RBE/CS 549 - P1: MyAutoPano

Using 2 Late Days

Blake Bruell
Worcester Polytechnic Institute
Worcester, Massachusetts 01609
babruell@wpi.edu

Cole Parks
Worcester Polytechnic Institute
Worcester, Massachusetts 01609
cparks@wpi.edu

***Abstract*—In this paper we discuss the implementation of a tradition panorama stitching pipeline along with two deep learning approaches. We begin in Phase 1 by describing robust corner detection, feature matching, and image blending to achieve a stitched panorama, and in Phase 2 we outline supervised and unsupervised deep learning approaches to estimate the homography between perturbed image patches.**

I. Introduction

This paper is split into two sections: the traditional approach, and the deep learning approach. Each section details the technical aspects of the approach, as well as the issues our team ran across and our teams solutions. Some results are presented in each section, but more extensive results can be found in the Appendix.

II. Phase 1: Traditional Approach

In this section, we detail our implementation of the classic panorama stitching algorithm. First, corner detection is performed on the input images. Then, adaptive non-maximal suppression (ANMS) is used to select the best corners. Next, feature descriptors are computed for each corner. Then, feature matching is performed to find the best matches between the two images. Finally, RANSAC is used to remove outliers and estimate the homography between the two images. The homography is then used to warp the images and blend them together to create the final panorama. We also discuss methods for blending the images together to create a cleaner final panorama.

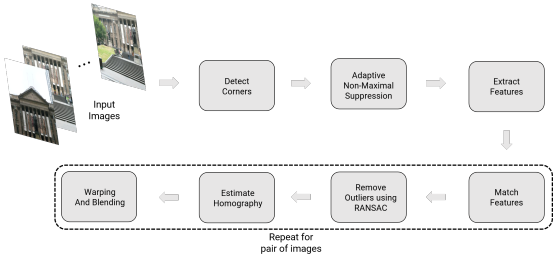
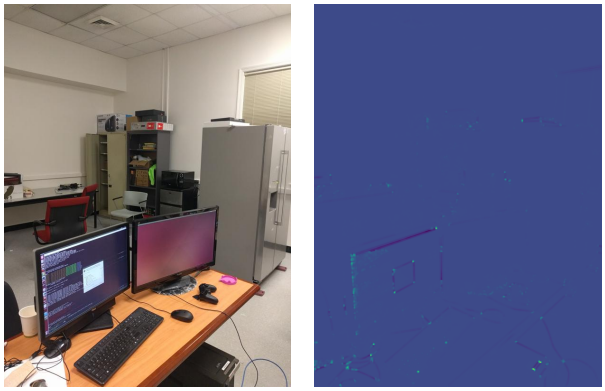


Fig. 1: Traditional Approach to Panorama Stitching

A. Corner Detection

To start the panorama stitching process, we first need to detect corners in the input images. In our implementation, the Harris corner detector is used to find corners in the images. Input images were converted to grayscale, and then the `cv2.cornerHarris` function was used to find corners in the image. The input and output of the Harris corner detector is shown in Figure 2b.



(a) Input image for Harris corner detector.

(b) Corner-score image from Harris corner detector.

Fig. 2: Corner Detection

B. Adaptive Non-Maximal Supression (ANMS)

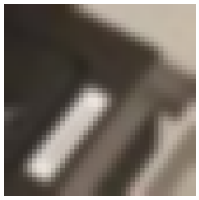
Once a corner-score image was generated, adaptive non-maximal supression (ANMS) was used to select the best corners. The ANMS algorithm takes in a corner-score image and

(a) Corner Peaks (*N=5000, MinDist=3*)

(b) Output (*N=500*)

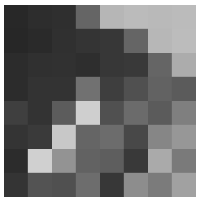
Fig. 3: Adaptive Non-Maximal Suppression (ANMS)

(a) Patch

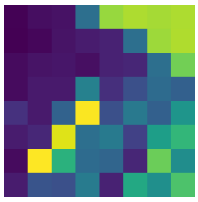


(b) Blurred

(c) Subsampled



(d) Grayscale



(e) Normalized

Fig. 4: Example Feature Descriptor Pipeline

outputs a list of the best corners. It first finds the local maxima in the corner-score image, shown in Figure 3a, which was performed with `skimage.feature.peak_local_max`. Then, for each local maxima, the distance to the nearest local maxima with a higher score was computed. The local maxima with the largest distance to a higher-scoring local maxima were added to the list of best corners. This ensured that the corners chosen are evenly-distributed across the image which in the end results in a more robust homography. The output of the ANMS algorithm is shown in Figure 3b.

C. Feature Descriptors

Once the best corners were selected, feature descriptors were computed for each corner. In our implementation, the feature descriptor is a 64x1 vector. To compute the feature descriptor, a 41x41 patch was taken around each corner. The patch was then blurred using a Gaussian blur with a kernel size of 5 and a standard deviation of 1. The blurred patch was then downsampled to 8x8, grayscaled, normalized, and finally reshaped into a 64x1 vector. An example of what the feature descriptor pipeline looks like for a patch around a specific corner is shown in Figure 4.

D. Feature Matching

Once feature descriptors were computed for each corner, feature matching was performed to find the best matches between the two images. A basic brute force matcher was implemented to achieve this task.

1) Brute Force Matcher: For each feature descriptor in the first image, the sum of squared differences (SSD) was computed between that descriptor and each descriptor in the second image. The best match is the descriptor in the second image with the lowest SSD, and the second best match is the descriptor in the second image with the second lowest SSD. If the ratio of the best match to the second best match was less than some threshold, the match was kept. This ensured that only the most unique matches were kept. A default of 0.7 was used for the threshold. The output of the feature matching algorithm is shown in Figure 5.

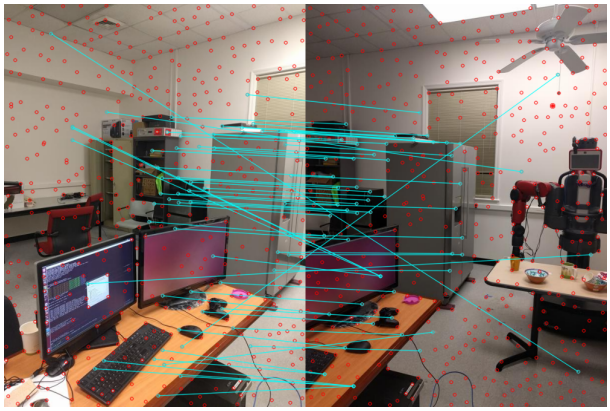
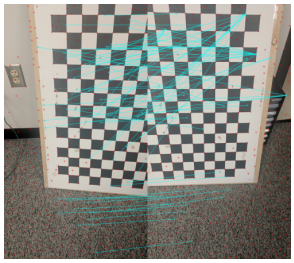
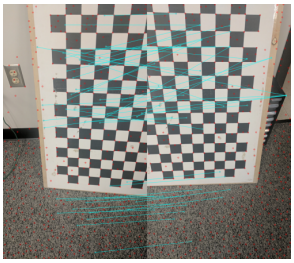


Fig. 5: Feature Matching Output (*threshold=0.7*)



(a) Without many-to-one filtering



(b) With many-to-one filtering (*N=3*)

Fig. 6: Many-to-One Filtering Example on Difficult Checkerboard Pattern

2) Many-to-One Feature Rejection: Sadly, this matching technique can fail catastrophically for certain inputs, and lead to what we will refer to as a **many-to-one features**. A many-to-one feature set is composed of a many features on a source image all forming matches with a *single* feature in the destination image. An example of a image with such features can be seen in Figure 6a. These sets are problematic as a trivial homography can lead to a many inliers when performing RANSAC, and so it is desirable to remove these anomalies. This was accomplished by removing all matches which are part of a many-to-one set above some threshold of N matches. The output of this filtering can be seen in Figure 6b.

E. RANSAC (Outlier Rejection and Robust Homography)

After naive feature matching, RANSAC was used to remove outliers and estimate the homography between the two images. The RANSAC algorithm takes in a list of matches, which correspond a keypoint from image 1 to a keypoint from image 2, and determines a homography which results in the maximal number of matches being included. It accomplishes this statistically, by randomly selecting four matches and computing the homography between the 4 keypoints from image 1 and image 2. Then, the keypoints in image 1 for each match are transformed using the homography, and the SSD is

Fig. 7: RANSAC Output

calculated for each match, between the transformed keypoint from image 1 and keypoint from image 2. RANSAC considers all matches which have an SSD below some threshold as inliers. Intuitively, a homography with many inliers is a homography which accurately aligns many matches.

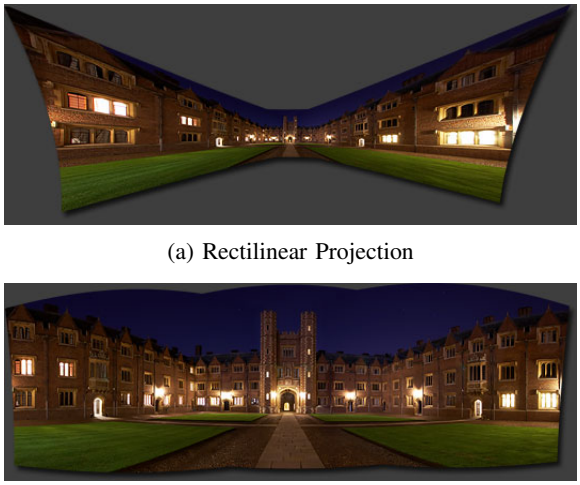
This process is repeated until the number of iterations exceeds a maximum number of iterations (in our case 2000) or until the number of inliers exceeds a minimum number of inliers. The homography with the most inliers is then returned. The output of the RANSAC algorithm is shown in Figure 7. Compared to Figure 5, the matches in the RANSAC output are much more accurate and there are no outliers.

F. Stitching and Blending Images

All of the steps leading up to this one are simply to find good homographies between image pairs. This is not sufficient for the production of a panorama, as the images must then be combined according to the calculated homographies. This process is actually quite involved, and easily as complicated as all the previous steps combined in implementation. As such, the process will be broken down into a few different sections.

1) Image Order and Outlier Images: The first aspect of the technique we will discuss is how outlier images and out of order images are dealt with. Before going any further, let us define what we mean by ”in order” or ”out of order” images. We define a of set of images as **in order** if they are presented in the order they should be stitched, and **out of order** if otherwise. Image sets can also contain images which do not belong in the panorama at all, regardless of how they are ordered. These images are henceforth referred to as **outlier images**. To address outlier images and out of order image sets our team implemented two techniques: image skipping, and a minimal path algorithm.

The first technique, image skipping, assumes that images are presented in order, but potentially with outlier images. The idea of the technique is simple: if an image doesn't have a sufficiently good homography with the previous image in the image set, it is skipped entirely. This yields good results on in order image sets with arbitrary outlier images. This doesn't deal with the case of out of order images, though. To deal with this case, the second technique must be used.



(a) Rectilinear Projection

(b) Cylindrical Projection

Fig. 8: Results of Different Image Projections [1]

In the second technique, we construct a directed graph in which each vertex is an image, and each edge is a homography between those images. To create this graph the homography calculation is performed on *every* pair of distinct images, in both directions, (which can get very expensive for large image sets). The directed edge weight between one vertex, image1, and another vertex, image2, is defined as the number of inliers yielded by RANSAC from image1 to image2 (note the order is important). If RANSAC does not converge, the edge weight is set to negative infinity. The *maximal path* is then calculated through this graph, which is defined as the path through the graph which contains any given vertex at most once (though it need not contain *all* vertices in the original graph) with the greatest sum of edge weights. This algorithm will never include create a path which cannot be stitched together, as all edges between images without good homographies have an infinitely negative cost, and will thus never be included. Due to the cost of this method, it must be enabled via an argument.

2) Cylindrical Warping: Thus far the assumption has been that the image sets presented need only a set homographies to correctly stitch them together. This is nominally true, but due to the rectilinear projection used by cameras stitching many images which cover a large FOV will result in undesirable behavior. This behaviour can be clearly demonstrated in Figure 8a, which demonstrates how without some form of projection correction, the images on edges of the panorama must be increasingly warped. This warp is not caused by some failure of the pipeline, but directly as a result of the rectilinear projection of images. This eventually causes highly distorted images, and in the worst case, will cause the program to abort due to overly large memory requirements. To correct for this we must use some other projection method.

The most common method for panoramas is cylindrical projection, as very often a panorama covers a large horizontal

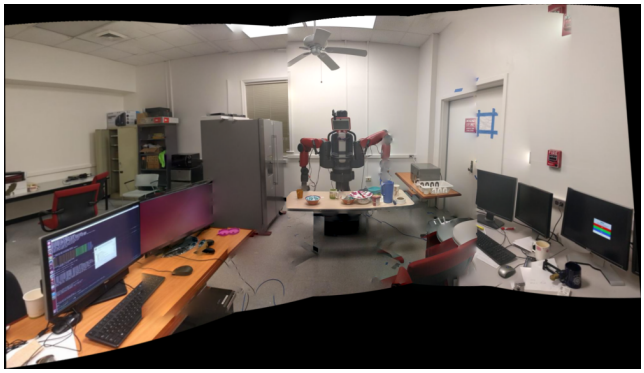


Fig. 9: Train Set 3 With Cylindical Warping (*start=1, cylindrical=750*)

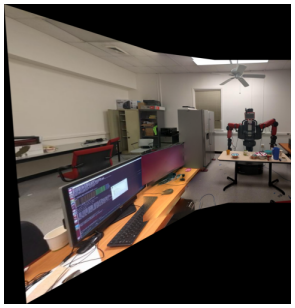
field of view, but comparatively narrow vertical field of view. The result of a cylindrical projection can be seen in Figure 8b. You may notice this is projection is a trade-off, since it does not preserve horizontal lines (but it does preserve vertical lines).

The warping is implemented as an image preprocessing step, as opposed to a post processing step, as otherwise the increased distortion will still cause memory usage issues. Since cylindrical projection is not always desirable (images must be aligned vertically, images must be rotated, panorama doesn't cover a large horizontal FOV, etc.), it is disabled by default, and must be enabled by providing the focal length of the images as a parameter. An example of cylindrical warping can be seen in Figure 9.

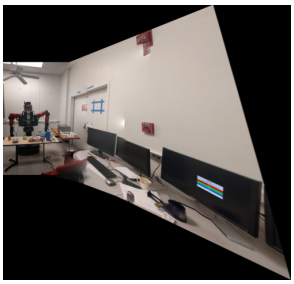
3) Stitching: Once an ordered list of images and the homographies between them is determined, the images are warped and blended together to create the final panorama. To warp the images, the left and right halves of the panorama were warped and blended independently, and then the two halves were combined to create the final panorama. The left set of images was recursively created by combining the right image with the warped left image, and then combining the next left with the partial panorama just created, until the entire left half of the panorama was created, as shown in Figure 10a. The right set was created in the same way, and then combined to create the final panorama.

4) Blending: To blend the images multiple techniques were explored, including mean value blending, maximum blending, and Poisson blending. We settled on Poisson blending as the default as it yielded the nicest blending for the image sets. The Poisson blending was performed using `cv2.seamlessClone`, using a the overlayed image as the source, and the result of maximum blending as the destination. This worked well at reducing seams, but did not correct exposure difference between images.

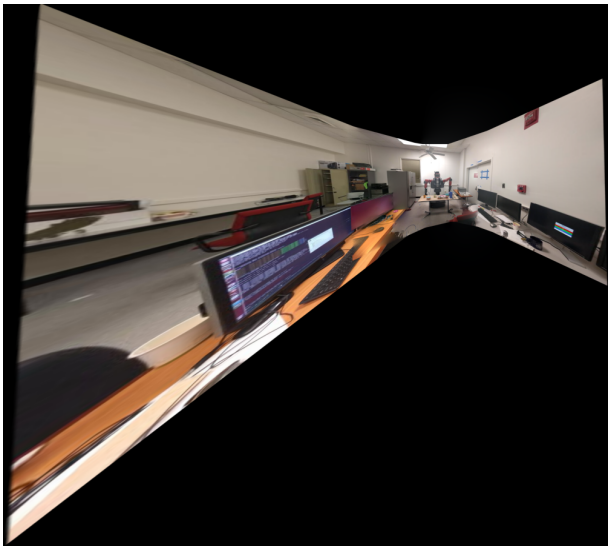
5) Results: The final panorama on Training Set 3 (minus the first image, due to the warping induced memory constraints discussed earlier) is shown in Figure 10c. The results from the rest of the training sets, custom sets, and test sets are shown in Appendix A.



(a) Left Stitch



(b) Right stitch



(c) Final Panorama Output

Fig. 10: Image Stitching Pipeline for Training Set 3

III. Phase 2: Deep Learning Approach

In this section, we detail our implementation of the deep learning approach to panorama stitching. We first discuss the data generation process, then we outline the supervised and unsupervised approaches to estimating the homography between perturbed image patches. Finally, we discuss the results of the deep learning approaches.

A. Data Generation

In order to generate data for the models to train on, we first need a pair of images with known homography between them. This is generally difficult to obtain, since a large number of corresponding images and correct homographies is required. Instead, we generate synthetic pairs of images to train the network. We use images from the MSCOCO dataset, which contains images of a variety of objects in natural scenes, and select a random patch out of the image. We then warp the original image using a random homography and extract the respective patch on the warped image. The steps outlined are as follows: 1. Obtain a random patch from the image such that all the pixels in the patch will lie within the image after warping the random extracted patch. 2. Perform a random perturbation in the range $[-p, p]$ to the corner points of the patch in the original image. We also add a random translation amount to the patch to ensure the network works for translated images as well. 3. Warp the original image with the inverse of the homography between the corners of the original patch and the perturbed patch. This gives us the warped image. 4. Extract the patch from the warped image using the corners of the original patch. This gives us the second patch. 5. Stack the image patches depthwise to obtain an input of size $MP \times NP \times 2K$, where K is the number of channels in each patch/image (one channel for grayscale images). The final output of data generation are these stacked image patches and the homography between them. These are then passed as inputs to the deep learning models, and the homography is used as the ground truth for supervised learning.

B. Supervised Approach

To train a supervised network to estimate the homography between a pair of images, we use the HomographyNet model outlined in Figure 11, as proposed by [2].

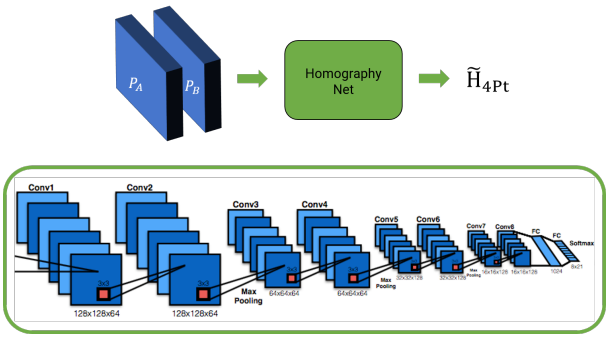


Fig. 11: HomographyNet Model Architecture

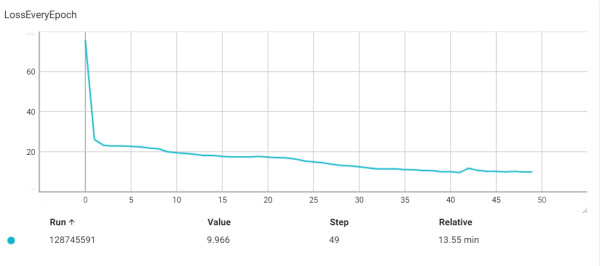


Fig. 12: Loss over Epochs for HomographyNet

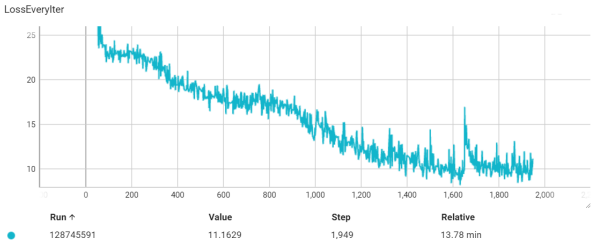


Fig. 13: Loss over Iterations for HomographyNet

We use the synthetic data generated in the previous section to train the network. The HomographyNet model is trained to regress the amount the corners of the patch need to be moved so that they are aligned with the second patch, which is denoted by H4Pt and is used as the labels for the model. The model is trained on the synthetic data, and the loss is calculated as the mean squared error between the predicted homography and the ground truth homography. The model is trained for a number of epochs, and the loss is monitored to ensure the model is learning. The model is then tested on a separate set of synthetic data to evaluate its performance. The network was trained for 50 epochs, with a batch size of 64, a learning rate starting at 0.005 and a weight decay of 0.0001. The EPE losses on the Train, Test, and Validation sets, along with their average inference times, are shown in Table 1, and the network structure is shown in Appendix B: Figure 26.

Additionally, the loss over epochs and iterations are shown in Figures 12 and 13 respectively.

C. Unsupervised Approach

The unsupervised model is trained in a similar way to the supervised version using the two input patches, but instead it tries to replicate the warp of the second patch from the first patch using the predicted homography. This way, the model does not learn what homography is, it just tries to recreate the warp performed between the two patches. The model is

Set	EPE Loss	Inference Time (s)
Train	9.7763	0.0020
Test	9.926	0.0010
Val	9.934	0.0020

TABLE I: EPE Losses and Inference Times for HomographyNet

Set	EPE Loss	Inference Time (s)
Train	493.49	0.0630
Test	492.62	0.0180
Val	493.12	0.0550

TABLE II: EPE Losses and Inference Times for Unsupervised Model

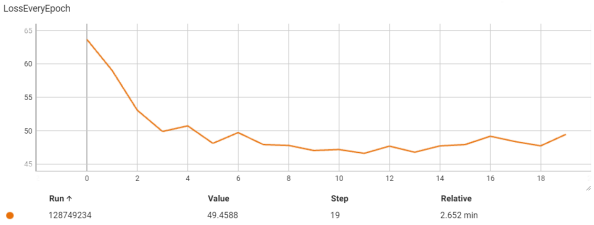


Fig. 14: Loss over Epochs for Unsupervised Model

trained to minimize the photometric error between the warped first patch and the second patch. In our implementation, the model was trained for 50 epochs, with a batch size of 64, a learning rate starting at 0.005 and a weight decay of 0.0001. The EPE losses on the Train, Test, and Validation sets, along with their average inference times, are shown in Table 2. Additionally, the loss over epochs and iterations are shown in Figures 12 and 15 respectively. Additionally, network structure is shown in Appendix B: Figure 26 with the addition of the TensorDLT and Spatial Transformer Network as proposed in [3]. Unfortunately, it appears as though the unsupervised model did not learn anything through training, as the EPE losses are very high and the model outputs are not accurate, always guessing the same homography. To try to combat this, the model was trained multiple times varying learning rate, weight decay, and batch size, but the model was not able to learn the correct homography.

D. Discussion

When comparing the outputs of the deep learning models to the actual perturbation between the two input patches, the supervised model performed much better than the unsupervised model. The supervised model was able to accurately predict the homography between the two patches, while the unsupervised model was not able to learn anything. This is likely due to the fact that the unsupervised model was not able to minimize the photometric error between the two patches, and thus was not able to learn the correct homography. The supervised model was able to learn perturbations between the two patches surprisingly well – photos with overlaid ground truth, supervised, and unsupervised homographies are shown in Appendix B: Figure 25. Even with large warps, the supervised model was able to predict the homography close to what the ground truth actually was very often. The last image was included to show a less-accurate prediction, but many of the predicted warps were closer to the ground truth, as shown in the first three images. This supervised model was trained for

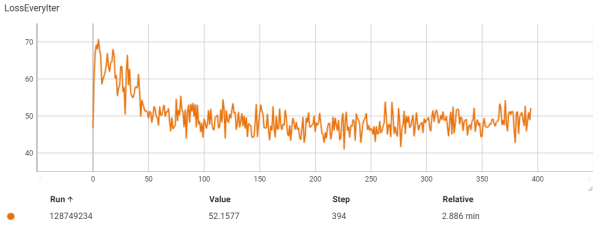


Fig. 15: Loss over Iterations for Unsupervised Model

50 epochs, but even training for as little as 20 epochs yielded comparable predictions.

One aspect of this project which was not implemented was the panoramic stitching using the deep learning models. This was due to the fact that the models did not map well to the task of panoramic stitching, as large translations were not part of training, and those are the main component of the homographies in a panorama. Secondly, the fixed image size of the models made it difficult to apply to images of arbitrary shape.

REFERENCES

[1] [Online]. Available: https://www.cambridgeincolour.com/tutorials/image-projections.htm.

[2] D. DeTone, T. Malisiewicz, and A. Rabinovich, *Deep image homography estimation*, 2016. arXiv: 1606.03798 [cs.CV].

[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*, 2018. arXiv: 1709.03966 [cs.CV].

APPENDIX A
RESULTS OF TRANDITIONAL PIPELINE

All panoramas were generated with default settings unless specified otherwise.

A. Training Sets

Fig. 16: **Final Pano for Train Set 1**

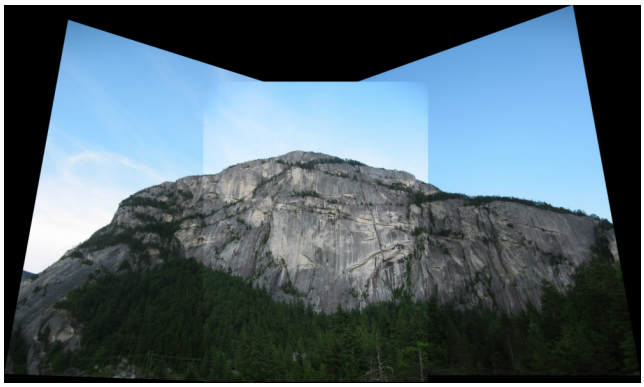


Fig. 17: **Final Pano for Train Set 2**

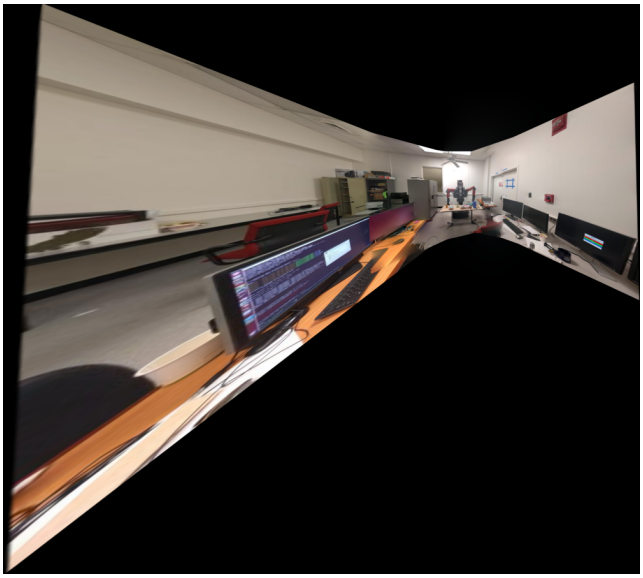


Fig. 18: **Final Pano for Train Set 3** (*start=1*)

B. Custom Sets

Fig. 19: **Final Pano for Custom Set 1** (*d=.5*).

Fig. 20: **Final Pano for Custom Set 2** (*d=.4, no-good-order*).

C. Test Sets

Fig. 21: **Final Pano for Test Set 1**. Note that the checkerboard pattern makes it incredibly difficult for the technique to find correct homographies, as features or not unique, so the result is not perfect. Despite this, the left and right halves stitched incredibly accurately.

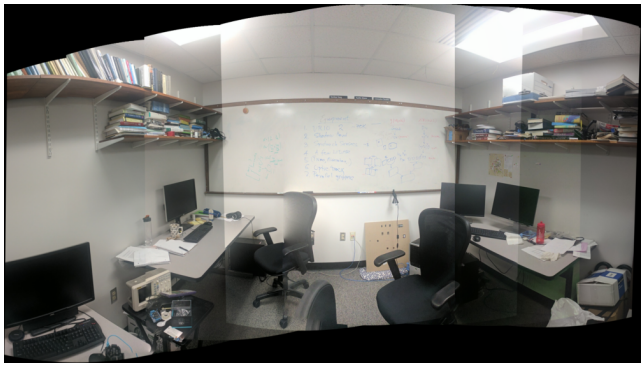


Fig. 22: **Final Pano for Test Set 2** (*cylindrical=950, no-good-order*). The automatic ordering algorithm is highly dependent on RANSAC, which itself is an inherently random algorithm, so it took multiple attempts to have all images included. The same parameters were used for each run.

Fig. 23: **Final Pano for Test Set 3**

Fig. 24: **Final Pano for Test Set 4**. Last two images are correctly ignored.

APPENDIX B
DEEP LEARNING APPROACH

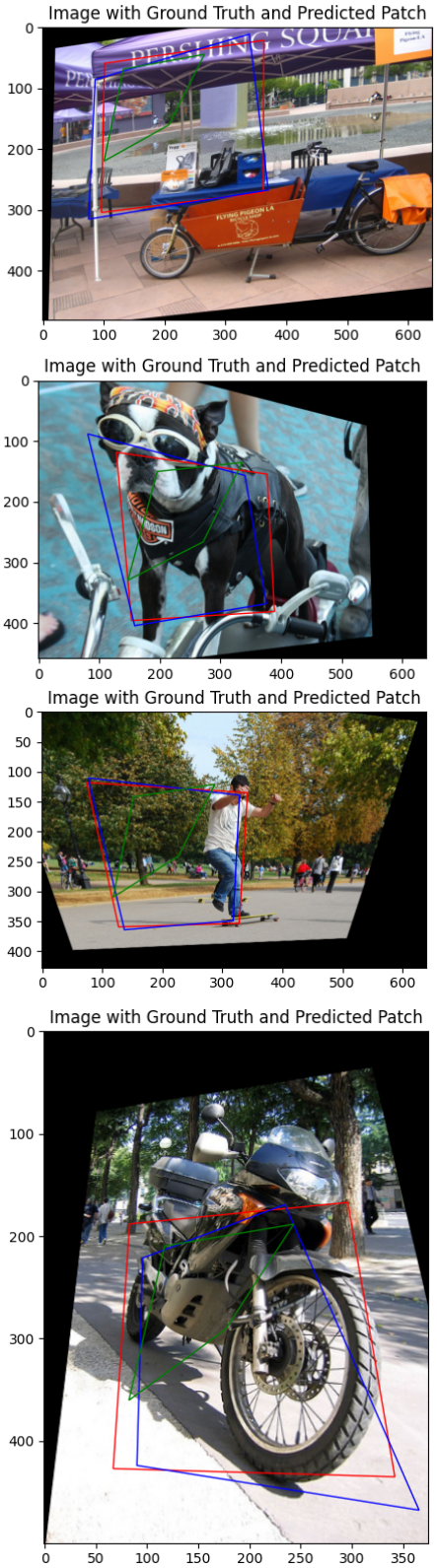


Fig. 25: Overlays of Ground Truth (Red), Supervised (Blue), and Unsupervised Homographies (Green)

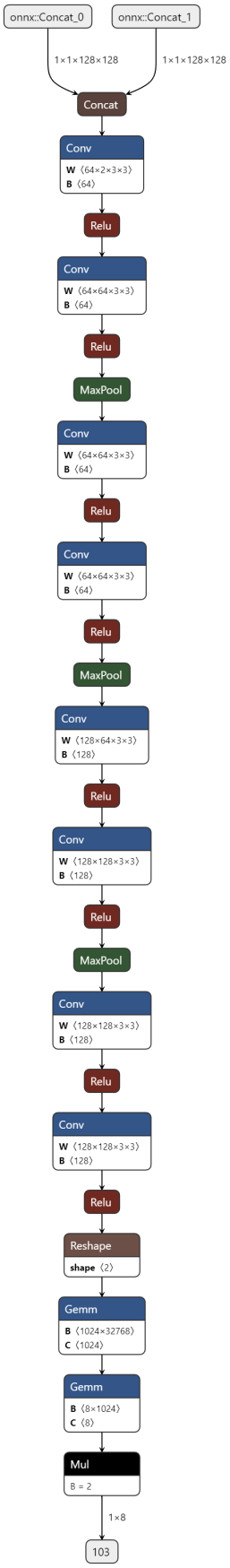


Fig. 26: HomographyNet Model Architecture Used in Both Approaches