Project 1 - My AutoPano

Karthik Mundanad
Robotics Engineering Department
Worcester Polytechnic Institute
Email: krmundanad@wpi.edu

Kushagra Srivastava
Robotics Engineering Department
Worcester Polytechnic Institute
Email: ksrivastava1@wpi.edu

***Abstract*—This report presents our implementation of classical and deep learning methodologies for homography estimation and image stitching. Phase 1 deals with classical approaches wherein we first discuss Feature extraction and matching and then outlier removal. Finally, we conclude by presenting our results for warping and blending. In Phase 2 we discuss supervised and unsupervised methods for homography estimation. We present results for comparison and thorough analysis for the same.**

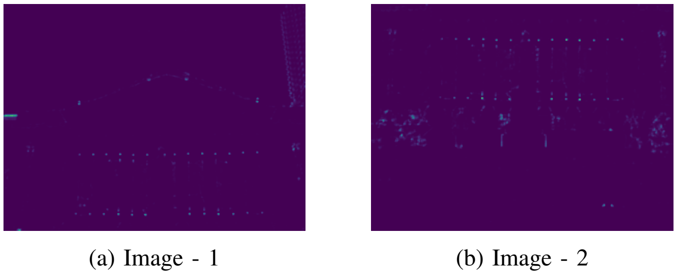
I. Phase - 1 Traditional Approach

A. Methodology

The general pipeline for image stitching using classical approaches is as follows:

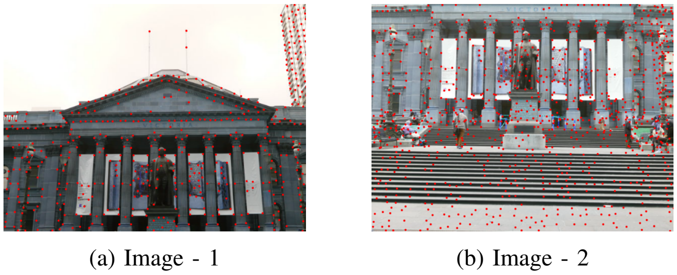
- Corner detection using the Harris Corner method (Section I-A1).
- Applying Adaptive Non-Maximal Suppression (ANMS) to obtain robust corners (Section I-A2).
- Describing each of these corner features by a feature descriptor (Section I-A3).
- Featuring matching across images using these feature descriptors (Section I-A4).
- Outlier rejection using RANSAC and estimating homography between two images using inliers (Section I-A5).
- Stitching and blending the original and warped images (Section I-A6).

1) Feature Detection: The Harris Corner (*cv2.cornerHarris*) function gives a score map indicating the probability of the pixel being a corner and the Shi Tomasi method (*cv2.goodFeaturesToTrack*) gives feature coordinates. We utilize the corner score map, take the local maxima, and then use non-maximal suppression (NMS) to get robust features. The result for the corner score map and Shi-Tomasi features are illustrated in Figures 1 and 2.


(a) Image - 1

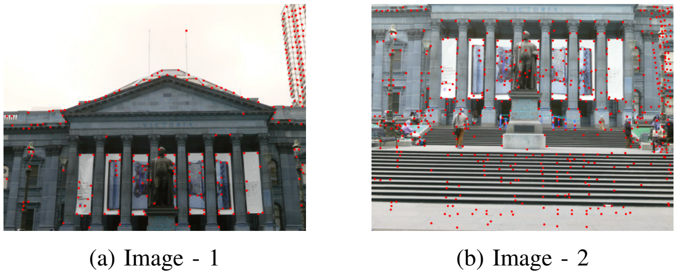
(b) Image - 2

Fig. 1: Corner Maps for Harris Corners


(a) Image - 1

(b) Image - 2

Fig. 2: Shi Tomasi Features


(a) Image - 1

(b) Image - 2

Fig. 3: Adaptive Non-Maximal Suppression on Harris Corners

2) Adaptive Non-Maximal Suppression: Adaptive Non-Maximal Suppression (ANMS) is a variant of NMS where the feature set is reduced to a fixed number by eliminating features that are too close to each other. This is implemented through an iterative search among the neighborhoods for each feature by measuring the euclidean distances between them. We pick out N (here 100) best features based on how spread out they are. The resultant image is shown in Figure 3.

3) Feature Descriptor: Each feature is assigned a unique feature descriptor which will be used by the feature matching algorithm. A patch of size 41×41, centered around the feature, is subjected to Gaussian Blur and subsampling which reduces the dimensions to 8×8. This reduced patch is flattened to form a 64×1 vector. Finally, we normalize the vector for illumination invariance.

4) Feature Matching: For feature matching, we employ a sum of squared differences (SSD) between feature descriptors corresponding to each feature. Iteratively, it yields the two best matches in terms of SSD. If the ratio of the best match to the next best match is above a certain threshold (here 0.8), then the match is rejected. The result is illustrated by Figure 4

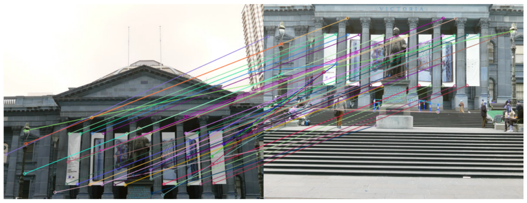


Fig. 4: Feature Matching

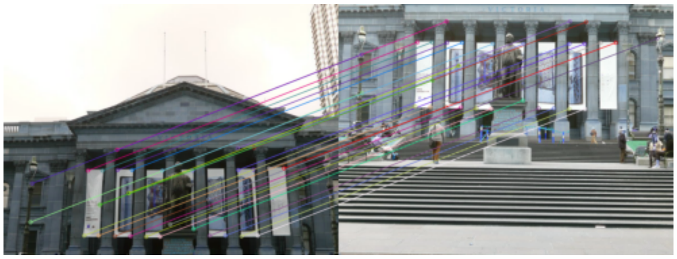


Fig. 5: Feature Matching after applying RANSAC

5) Outlier Rejection and Homography Estimation: Naive feature matching using SSD results in incorrect matches. To refine these matches, we use RANSAC. We sample M features (here 4) and estimate the corresponding homography matrix. We apply this homography to the points of the first image and compute the SSD between the features in the second image and the warped image. If it is above a threshold we reject it. The threshold dynamically increases if the number of inliers is below 4 after 10000 iterations. At termination, the best set of inliers is computed, and the corresponding homography matrix is computed using singular value decomposition (SVD). The set of inliers generated by RANSAC is illustrated in Figure 5.

6) Blending Images: The second image is warped using the homography matrix (using *cv2.perspectiveTransform*). To find the overlapping region, the width and height of the resultant image from the limits of the 1st image and the warped 2nd image are calculated. The homography and limits are then used in the *cv2.warpPerspective* to generate the stitched image. This function uses alpha blending (*cv2.INTER_LINEAR*) which is the weighted stitching of images.

B. Test Set Results

For Test Set 1 (see Figure 17), the chessboard pattern is too similar for the feature descriptor and matcher. This means even after removing the outliers via RANSAC with a high SSD threshold, the number of inliers is low. Thus stitching is not achieved. For Test Set 2, there are 9 unordered images. Computing inliers for each pair of images was done and the images were stitched for pairs with the highest number of inliers. This is computationally expensive and due to the large illumination changes, the inliers were low and images couldn't stitch properly. Figure 19 shows the resultant stitched image for Test Set 3. As the SSD threshold in RANSAC is dynamically increased, enough inliers are estimated for satisfactory stitching. In Test Set 4, two images do not belong to the scene. Our algorithm is not able to compute inliers between them and thus correctly discard them.

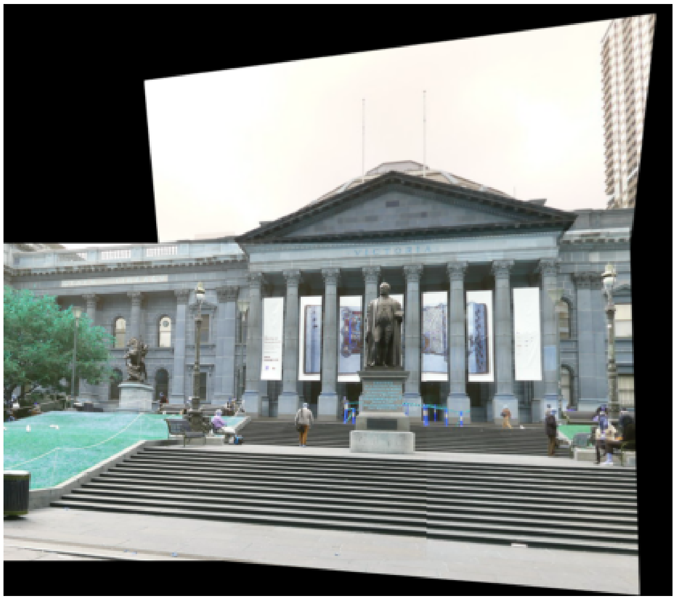


Fig. 6: Panorama Stitching for Set 1

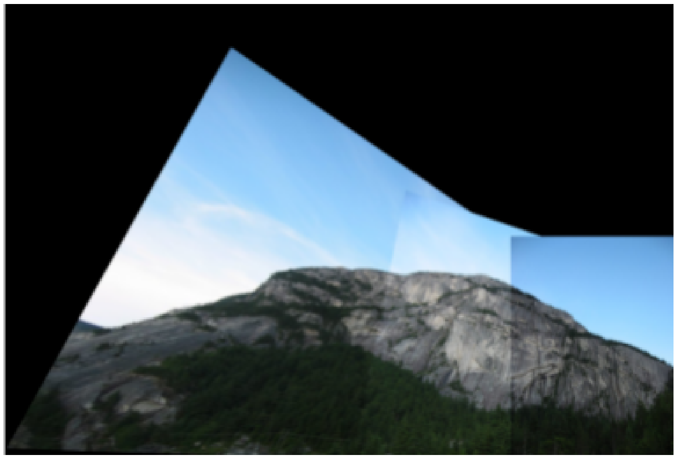


Fig. 7: Panorama Stitching for Set 2

C. Conclusion

In conclusion, our algorithm stitches and blends images satisfactorily if the images are well and equally illuminated and there are no repeating patterns.

II. Phase 2 - Deep Learning

In this section, we present our implementation of learning-based models for estimating homography between images. These robust models replace classical approaches for feature

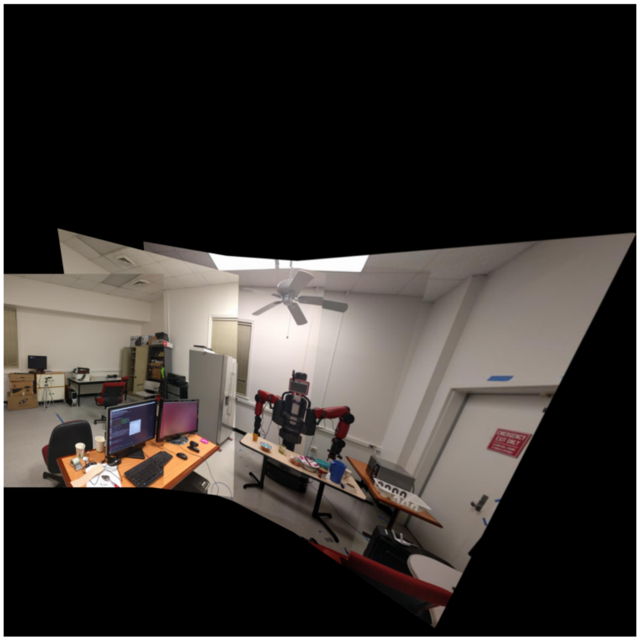


Fig. 8: Panorama Stitching for Set 3: For this set we selected the middle image as the reference and stitched it in left and right directions. The left and right stitched images were blended to form the resultant panorama.

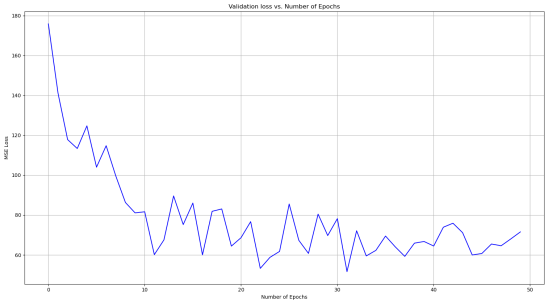


Fig. 9: Validation Loss vs Epochs for Supervised Learning

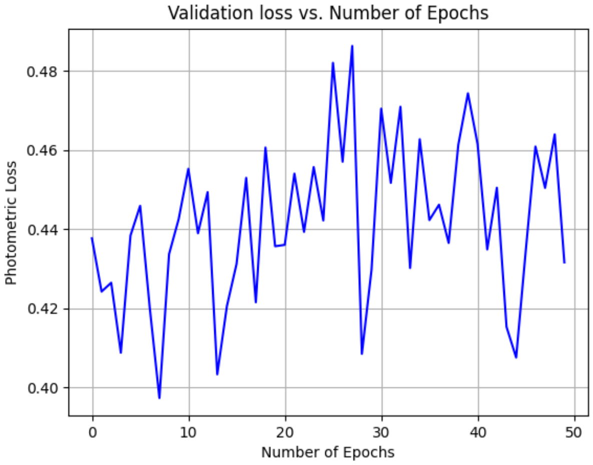


Fig. 10: Validation Loss vs Epochs for Unsupervised Learning

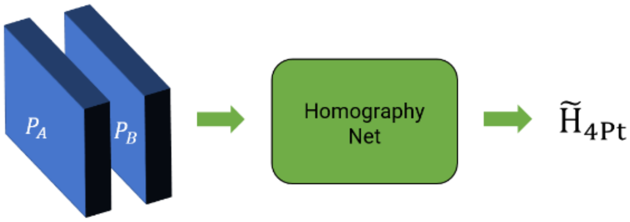


Fig. 11: Overview of the supervised learning methodology. Patches A and B, generated using the steps mentioned in II-A, are stacked along the channel dimension. The MSE loss function uses the ground truth H_{4pt} and the $\tilde{H}_{4pt}$ predicted by the network.

detection, matching, and homography estimation explained in the previous section. More specifically, we present our data generation (Section II-A), supervised model (Section II-B), and unsupervised model (Section II-C) in this section.

A. Data Generation

The data generation methodology is explained below.

- We define an active region as a patch obtained by cropping the image, resulting in dimensions $(H - 50) \times (W - 50)$, where H and W represent the height and width of the image, respectively. This cropping implies a border of 20 pixels on each side of the image.
- A patch P_a (of size 128×128) is generated from an image I_a such that the top left corner of the patch lies within the active region. This ensures that the corresponding warped patch does not have background padding.
- A random perturbation performed in the range $[-32, 32]$ to each corner to obtain a patch $P_a^{'}$.
- Since the corners of P_a (C_a) and P_a' (C_b) are known, we can calculate the homography matrix, H_{ab}, between the patches using the *cv2.getPerspectiveTransform* function.
- Instead of warping the patch we warp the image to generate the corresponding warped image, I_b. The corresponding warped patch P_b can be extracted from I_b using C_b.
- We store C_a, C_b, $H_{4pt} = C_a - C_b$ in a JSON file. Data is generated on the fly during supervised and unsupervised training. 10 patches were generated for each image in the MS COCO dataset [1]. The training and validation sets comprise 49800 and 10000 images respectively.

B. Supervised learning

For supervised learning, we train a CNN-based architecture, similar to [2]. The architecture is presented in Figure 12.

1) Training Details: The loss function is defined as the mean square error (MSE) loss between the ground truth H_{4pt}

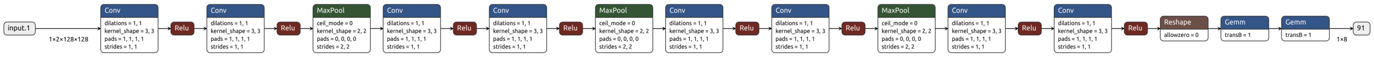
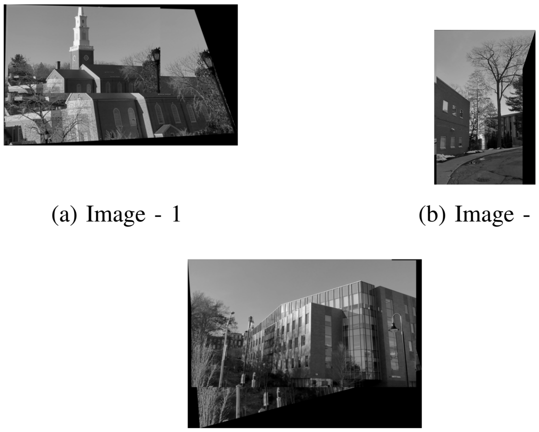


Fig. 12: Homography Net Architecture for Supervised Learning

and the $\tilde{H}_{4pt}$ predicted by the network. The hyperparameters and the optimizer chosen have been listed in Table I.

2) Results: A plot of the validation loss (MSE) against epochs is illustrated in Figure 9. We also report the EPE loss for training, validation, and testing as well the inference time (per image set) in Table II. The warped images with predicted and ground truth homography are presented in Figure 23. Sample results for panorama stitching is shown in Figure 13.



(a) Image - 1

(b) Image - 2

(c) Image - 3

Fig. 13: Panorama Stitching for 2 images for each of the test set for supervised learning: Images aren't stitched as pure translation between the images is not learned by the model. The resultant homography matrices are estimated incorrectly

Methodology	Learning Rate	Epochs	Batch Size	Optimizer
Supervised	0.0001	50	64	AdamW
Unsupervised	0.00001	50	32	AdamW

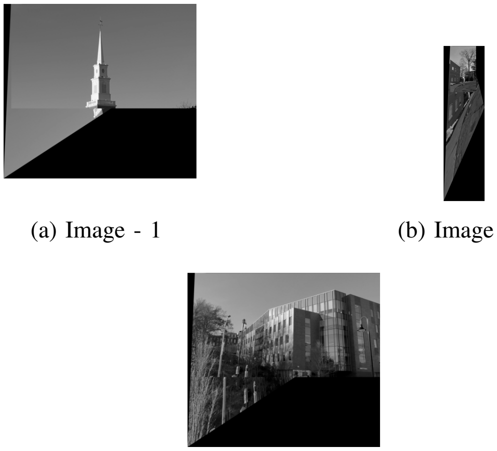
TABLE I: Optimizer and hyperparmeters for supervised and unsupervised approaches.

Methodology	Metric	Train	Val	Test
Supervised	EPE (pixels)	4.208	8.683	9.101
	Runtime (secs)	0.0525	0.0545	0.0542
Unsupervised	EPE (pixels)	24.942	25.431	20.331
	Runtime (secs)	0.0782	0.0793	0.0782

TABLE II: EPE and Runtime for supervised and unsupervised approaches.

C. Unsupervised Learning

For unsupervised learning, an architecture similar to [3] was adopted. The supervised network described in the previous section was used to predict $\tilde{H}_{4pt}$ from channel-wise stacked image patches. This $\tilde{H}_{4pt}$ was converted to $H \in R^{3\times3}$ through a differentiable Direct Linear Transform (DLT) layer. Differentiable warping was implemented using a Spatial Transformer Layer (STL) which further comprises of Parameterized Sampling Grid Generator (PSGG) and Differentiable Sampling (DS) (We referred to online resources to convert TensorFlow functions implemented in [3]). All these intermediate layers were implemented as described in [3]. A high-level diagram is illustrated in Figure **??**.



(a) Image - 1

(b) Image - 2

(c) Image - 3

Fig. 14: Panorama Stitching for 2 images for each of the test sets for unsupervised learning

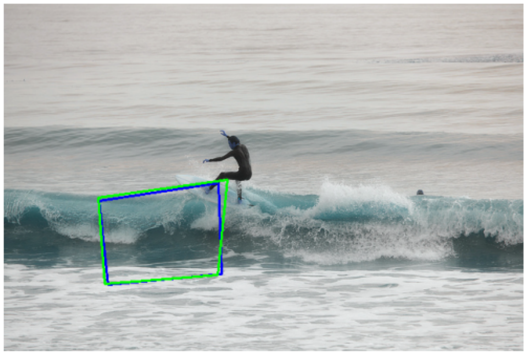


Fig. 15: Ground truth C_b indicated in green and Predicted C_b for supervised learning on sample Test image

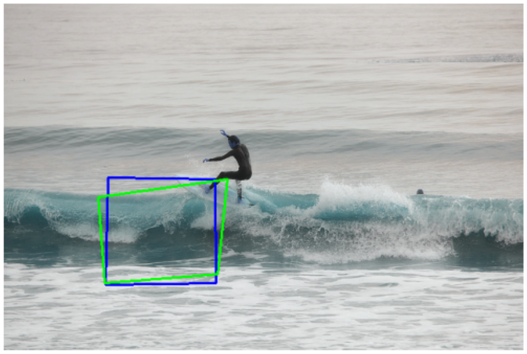
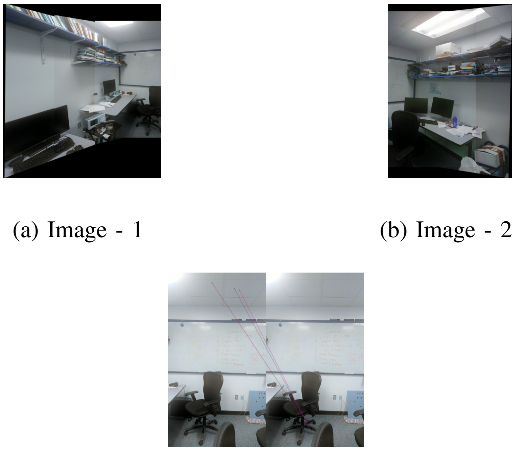


Fig. 16: Ground truth C_b indicated in green and Predicted C_b for unsupervised learning on sample Test image

(a) Image - 1

(b) Image - 2

Fig. 17: Panorama Stitching for Test Set 1: Image (b) indicates the wrong estimation of inliers due to repeating pattern



(a) Image - 1

(b) Image - 2

(c) Image - 3

Fig. 18: Panorama Stitching for Test Set 2: Image(a) and Image (b) indicate stitching of 2 separate subsequences. Image (c) indicates the wrong estimation inliers due to illumination change

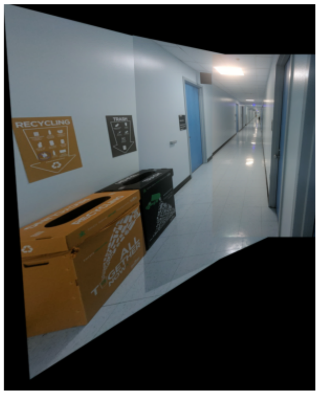


Fig. 19: Panorama Stitching for Test Set 3

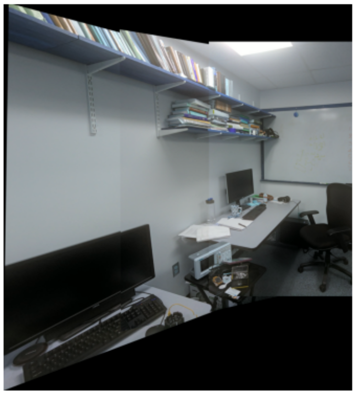


Fig. 20: Panorama Stitching for Test Set 4

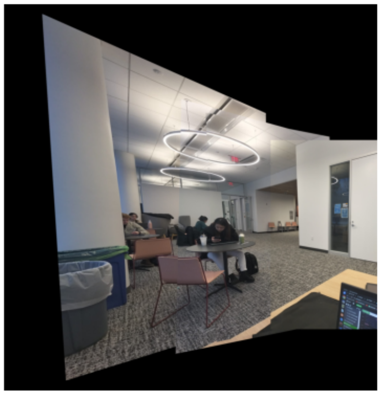


Fig. 21: Panorama Stitching for Custom Data 1

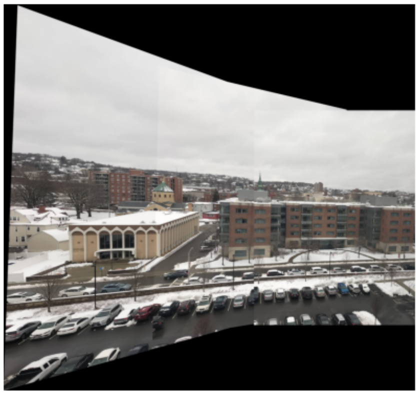


Fig. 22: Panorama Stitching for Custom Data 2

1) Training Details: The loss function is defined as the photometric loss between the ground truth patch P_b and the warped patch $P_b^{'}$ predicted by the network. The hyperparameters and the optimizer chosen have been listed in Table I. Note that we use a smaller learning rate because the loss was diverging to NaN after a few epochs.

2) Results: A plot of the validation loss (photometric) against epochs is illustrated in Figure 10. We also report the EPE loss for training, validation, and testing as well the inference time (per image set) in Table II. Finally, the warped images with predicted and ground truth homography are presented in Figure 16.

III. Conclusion

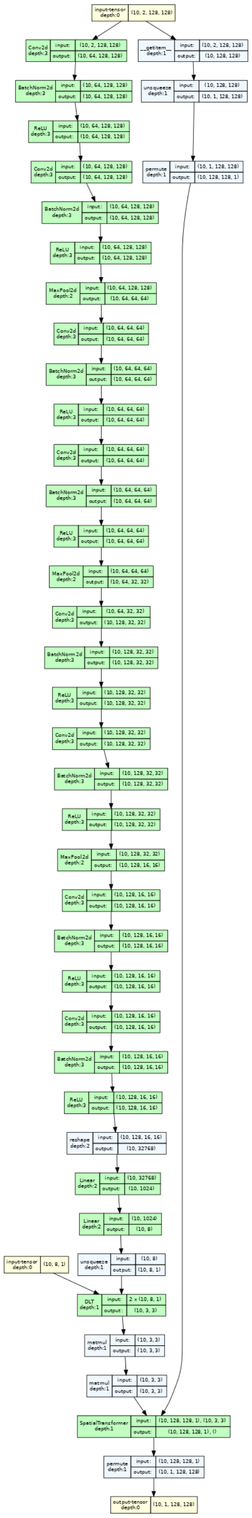
In conclusion, we have trained supervised and unsupervised models for homography estimation. As indicated by Table II, EPE for the supervised model is lower compared to the unsupervised model. As seen in Figure 16, the unsupervised model doesn't learn the homography matrices on the corners well. The results for panorama stitching for unsupervised learning are not presented. The panoramas stitched indicate that the model does not account for translations.

References

[1] T. Lin, M. Maire, S. J. Belongie, L. D. Bourdev, R. B. Girshick, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft COCO: common objects in context," *CoRR*, vol. abs/1405.0312, 2014. [Online]. Available: http://arxiv.org/abs/1405.0312

[2] D. DeTone, T. Malisiewicz, and A. Rabinovich, "Deep image homography estimation," *CoRR*, vol. abs/1606.03798, 2016. [Online]. Available: http://arxiv.org/abs/1606.03798

[3] T. Nguyen, S. W. Chen, S. S. Shivakumar, C. J. Taylor, and V. Kumar, "Unsupervised deep homography: A fast and robust homography estimation model," *CoRR*, vol. abs/1709.03966, 2017. [Online]. Available: http://arxiv.org/abs/1709.03966



input-tensor depth:0 (10, 2, 128, 128)
Conv2d depth:3 input: (10, 2, 128, 128) output: (10, 64, 128, 128)
__getitem__ depth:1 input: (10, 2, 128, 128) output: (10, 128, 128)
BatchNorm2d depth:3 input: (10, 64, 128, 128) output: (10, 64, 128, 128)
unsqueeze depth:1 input: (10, 128, 128) output: (10, 1, 128, 128)
ReLU depth:3 input: (10, 64, 128, 128) output: (10, 64, 128, 128)
Conv2d depth:3 input: (10, 64, 128, 128) output: (10, 64, 128, 128)
permute depth:1 input: (10, 1, 128, 128) output: (10, 128, 128, 1)
BatchNorm2d depth:3 input: (10, 64, 128, 128) output: (10, 64, 128, 128)
ReLU depth:3 input: (10, 64, 128, 128) output: (10, 64, 128, 128)
MaxPool2d depth:2 input: (10, 64, 128, 128) output: (10, 64, 64, 64)
Conv2d depth:3 input: (10, 64, 64, 64) output: (10, 64, 64, 64)
BatchNorm2d depth:3 input: (10, 64, 64, 64) output: (10, 64, 64, 64)
ReLU depth:3 input: (10, 64, 64, 64) output: (10, 64, 64, 64)
Conv2d depth:3 input: (10, 64, 64, 64) output: (10, 64, 64, 64)
BatchNorm2d depth:3 input: (10, 64, 64, 64) output: (10, 64, 64, 64)
ReLU depth:3 input: (10, 64, 64, 64) output: (10, 64, 64, 64)
MaxPool2d depth:2 input: (10, 64, 64, 64) output: (10, 64, 32, 32)
Conv2d depth:3 input: (10, 64, 32, 32) output: (10, 128, 32, 32)
BatchNorm2d depth:3 input: (10, 128, 32, 32) output: (10, 128, 32, 32)
ReLU depth:3 input: (10, 128, 32, 32) output: (10, 128, 32, 32)
Conv2d depth:3 input: (10, 128, 32, 32) output: (10, 128, 32, 32)
BatchNorm2d depth:3 input: (10, 128, 32, 32) output: (10, 128, 32, 32)
ReLU depth:3 input: (10, 128, 32, 32) output: (10, 128, 32, 32)
MaxPool2d depth:2 input: (10, 128, 32, 32) output: (10, 128, 16, 16)
Conv2d depth:3 input: (10, 128, 16, 16) output: (10, 128, 16, 16)
BatchNorm2d depth:3 input: (10, 128, 16, 16) output: (10, 128, 16, 16)
ReLU depth:3 input: (10, 128, 16, 16) output: (10, 128, 16, 16)
Conv2d depth:3 input: (10, 128, 16, 16) output: (10, 128, 16, 16)
BatchNorm2d depth:3 input: (10, 128, 16, 16) output: (10, 128, 16, 16)
ReLU depth:3 input: (10, 128, 16, 16) output: (10, 128, 16, 16)
reshape depth:2 input: (10, 128, 16, 16) output: (10, 32768)
Linear depth:2 input: (10, 32768) output: (10, 1024)
Linear depth:2 input: (10, 1024) output: (10, 8)
input-tensor depth:0 (10, 8, 1)
unsqueeze depth:1 input: (10, 8) output: (10, 8, 1)
DLT depth:1 input: 2 x (10, 8, 1) output: (10, 3, 3)
matmul depth:1 input: (10, 3, 3) output: (10, 3, 3)
matmul depth:1 input: (10, 3, 3) output: (10, 3, 3)
SpatialTransformer depth:1 input: (10, 128, 128, 1), (10, 3, 3) output: (10, 128, 128, 1), ()
permute depth:1 input: (10, 128, 128, 1) output: (10, 1, 128, 128)
output-tensor depth:0 (10, 1, 128, 128)

Fig. 23: Unsupervised model architecture