytechnic Institute
Email: amdisa@wpi.edu

Using 1 LATE day

***Abstract*—In this project assignment, the phase 1 of the project involves stitching multiple images with $30 - 50\%$ overlap to generate a panorama using the classical approach. And in phase 2 we implement deep learning approaches to estimate the homography between two images, bassically combining the classical methods like corner detection, ANMS, feature extraction and matching, RANSAC and estimating homography into one.**

I. Phase 1: Classical approach

The overview of the method followed is shown in fig. 1 and a given test test consisting of 3 overlapping images is shown in fig. 2.

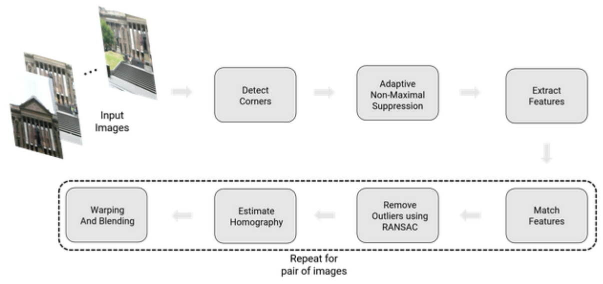

Fig. 1. Overview of the method

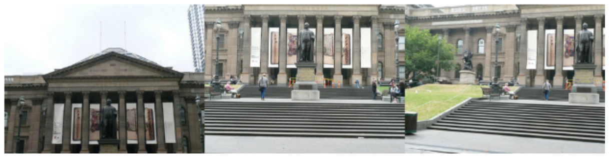

Fig. 2. Input to the panorama stitching.

A. Corner Detection

We first detect corners of the input image using the $cv2.cornerHarris()$ function. Output is shown in fig. 3. The output of the function is a matrix of same size as the image with probability of a corner as each element. We also tried the $cv2.goodFeaturesToTrack()$ function which combines the corner detection and ANMS (Adaptive Non-Maximal Suppression) to output the N_{best} corners in the image. The output of this method is shown in fig. 4.

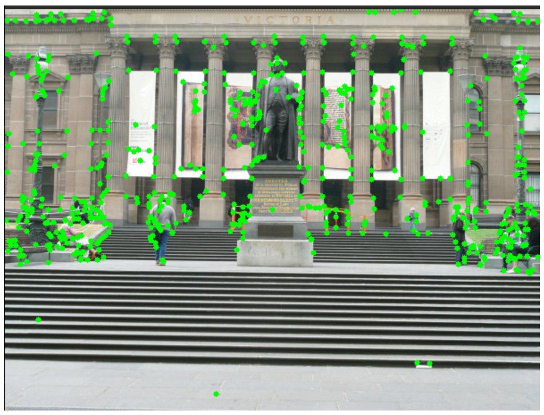

Fig. 3. Corners extracted using Harris corners method.

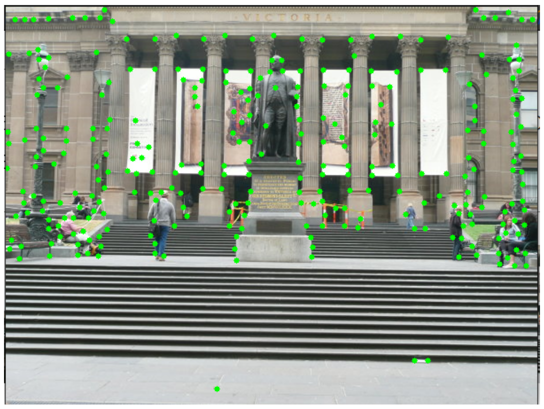

Fig. 4. Corners extracted using cv2.goodFeaturesToTrack() method.

B. ANMS

The objective of this step is to detect corners such that they are equally distributed across the image in order to avoid weird artifacts in warping. In a real image, a corner is never perfectly sharp, so we choose the N_{best} corners which are true local minima. The output of the ANMS is shown in fig. 5. The result is similar to what we got from the $cv2.goodFeaturesToTrack()$ function.

C. Feature Descriptor

Using the N_{best} corners from the ANMS method, we describe each feature point by a feature vector, this is like encoding the information at each feature point by a vector.

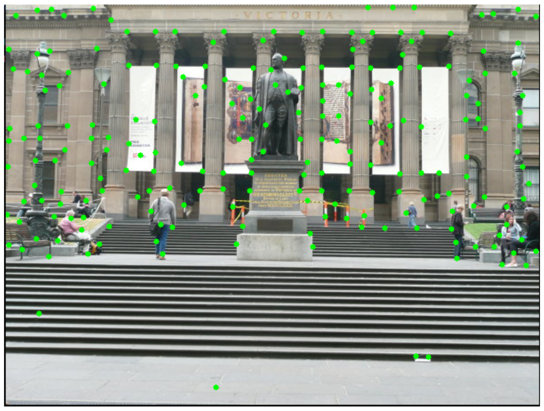

Fig. 5. Output of ANMS method.

We take a patch of size 41×41 centered around the keypoint. We blur the patch using the $cv2.GaussianBlur()$ function. Before that, we pad the image by half the size of the patch to make sure the edge pixels are considered. The blurred patch is resized to 8×8 and reshaped to a 64×1 vector. This vector is standardized to have zero mean and variance as 1. This done to reduce bias and to achieve some amount of illumination invariance.

D. Feature matching

All N_{best} keypoints of all images are encoded as a 64×1 feature vector. Now, we match these feature vectors to stitch the images. We pick a point in image 1 and compute the sum of squared differences between all points in image 2, take the ratio of best match(lowest difference) to the second best match (second lowest) and if this is below a threshold (here 30), we keep the matched pair. We repeat this step for all keypoints in image 1. We are 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. we use the function $cv2.drawMatches()$ to visualize the feature correspondences. This is shown in fig. 6 and fig. 7. If the algorithm is not able to detect enough features between given two images, the image is skipped and the next image is taken for comparison. This helps to make the algorithm more robust, incase a different image is added to the list of images for panorama.

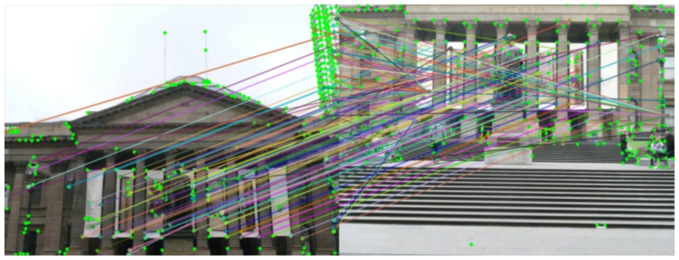

Fig. 6. Feature matching between two images.

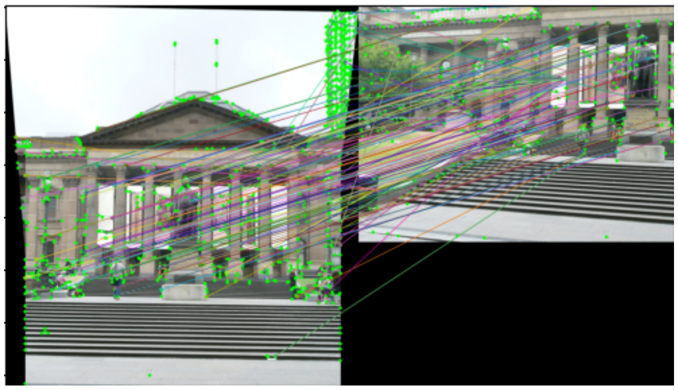

Fig. 7. Feature matching between stitched images and the third image.

E. RANSAC

As it can be seen from figures shown, not all features matched are accurate. To remove the inaccuracies, we implement a robust method called RANSAC (Random Sample Concensus) to reject the outliers to generate the homography. We extract 4 random feature pairs from both images and compute the homography (transformation) based on these random points. We compute the inliers by calculating the sum of square differences and compare with a threshold (here 30), the ones lower are the inliers. We repeat these steps for 1000 times to maximise out chances of getting the accurate inliers. Finally we estimate the homography using the inliers which is used for stitching the images. The homography is computed using the $cv2.findHomography()$ function. The accurate matches or inliers are shown in fig. 8 and fig. 9. The lines joining two matching features are parallel to each other tell us that the feature matching is accurate and necessary to estimate the homography precisely.

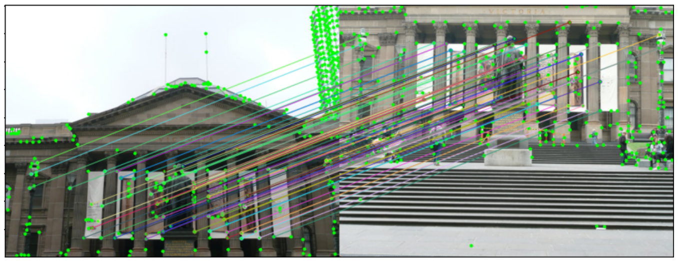

Fig. 8. Feature matching between two images after RANSAC.

F. Stitching

Using the estimated homography from the previous section, we warp the image2 onto the image 1 using the $cv2.warpPerspective()$ to obtain the required stitched image. The final results are shown in fig. 10 and a stitched image from the test set are shown in 11 and 12. The test set 4 is given with 2 images that do not belong to the panorama, the algorithm is able to successfully reject these outliers and return the correct output.

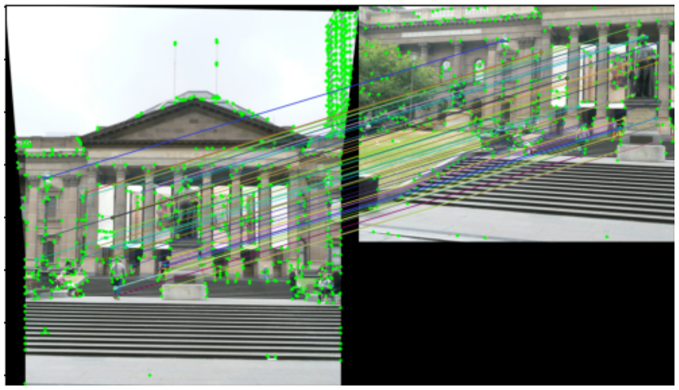

Fig. 9. Feature matching between stitched images and the third image after RANSAC.

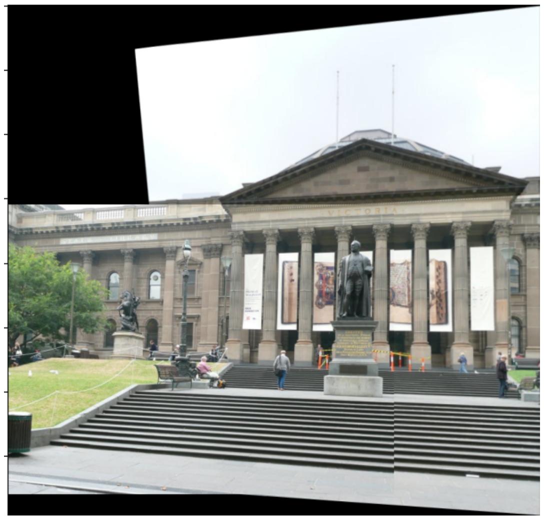

Fig. 10. Panorama of 3 stitched image.

II. Phase 2: Deep Learning Approach

A. Data Generation:

For training our model, we generate a dataset using a subset of the MSCOCO dataset(5000) images. We generate a synthetic dataset using these images. For this, we first resize all the images to 480*480. Now, utilizing the resized images, we decide on a region of interest, as mentioned in the problem, which is the central part of the image, excluding the 160-pixel margin on each side. Next we generate a random point in the ROI and extract a 128*128 patch. We generate random perturbation, which is between [-24,24], and add this to the four corner pixel coordinates to get the perturbed corners. Using this set of corners, we compute homography. The inverse homography is used to warp the image, and we extract the patch from the original patch coordinates to get the warped patch, as mentioned in the problem statement. We save the original patch, the warped patch, the perturbed corners, and the original corner coordinates.

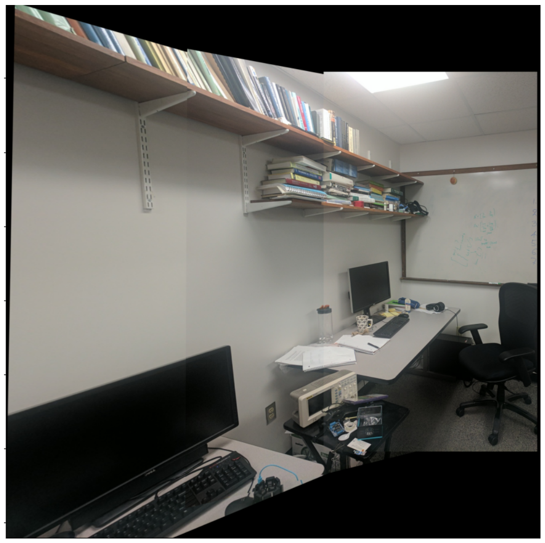

Fig. 11. Panorama of the test set2.

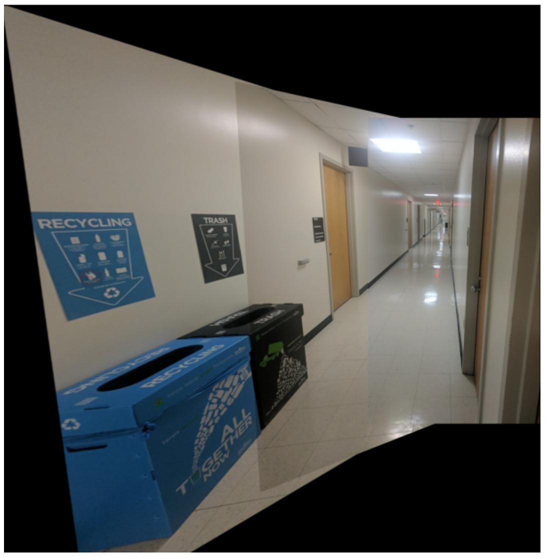

Fig. 12. Panorama of the test set3.

B. Supervised approach:

We employ the same model architecture as described in the HomographyNet paper. This consists of 8 convolution layers followed by Batch Norm and activation function(ReLU in this case). As seen in Figure 13, we employ two conv layers at each resolution halved after every two layers using a max pool operation [128*128, 128*128, 64*64, 64*64, 32*32, 32*32, 16*16, 16*16]. The number of channel dimensions/feature maps doubles at every other resolution drop, which is [64, 64, 64, 64, 128, 128, 128, 128]. The 16*16*128 feature map is then given to a fully connected layer, bringing these features down to 1024. Finally, the last layer brings these down to 8, our model output. It is to be noted here that we solve this as a regression problem, unlike as shown in Figure 13. Figure 15 details the model architecture.

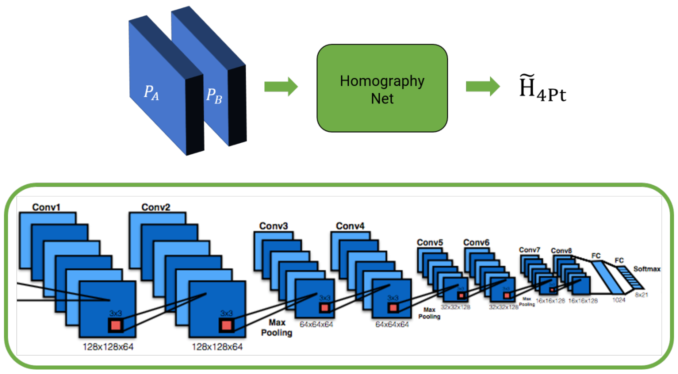

Fig. 13. Supervised Model architecture

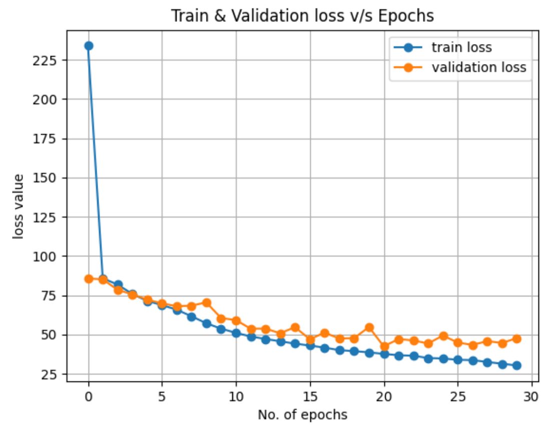

Fig. 14. Train and Validation Loss for Supervised model.

We train the model using the MSE loss function. The Adam optimizer is used with a learning rate of 0.0025 and a weight decay 0.0001. The model was trained on 30 epochs. We observe overfitting after about 20 epochs, as seen in the validation loss plot in Figure 14. We achieved a validation loss of 42.4464225769043 and a training loss of 37.63188280936999 for the best model at 20 epochs.

Now, to get the EPE, as we are using MSE loss, we need to take the square root for calculating the EPE. Following are the EPE scores on train, val and test for the supervised network.
Train EPE: 6.313
Val EPE: 6.515

The result of the supervised homography model gives pretty close results, as seen from the results in Figure 16 and Figure 17 in the test set. Which reinforces the belief that the model has been trained properly and converged.

C. Unsupervised approach:

The unsupervised approach consists of three main components Figure 18: the homography model, tensor DLT, and the spatial transformer. As mentioned in the paper, we implemented the tensor DLT, which gives us the homography matrix in a differentiable manner. Also, the spatial network was implemented using Kornia to warp the image and calculate

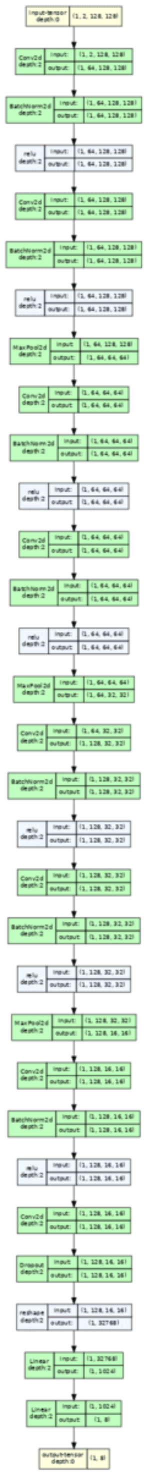

Fig. 15. Detailed Supervised Model architecture

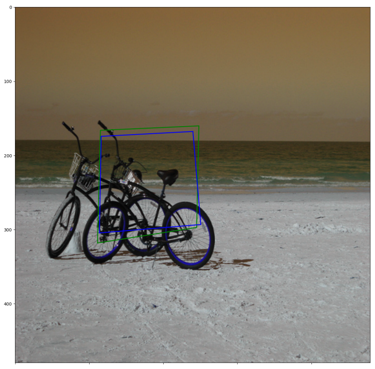

Fig. 16. Supervised Result (Image 26): Green ground truth, Blue prediction

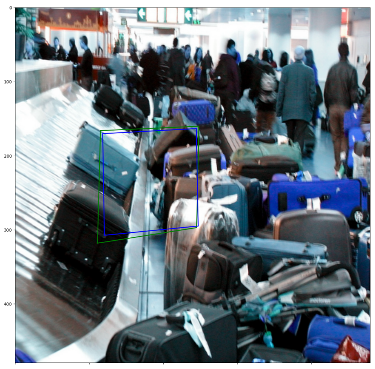

Fig. 17. Supervised Result (Image 31): Green ground truth, Blue prediction

the photometric loss function, which is the L1 loss function. Figure 19 showcases the model components, though it is to be noted that the spatial transformer is not visible in the summary as it was implemented using Kornia in the forward pass and hence doesn't show up in the summary.

We train the model using the L1 loss (photometric loss) function. The Adam optimizer is used with a learning rate of 0.0025 and a weight decay 0.0001. The model was trained on 15 epochs. In our case, the loss wasn't decreasing and was stuck at a constant around 0.43595412373542786, as seen in the validation loss plot in Figure 20. We achieved a validation loss of 0.43595412373542786 and a training loss of 0.43378835481504474. Since the model didn't converge, the results are not up to par with the supervised network, as seen in Figure 21 and Figure 22. We use the Full image instead of the patch in the spatial transformer layer as mentioned in the

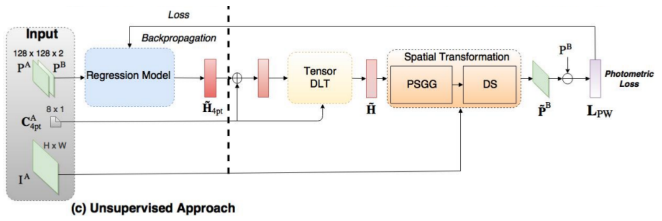

Fig. 18. UnSupervised Model architecture

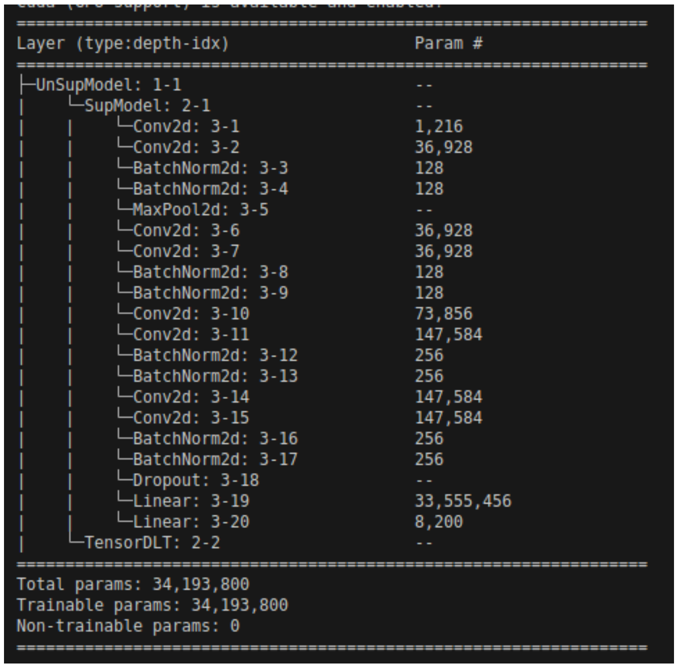
```
================================================================
Layer (type:depth-idx)                   Param #
================================================================
├─UnSupModel: 1-1                        --
|    └─SupModel: 2-1                     --
|    |    └─Conv2d: 3-1                  1,216
|    |    └─Conv2d: 3-2                  36,928
|    |    └─BatchNorm2d: 3-3             128
|    |    └─BatchNorm2d: 3-4             128
|    |    └─MaxPool2d: 3-5               --
|    |    └─Conv2d: 3-6                  36,928
|    |    └─Conv2d: 3-7                  36,928
|    |    └─BatchNorm2d: 3-8             128
|    |    └─BatchNorm2d: 3-9             128
|    |    └─Conv2d: 3-10                 73,856
|    |    └─Conv2d: 3-11                 147,584
|    |    └─BatchNorm2d: 3-12            256
|    |    └─BatchNorm2d: 3-13            256
|    |    └─Conv2d: 3-14                 147,584
|    |    └─Conv2d: 3-15                 147,584
|    |    └─BatchNorm2d: 3-16            256
|    |    └─BatchNorm2d: 3-17            256
|    |    └─Dropout: 3-18                --
|    |    └─Linear: 3-19                 33,555,456
|    |    └─Linear: 3-20                 8,200
|    └─TensorDLT: 2-2                    --
================================================================
Total params: 34,193,800
Trainable params: 34,193,800
Non-trainable params: 0
================================================================
```

Fig. 19. Detail UnSupervised Model architecture

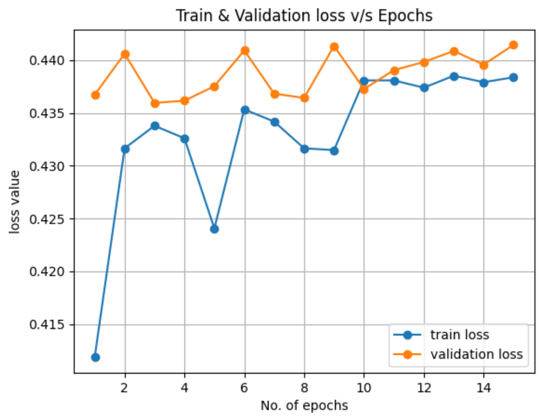

Fig. 20. UnSupervised Train and Validation Loss

paper.

Since the model didn't converge, we didn't calculate the EPE for the supervised model. But to calculate the EPE for the unsupervised approach, we would need to take the H4Pt from the Homography model, add it to the original corners, and calculate it against the ground truth we saved during data generation.

The images in the Phase 2 Pano set are very close to each other in FOV because the model was trained on the same warped patches; hence, if there is a drastic change in two consecutive images, the homography won't be accurate.

We also tried to stitch the Phase2 pano images using the tomography from the supervised model. First, we resized the image to 128*128, which is the model input size. We computed the homography but had trouble scaling the homography matrix for stitching. The output size kept exploding. Figure

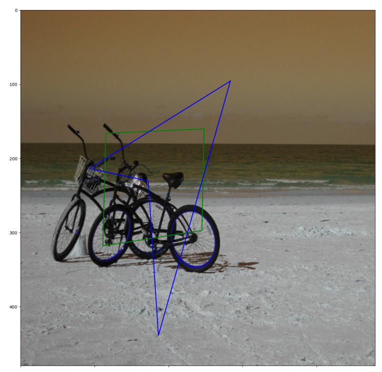

Fig. 21. UnSupervised Result (Image 26): Green ground truth, Blue prediction

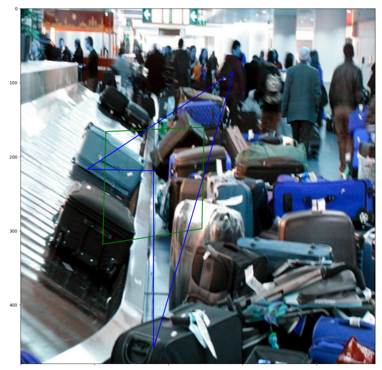

Fig. 22. UnSupervised Result (Image 31): Green ground truth, Blue prediction

23 shows an example for the image stitched with supervised homography.

III. Conclusion

The classical approach works well and was successfully implemented, which has a robust pipeline to reject images with low feature matches. The supervised model was trained successfully and converged in 20 epochs. The model output was also pretty accurate, but we need to fix the scaling of the homography matrix due to resizing to stitch the panoramas properly. The unsupervised model loss wasn't reduced, implying issues with the model itself. We tried various hyper-parameters as well as optimizers, but we weren't able to train them properly.

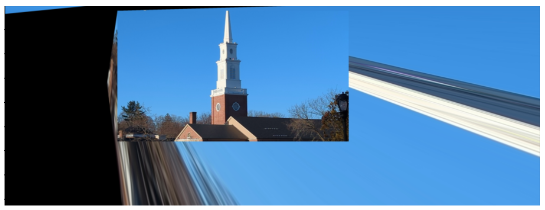

Fig. 23. Supervised stitching failure