Project 1: My AutoPano

Radha Saraf
Email: rrsaraf@wpi.edu

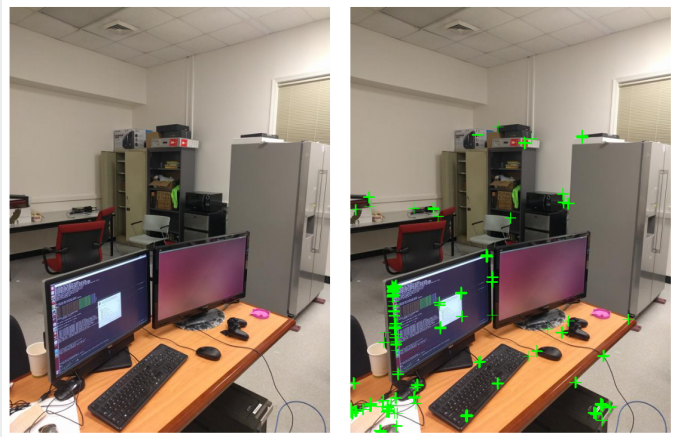
Sai Ramana Kiran Pinnama Raju
Email: spinnamaraju@wpi.edu
Using 2 late days

I. Phase I: Classical Approach

In this part of the project, we explore the classical methods to create a panoroma by stiching the image sequences. Each subsection explains in detail about the methodology followed and subsequent output of the images

A. Corner Detection

Idea is to draw relationship between how images using a set of features. Corners are best way to do that since they are visible from many different views. We can detect as many corners are possible from the given image and compare the features across the images. This comparison would tell us how the images are geometrically related to one another. For corner detection, we used OpenCV's Harris corners detection functionality. Following parameters are for corner detection are found to be optimal; kernel size 7, harris K parameter 0.04, harris sobel kernel size 11. Figure 1 show the corners detected across the images.



(a) Train Set3 2 (b) Harris Corners

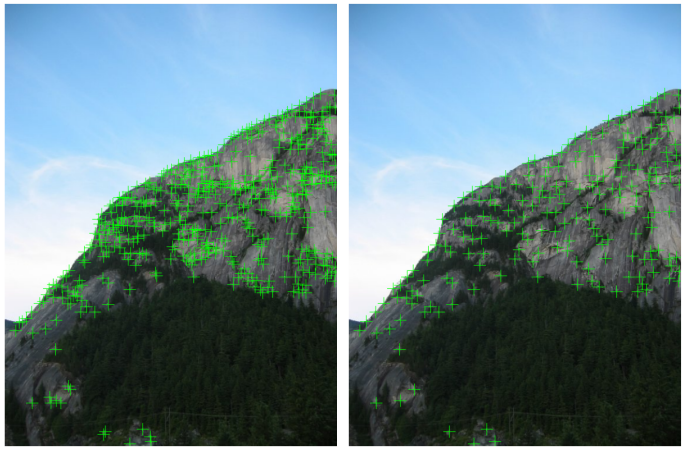
Fig. 1. image corners

B. Adaptive Non Maximal Suppression

Now that we have detected corners in each image, we need to find out "best" corners. Best corners are those which stand out among the local peer corners. Moreover, we want these corners evenly spread across the image so that we can get better homographies. To this end, we use Adaptive Non Maximal Suppression (ANMS) which does 2 parts

1) take local maxima over corners
2) consider only those corners which have a larger distance from relatively stronger corners

Point 2 is basically the main part of the ANMS which gives evenly spread out corners from a set of "clusters". Problem with harris corners or any other corner detector is that they detect a cluster of corners instead of just one corner. This makes sense since a corner is a set of pixels and based on resolution of the image many pixels can be accurately called corners. Even if we do local maxima of the image, the cluster might still persist. To circumvent this problem ANMS takes a point which is maximally distant to the other stronger corners and when we take sorted N_{best} corners we will be able to get a point from cluster. Figure 2 shows how the ANMS has decreased the cluster of corners to good corners.



(a) Test Set 2 Img 1 Harris Corners (b) ANMS output

Fig. 2. ANMS output

C. Feature Descriptor

We need to give each corner an "identity" in order to compare them across the images. This identity is called Feature Descriptor. Our appraoch to derive this unique identification is as mentioned in the problem statement

1) chosen corners only that can fit well within the dimensions of $(40, 40)$
2) flattened this sample into a 1D array
3) took pixels at every 25th index, idea is to take every 5th pixel rowwise or columnwise
4) finally reshaped into a (8×8) patch, standardized and blurred to make a smooth variation

One of the takeaways from this process of feature descriptor is how important the standardization is. Initially when we were playing around with the parameters, we removed the standardization and tried the feature matching that is described in the next step. The unnormalized patch didnt match well with the feature descriptors from other images. The reason being the contrast, intensity differences. The standardization is making the patch invariant of these intensity differences and making the patch more "comparable". A sample patch of an image corner can be seen in figure **??**

D. Feature Matching

The feature descriptors derived from the above process now needs to matched with other image feature descriptors. This matching is the important step since this is what tells us how much percentage of the images are overlapped with each other. The better and more accurate approach to reason about overlapping is ideally using photometric comparisons. But these direct methods are expensive in nature and hence we resort to using geometric features like above. In this case we compare geometric features like corners across the images. The algorithm for comparison is based on David Lowe's ratio test. From the above set of feature descriptors the naive and simple way would have been to minimize the 'distance' between feature descriptor vectors. The problem with this approach is that there cant be false positive that can minimize the distance and we might end up with bad set of matches. To prevent this the ratio test compares the first and second best feature descriptors distances. If the ratio is below certain threshold ϵ, we add the first best feature pair to the feature match set. Basically, when we select a match for a feature we need to make sure that the corresponding feature is adding value making the pair unique. In case the second best feature match is also within comparable distance of the first one, i.e it is above certain threshhold, then first feature correspondence is not truly unique and thereby not adding any value. With this rationale, we basically discard the pair and move on. This is truly a simple and elegant way to test for false positives while matching features or any data for that matter. Figure **??**

Fig. 3. Feature Matches in CustomSet2

E. Random Sampling and Consensus(RANSAC)

Feature matches identified above are not truly correct and prone to outliers like any other data generated in the world. To identify which data is part of the set and which data is outlier, we need to create some sort of data model and fit the model and reason about them. Again, the naive way to do is see where the data points lie and manually prune those data points. Or programatically check standard deviation of the data scatter and select some quartiles of the standard deviation. Although these methods are not wrong and not bad, they need lot of tuning and doesnt generalize well across the data points. Instead, to achieve this, we perform RANSAC, Random Sample Consensus. This robust method gives a probabilistic reasoning on creating a dataset without outliers [1]. The idea is that we want to identify the minimum data which give us best fit to our model and generalize well for all the other data points. In our case, model is the Homography estimation given a pair of images. We follow the RANSAC algorithm and get a good feature matches as shown in figure 3. Clearly, we can see that some of the false positive matching across the towers of the building is removed after performing RANSAC.

Fig. 4. Feature Matches in CustomSet2 after RANSAC

F. Stitching and Blending

Once we get the relative homography between a pair of images, we need to stitch them together. We found that programatically, stitching a little harder and we had inefficient implementation of it. Below steps outline our approach to stitching

1) To simplify, let's say image 2 is being transformed with H and stitched to image 1
2) we first take the bounding box coordinates of the image 2
3) identify the final bounding box coordinates of image 2 after transformation
4) if the bounding box has negative coordinate element, we take minimum negative element along each coordinate axis and translate the transformed image 2 by T
5) Now, since the projected image 2 is translated by T, we perform the same translation to image 1 to match the frames in which they are depicted
6) Once they are in the same frame, we find the intersected polygon of overlapping images. This operation is computationally expensive and hoping to find efficient ways to do that using OpenCV functionality
7) We perform alpha blending giving different weights to each pair of the image and merge the overlapped part

The results of following the above steps for stitching can be seen in 5

Fig. 5. Set1 Stitching

G. Recognizing Panorama

After perfoming the above steps to all the possible pairs, it is high time to recognize how the panorama has to be created. Given N unordered images, there are $N!$ ways in which panorama can be created. Moreover, selecting the correct panorama with best overlap is another major task in this bruteforce approach. A little better way to do is to decrease the difficulty by making a ordered set of images and stitching them one by one at the cost of generalizability. Another better and more generalizable way is to create a DAG of image nodes showing the relationship of one image with another. We explore both the approaches and detail it below.

1) Ordered Stitch: To get things started and make sure tht our code is intact, we explore this method of ordereing the images and stitching them in sequential manner. Idea is simple,

1) Visually inspect the images and order the images in which they have to be stitched
2) Initialize the reference image as first image
3) Get the image correspondence between reference image and second image.
4) Stitch the reference image and second image based on the estimated homography
5) store the output of the stitch as reference image for subsequent updates
6) Store this image as a reference image for the third image and perform the above steps

Figure 6 shows the final result of this process. Although this method is rudimentary, it performs relatively better to get started. We observed following problems with this approach

1) Unable to find good feature correspondences in subsequent runs once the image is stitched. This is due to the loss in clarity of the image due to view point changes.
2) Even after optimizing the memory issues perspective transform is giving zoomed versions of pictures as shown in figure 7 due to incorrect homographies.
3) More technically, our code is not optimized well and memory crashes are occuring as the number of images to stitch is increasing

Fig. 6. Ordered Stitch of Train set 1

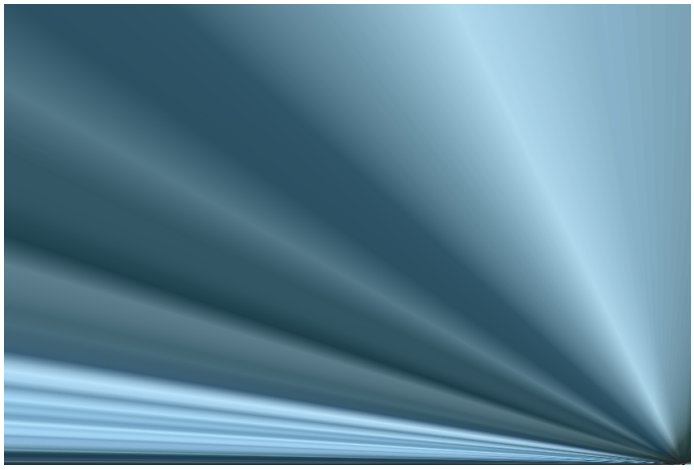


Fig. 7. Perspective transforms in Ordered Stitch

2) Image relationship Graph Construction: In this approach, we create a Directed Acylic Graph of images, which denote their relationship to one another. Here relationship or edges are created based on their number of features. It's more elegant and well generalized across the data sets. Below steps outline the procedure for the same

1) Create all the possible image to image correspondences within the given panorama
2) Store this information in a adjacency matrix where i, j element denote the number of features correspondences they share if image j where to transform to image i perspective. i.e. H_i^j
3) Now, perform a minimum number features thresholding to remove the edges which has less overlap

4) Fix directions such that we have only one directed edge to neighbouring nodes. This is done by taking the maximum feature route
5) In case the number of features are same from both the directions then we preserve the edge whose pointed node has more connections and more features. Essentially indicating that this node or image has better overlap with many other nodes/images
6) From here, we will end up a graph with one root node or many island graphs which are well connected
7) we take the graph which has maximum number of nodes and use the corresponding nodes for panorama stitching
8) the reference node will be the root node of the graph
9) homographies to the root from any node is computed by multiplying edge homographies as shown in equation 2

If H_i^j is the homography of transforming ith image onto jth image, and H_j^k is homography of transforming jth image onto kth image, then H_i^k is given by

$$H_i^k = H_j^k H_i^j \tag{1}$$

Generalizing it, for computing homography of ith image onto lth image connected by a list of nodes is given by

$$H_i^l = \prod_{k=i}^{l-1} H_k^{k+1} \tag{2}$$

It's a fairly involved and programmatically interesting process. Figure 8 outlines these steps in a pictographical manner. Figure 9. The problem in the image is that algorithm didnt account for the original image translation that happens every time stitching is done to offset the negative portion after stitching. Hoping to improve this in the future and show that this approach is indeed more robust and elegant than any brute force methods.

Results of test,train and custom set are dropped in the Phase I Results III

II. Phase 2: Deep Learning Approach

A. Data generation

The MSCOCO dataset with 5000 images was used to create the synthetic data used in the unsupervised and unsupervised approaches. As described in the problem statement, an active region is defined for every image. We chose this to be the central region of the image excluding 150 pixels worth of length on all four sides. Then, a patch of size 128x128 was chosen in the active region. This formed a part of the 'Orig' set. For the same patch we chose perturbations for all its four corners in the range [-32, 32]. The perturbed corners together with the original corners were used to calculate the homography. The inverse homography was then used to warp the original image and subsequently a patch was cropped from the warped image at the same location as the original image corners. This formed a part of the 'Warped' set. The patch is then translated by a stride of 32 pixels and the same process is repeated. Once the horizontal stride exceeds the limits, the patch is translated vertically downwards. This way based on the original image size we gather multiple pairs of data from one image. We end up with around 29000 image pairs from the original 5000 in this way. Both gray scale and color patches

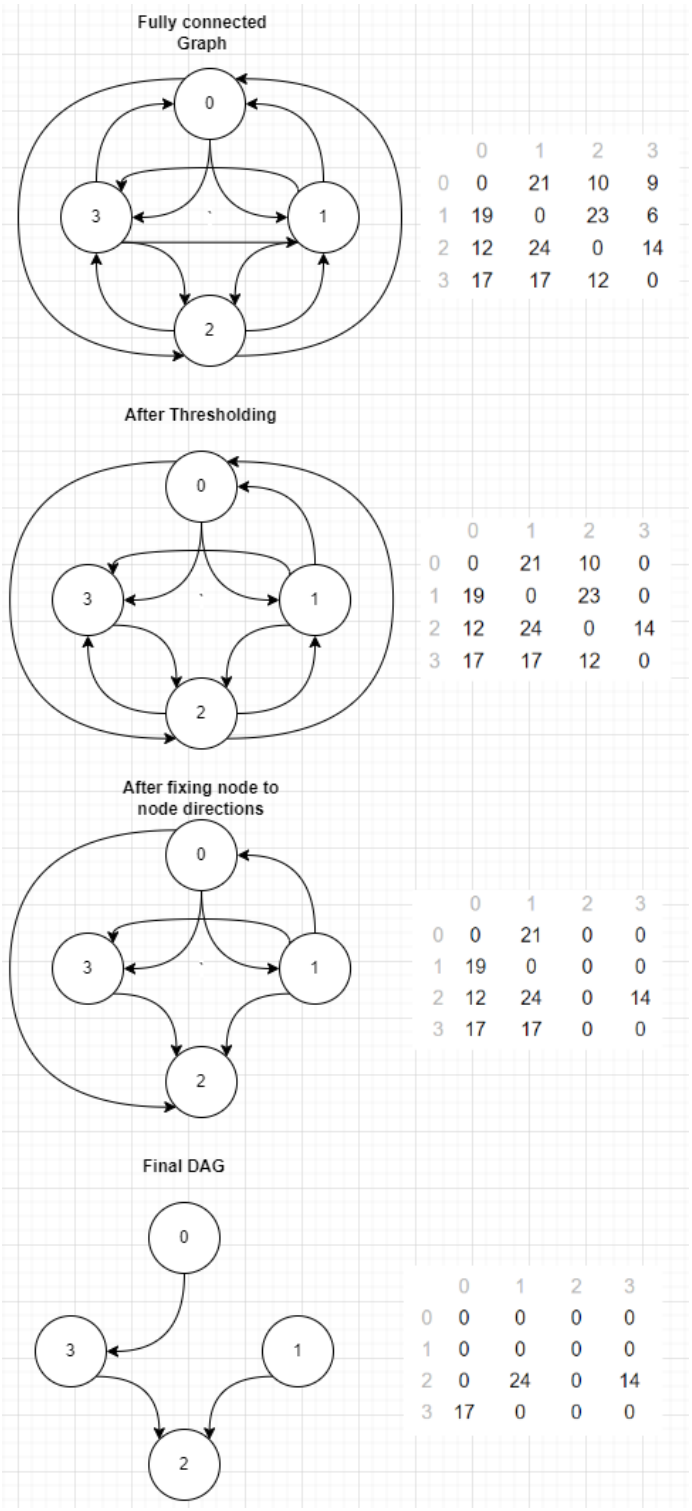


Fig. 8. TestSet1 DAG of images relationship along with adjacency matrix of feature edges

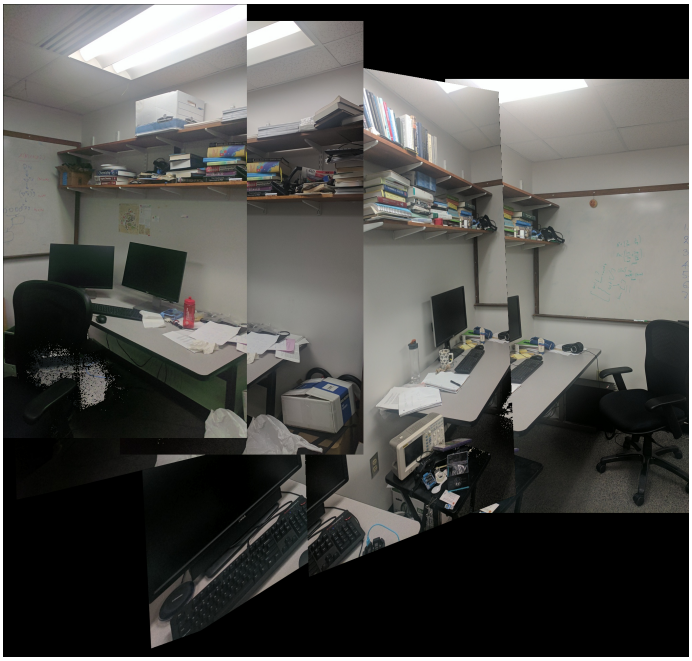


Fig. 9. TestSet2 output from graph based panorama stitching

were created to compare which one performed the best.

B. Supervised Approach

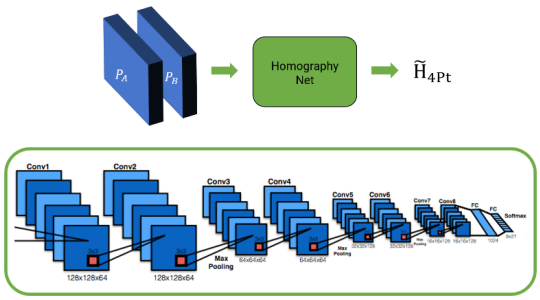


Fig. 10. Homography Net

The supervised deep learning approach uses the data generated using section 2A. We first used the same network architecture as used in the original HomographyNet paper. It is architecturally similar to Oxford's VGG Net and uses 3x3 convolutional blocks with BatchNorm and ReLUs. (see Figure 1). We use 8 convolutional layers with a max pooling layer (2x2, stride 2) after every two convolutions. The 8 convolutional layers have the following number of filters per layer: 64, 64, 64, 64, 128, 128, 128, 128. The convolutional layers are followed by two fully connected layers. The first fully connected layer has 1024 units. Dropout with a probability of 0.5 is applied after the final convolutional layer and the first fully-connected layer. The network takes as input a two-channel grayscale image sized 128x128x2. In other words, the two input images, which are related by a homography, are stacked channel-wise and fed into the network.

For this network architecture and data, we saw an exponentially decreasing loss on the train set whereas a constant one on the validation set. The learning rate was kept at 0.005

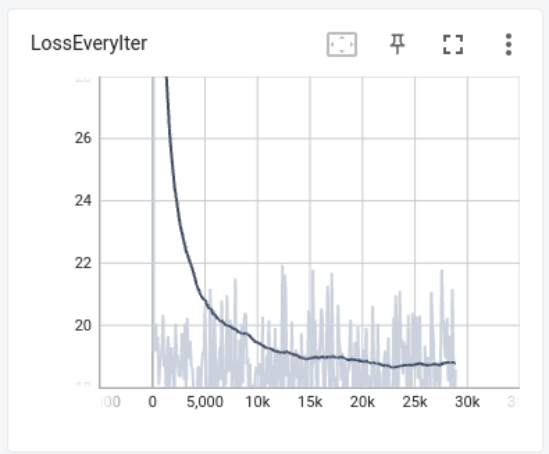


Fig. 11. HomographyNet: Training loss

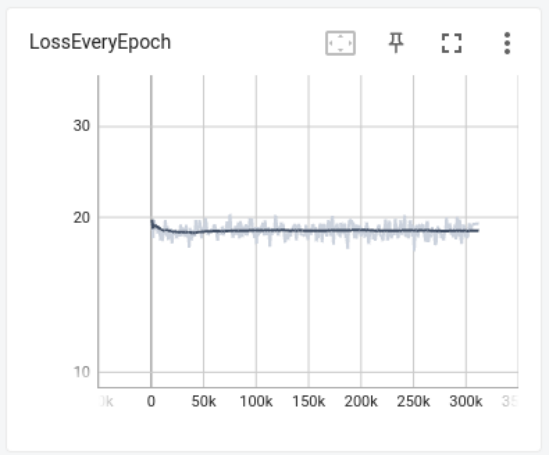


Fig. 12. HomographyNet: Validation loss

We changed the input to RGB images, so this time around the input was a 128x128x6 tensor formed by the two color images stacked channel-wise. The results were pretty much the same on this one as well.

We decided to increase the dropout between one of the later convolutional layers as a potential solution to the overfitting. We also tried changing the optimizer to AdamW which is believed to be better at generalizing than the Adam optimizer that we were previously using. Figure 13 shows a comparison between these two alterations. The dropout change performance is seen in blue whereas the dropout along with the optimizer change is seen in pink. Pink is clearly even worse. The learning rate was changed to 0.001 for both of these tests.

C. Unsupervised Approach

The unsupervised deep learning approach also uses the same data generated using section 2A and the same architecture as

Fig. 13. Network changes: Dropout, Dropout with AdamW

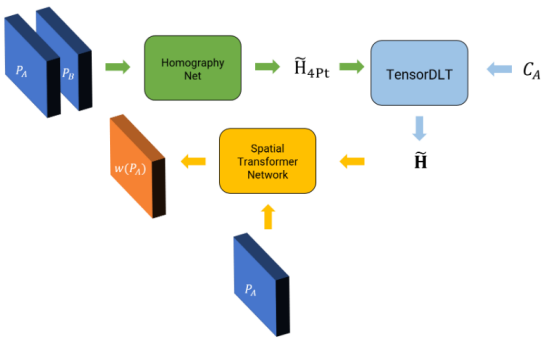


Fig. 14. Unsupervised overview

HomographyNet for generating the 4 point parametrisation of the Homography matrix, H4pt. A tensor direct linear transform (TensorDLT) formulation gives the 3x3 Homography matrix from the H4pt output out of HomographyNet. This layer has to be differentiable so as to have the gradients propagated through the network. We used functions from the pytorch library to create this layer. After obtaining the 3x3 homography matrix using tensorDLT, the original image is warped with this homography using a Spatial Transformer Network(STN) layer to obtain the warped image from the model. The STN is another differentiable layer in the network which facilitates the photometric loss calculation of the warped image against the ground truth input data.

Figures 15 and 16 show the training and validation losses for the unsupervised model. The model is shown in figure 17

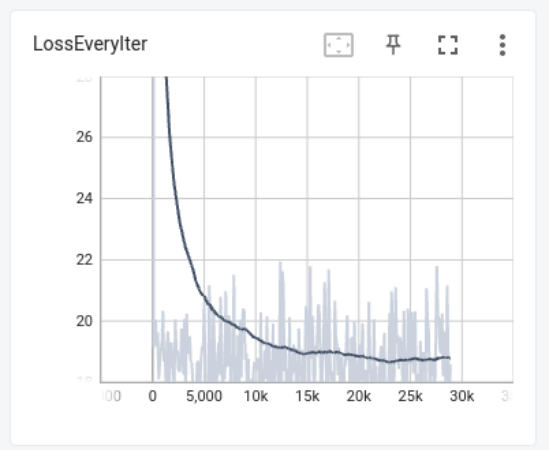


Fig. 15. Unsupervised Network: Training loss

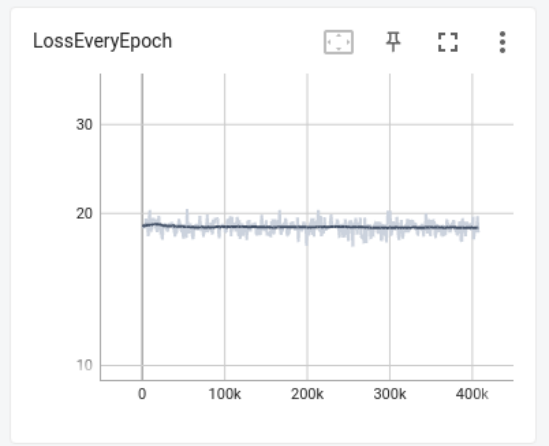


Fig. 16. Unsupervised Network: Validation loss

III. Phase I Results

1) Train set results: Figures 18, 19 and 20 show the outputs of Ordered stitch algorithm

2) Custom set results: Figure 21

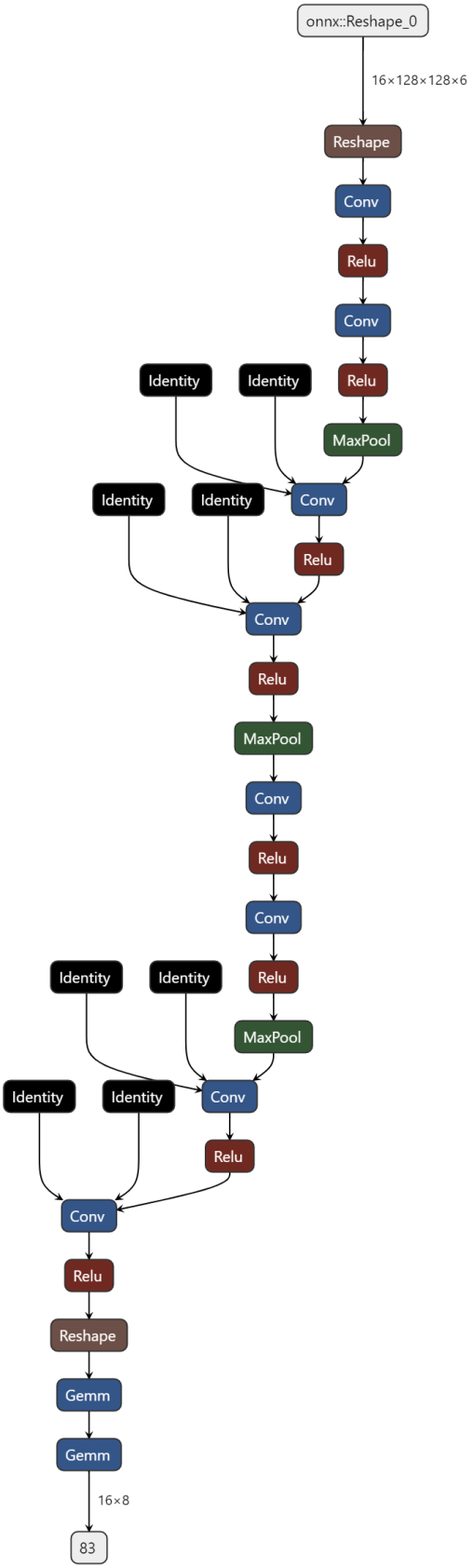


Fig. 17. Supervised and Unsupervised model

Fig. 18. Train Set I

Fig. 19. Train Set II

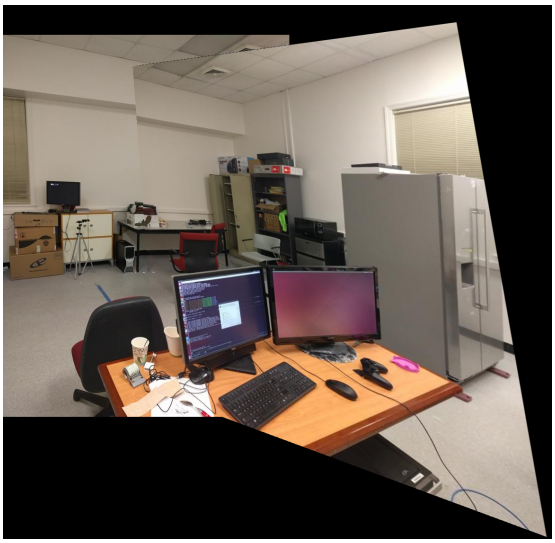


Fig. 20. Train Set III

Fig. 21. Custom Set I

A. *Test Set results*

Figures 22, 23 and 24 show the outputs based on Ordered stitch algorithm

Fig. 22. Test Set I

Fig. 23. Test Set II

Fig. 24. Test Set III

IV. Phase II Results

Figures 25, 26, 27 show the sample original image, the image warped using perturbations(ground truth) and the warped image using the model.

Fig. 25. Sample image

References

[1] https://www.youtube.com/watch?v=9D5rrtCC_E0ab_channel=CyrillStachniss

Fig. 26. Image warped using perturbed corners

Fig. 27. Image warped using model