CMSC733: Project 1 MyAutoPano

Varun Asthana
vasthana@umd.edu

Saumil Shah
sshah293@terpmail.umd.edu

I. PHASE 1: TRADITIONAL APPROACH

This section provides the details of the pipeline used to generate a panoramic view from multiple related images using classical approach. The classical approach involves the following steps: a) Corner detection and filtering by Adaptive Non-Maximal Suppression (ANMS), b) Generation of feature descriptors, c) Matching of features in a pair of images, d) finding best homography for the matched pairs, and e) stitching and blending of images.

A. Corner Detection and Filtering by ANMS

The very first step to find correspondence between any two pair of images is to identify local features in each image and then compare if features of first image matches with the features of second image.

Corners in the images are good feature points that can be compared. We have used the *cv2.goodFeaturesToTrack* function to find the feature points and their corresponding co-ordinates. Figure 1 shows the sample outputs of this function. Since the number of detected points are more than required, we thus filter the points based on its grayscale brightness and its distance from the other points. The method is called Adaptive Non-Maximal Suppression (ANMS). Results of filtered corners is shown in Fig 2.

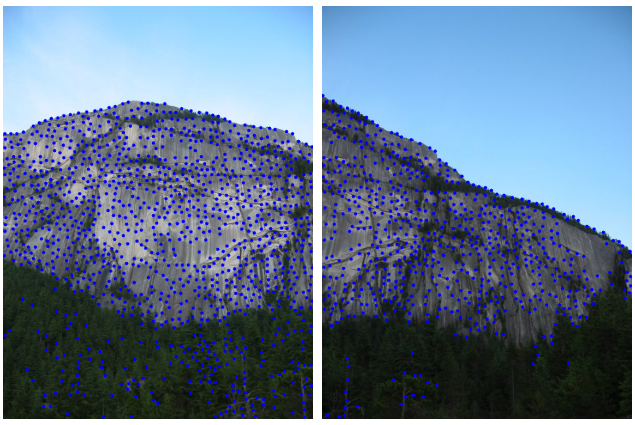


Fig. 1: Corner Detection

B. Feature Descriptors

Next step is to generate feature descriptors for the filtered corners. A patch of size 40 pixels x 40 pixels is taken around each detected corner (with corner being at the center of the patch) and then *cv2.GaussianBlur* is applied on the extracted patch. A padding of 20 pixels is also added on all the 4 edges

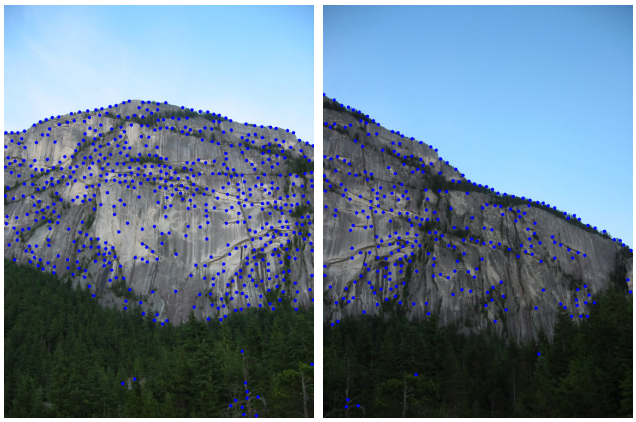


Fig. 2: Corner Detection after ANMS

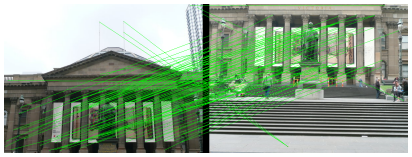
of the image in order to avoid any error in patch extraction if the detected corner is at the boundary of the image. The patch is then scaled down to a size of 8x8 and then reshaped into a row vector. This vector for a particular detected corner is termed as the feature descriptor for that corner. Output of the padded image after Gaussian blur is shown in Fig 3.

Fig. 3: Blurred Image for feature descriptors

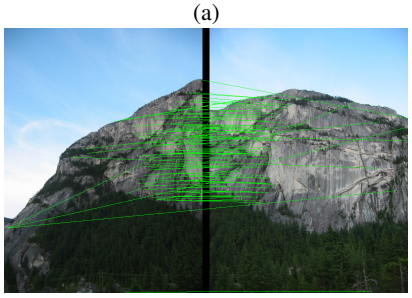
C. Feature Matching for a pair of Images

All the feature descriptors for an image are stored together and referred to as feature map. Once the feature map is available for all the images, we can now compare the feature descriptors after normalizing them. Features are matched by computing the squared sum difference (SSD) of the two 64×1 vectors (one from each image). For each feature descriptor of image 1, we maintain the minimum and the last to minimum SSD values. If the ration of minimum value to last to minimum value is less than 0.5, then the corresponding

features from both images are said to match with each other. Some results of feature matching are shown in Fig(4).



(a)

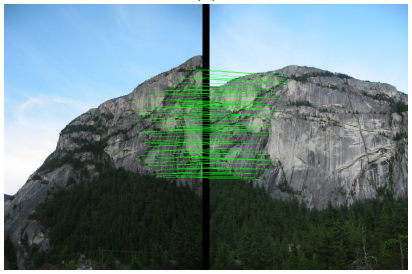


(b)

Fig. 4: Feature Matching

From the Fig 4 it is visible that many features are wrongly matched. These can be seen as the noise in the data of correct matches. These bad matches will affect our calculated homography matrix, and hence it is important to remove the noise from our data. A good technique to remove noise from a set of data is to utilize Random Sample Consensus (RANSAC). We randomly select a set of 4 corresponding matched points, compute homography matrix by using *cv2.getPerspectiveTransform*, compute SSD for other matched pairs and categorize the pairs as inliers or outliers against a threshold. Repeat the steps multiple times unless a minimum inlier ratio is achieved or if the maximum number of allowed iterations are completed. The results of matched pairs after applying RANSAC algorithm on the images in Fig 4 is shown in Fig 5.

(a)



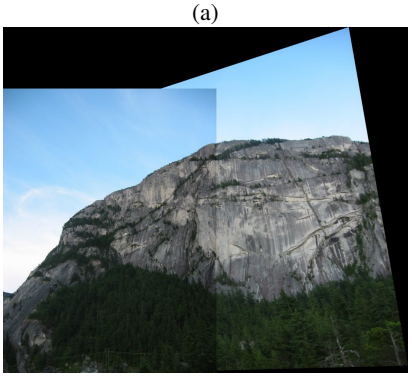
(b)

Fig. 5: Feature Matching after RANSAC

D. Image Stitching

All the correct matched pairs (inliers) after applying the RANSAC algorithm are then used to find the best homography between the 2 images. We re-compute the least-squares estimate on all of the inliers by using the function *cv2.findHomography*. The obtained homograpghy matrix is then used with the *cv2.warpPerspective* function to merge the 2 images. Results of warping and merging the images is shown in Fig 6. To achieve the output, image 2 was warped into the perspective of image 1 and then after resizing the output to correct dimensions, image 1 was superimposed on the warped image.

(a)



(b)

Fig. 6: Image Stitching

From the Fig 6 we can notice 2 issues, namely:

- Color Mismatch
- Black Patches in the final output
- We used the merged image and repeated the entire process to merge it with the next image. In doing so we encountered another issue of wrong feature matches as the warped image 2 is (at times) slightly rotated. An example of this error is shown in Fig 7.

E. Image Stitching Improvements

To get better results and improve homography, instead of warping image 2 into the perspective of image 1 we reversed this order and warped image 1 into the perspective

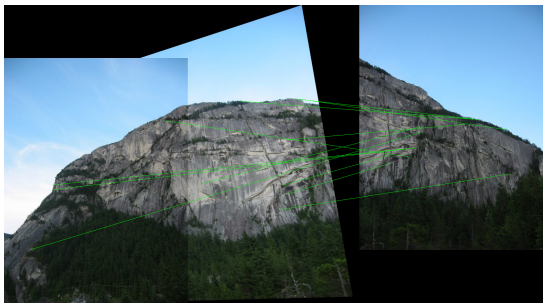
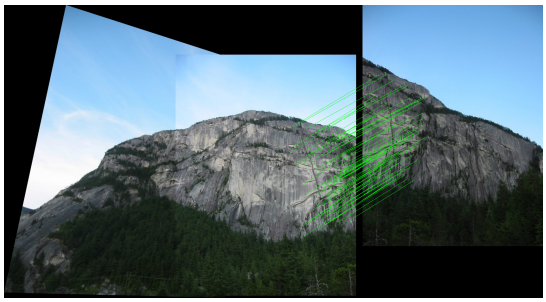


Fig. 7: Unable to get good homography due to less feature matches resulting from previously warped image 2

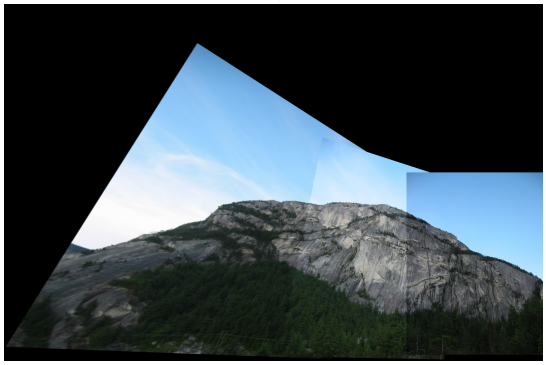
of image 2. This methodology produced good results, but is based on one **assumption**:

- All the input images provided are in sequential order, i.e. each image has good matching features with the next image in the data set.

Now since the image 2 is being superimposed upon the warped image 1, there were no black patches. Improved results are shown in Fig 8.



(a)



(b)

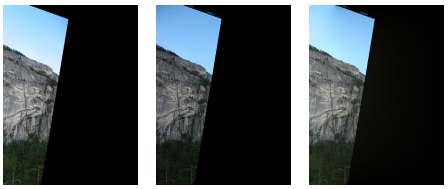
Fig. 8: Warping image 1 and superimposing image 2 resulting in no black patches, and improved feature matching

Pending issues:

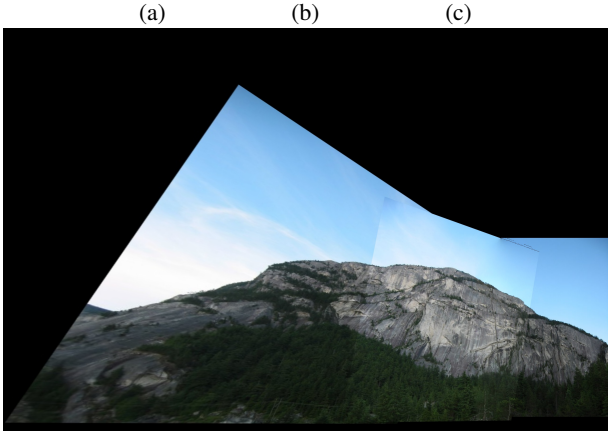
- From Fig 8 we can see that the only remaining issue is of partition line while superimposing image 2 on warped image.

Resolving Image Blending-

To resolve the issue of partition line on the merged images, we used the *cv2.seamlessClone* function. For good results, after warping we first obtained the common area in both the images and applied the function only over the common area.Image 2 was then superimposed on the warped image and then the common area was replaced with the output from the function *cv2.seamlessClone*.



(a) (b) (c)



(d)

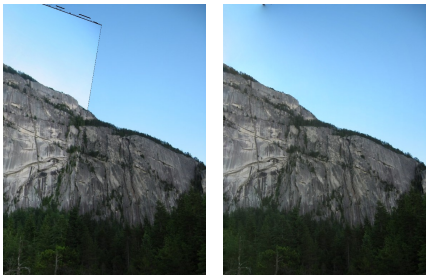
Fig. 9: Trial 1 for correct image blending. a) Common area in Warped Image, b) Common area in Image to be Superimposed, c) Output of cv2.seamlessClone function, and d) Final merged image

From the Fig 9c it was observed that the partition line of the image 2 over warped image was blended correctly, but the partition line of warped image (from background) became dominant. This partition line became dominant due to the black line boundary between the common area and the remaining image. To correct this, the remaining black area in Fig 9a,b was replaced with the image 2. Hence while blending both images, the common area was adjusted according to both the images, while the remaining area was not affected (as both images had same data of image 2).

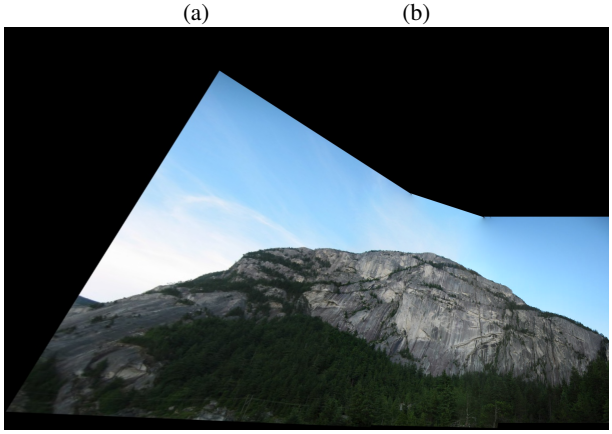
Using this approach of blending, we were able to remove all the partition lines giving a good overall Panorama of all the images.

F. Conclusion

- RANSAC algorithm is performed by randomly selecting a set of 4 points, it is not guaranteed that a



(a) (b)



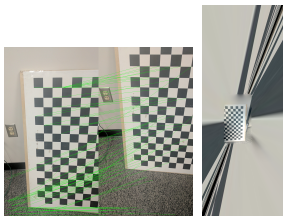
(c)

(d)

Fig. 10: Image Blending with no partition line, a) Source image for seamlessColne with common area from warped image and remaining area from image 2, b) Output of sealessClone function, c) and d) Final warped image super-imposed output from seamlessClone function

good output of all images will be obtained at once. Many multiple iterations were needed to extract good output where the inlier ratio was above 0.5. While we have used a loop in the code to check for a good inlier ratio for 8 times (unless minimum condition of 0.5 is matched). Irregular behaviour due to random selection of 4 matched coordinates for RANSAC is show in Fig 11.

- We were able to successfully complete the test set 3 and 4. In test set 4, all the non-related images were rejected by the algorithm. Results shown in Fig 12.



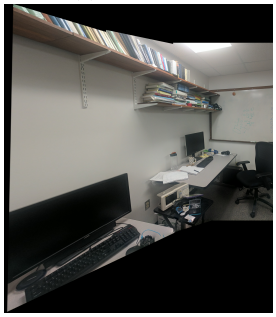
(a) Test Set 1

Fig. 11: Irregular Outputs

- With the assumption of sequential image data, if any image is not ordered correctly then the entire Panorama is affected. If any image coming in between, is not a good match, will be rejected. But this does not affect the sequence. The very first misplaced image is rejected while the next image does not match properly due to limited matching features. And hence all the images were then rejected. This was evident in the test data set 2. Result of test set 2 was similar to the result of test set 4.

- Feature descriptors are formed by considering a patch around the detected corners, hence the matching of images with very similar (or repetitive) data is not robust. This is evident in the test case 1 of chess board. This data-set also had similar features on the floor carpet.

(a) Test Set 3



(b) Test Set 4

Fig. 12: Test Set Outputs

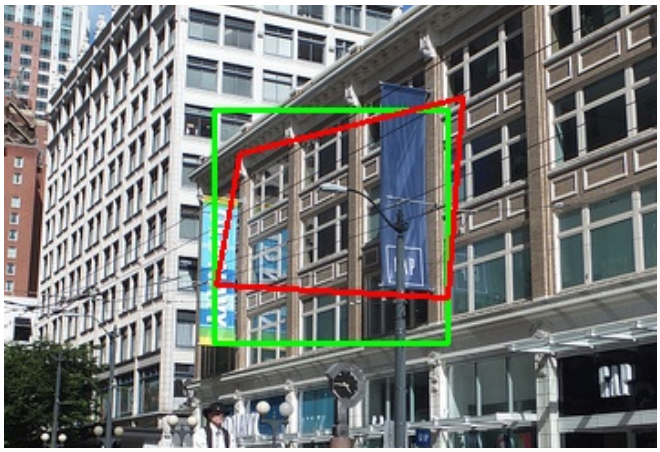


Fig. 13: Taking random patch from an image and creating a new quadrilateral with random perturbations on every corner of the patch. Green box indicates the randomly sleeted patch and red box indicates the randomly perturbed points taken to generate new homography.

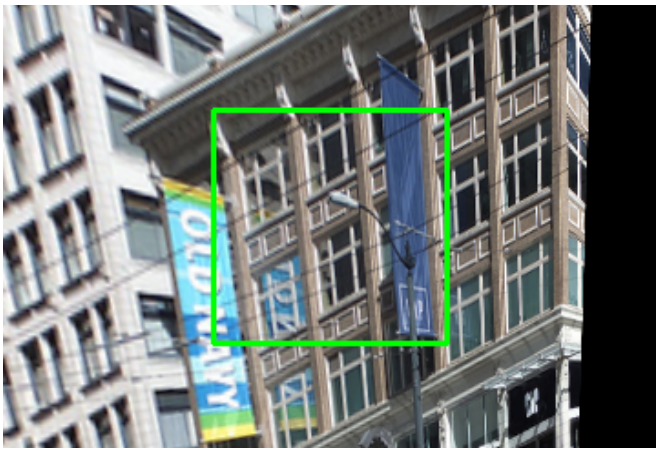


Fig. 14: Warp image such that new points fall on previously selected points (green box in Fig 13)

II. PHASE 2: DEEP LEARNING APPROACH

This section represents homography estimation between two images using deep learning. We have trained one model with two different training methods, supervised and unsupervised. Our model takes two channels input, each containing a gray-scale image, and gives the value of 4 point homography between those two images.

A. Data Generation

In order to train our deep network, we need real world data, but generating such a dataset is quite expensive and generated images will have lots of errors, because of possible position errors in camera placements. To take a practical approach, we have used MS COCO dataset, to artificially generate images with known 4 point homographies. We have created an artificial dataset for training and validation, using following steps.

(a) Take a random patch from an image, and with random perturbations at every corners of that patch, get four new points as shown in Fig 13. We have chosen image size to be 128×128, and $\rho = 32$ as maximum allowable perturbation. Images were randomly flipped vertically and random changes in brightness were made for better data augmentation.

$$H_{4point} = \begin{bmatrix} \Delta u_1 \\ \Delta v_1 \\ \Delta u_2 \\ \Delta v_2 \\ \Delta u_3 \\ \Delta v_3 \\ \Delta u_4 \\ \Delta v_4 \end{bmatrix} \quad (1)$$

4 point homography can be defined by Equation 1. Δu_i shows the perturbation of point i in x direction and Δv_i shows the perturbation of point i in y direction.

(b) Using these points, get the homography between the two images. We have used *cv2.getPerspectiveTransform* to calculate the homography.

(c) Warp the image for this homography and get a new patch as shown in Fig 14, with the same points selected in (a).

Fig. 15: Random patches taken from MS COCO dataset images

(d) It is necessary for these patches to have some features, otherwise model will not converge well. We saved these

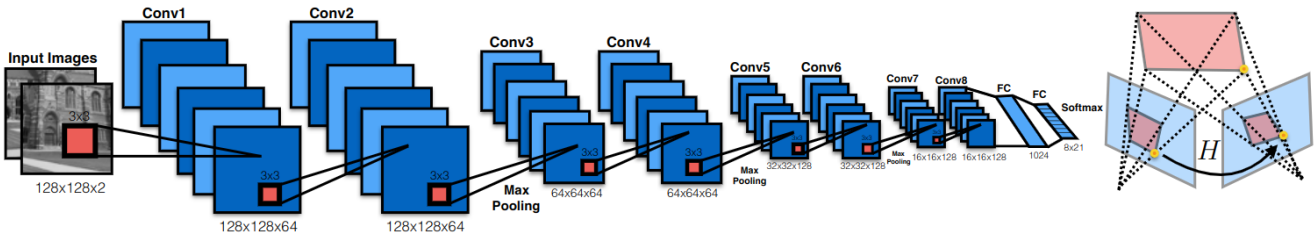


Fig. 16: VGG like architecture with 8 layers [2]

patches after making sure it had some good features to track (by the use of *cv2.goodFeaturesToTrack* function). A total of 46052 pairs of training and 4668 pairs of validation images were generated and saved.

B. Model Generation

We tried taking state of the art VGG and RESNet with less layers, as our homography net, but primary results of RESNet were not good and model was not converging, which left no choice but to go with standard VGG like model shown in Fig 16. Model has 8 convolution layers with 3 maxpooling layers. All the convolution layers has kernel size of 3 and number of filters are 64, 64, 64, 64, 128, 128, 128 and 128 respectively. Last convolution layer is flattened out and is connected to next fully connected layer with 1024 neurons. A dropout layer with 50 % probability. Final layer has 8 neurons and they directly give the 4 point homography values.

C. Supervised Approach

In this approach, we trained our model with the data, generated in data generation section. We started by normalizing our images in [-1,1]. Data augmentation was already taken care of while generating the dataset. We trained our model with mini-batches of 16 and with variable learning rate. We started with the learning rate of 0.0001 and after every 8 epochs, we reduced it by factor of 10. Our model has been trained for 20 epochs and plots of errors and loss can be seen in Fig 19. L2 euclidean loss was used to train the model.

$$L2 - Loss = \frac{1}{2}||H_4^{original} - H_4^{predicted}||^2 \tag{2}$$

Accuracy can be used to measure the performance of model in classification problem, but for the regression problem like this, we defined the error as mean of absolute difference between original and predicted 4 point homography.

$$Error = \frac{1}{8}|H_4^{original} - H_4^{predicted}| \tag{3}$$

D. Unsupervised Approach

In unsupervised approach, we train our model to give 4 point homography, without actually showing it the ground truth values. We have used the same homography net architecture as used in supervised approach, but with that, we need to add DLT (direct linear transformation) and spatial transformation layer. DLT layer shall calculate the homography matrix using 4 point homography predicted by homography network. Spatial transformation layer takes that homography matrix and use it to warp our second input image. Warped image should give us the first input image and this information can be used to train our model. We get the L1 loss between warped image and input image and using that loss we trained out homography net.

It is necessary that gradients should pass through DLT and spatial transformation layer in order to change the weights of homography net. Both this layers should be differentiable for that to happen and from paper [3], we get the idea of how to do that. Code for general spatial transformation layer was provided and it was used as it is. Main challenge was to write DLT in tensor form. We need to solve $Ax = B$, where all the columns of A and B matrix were generated with the Δu and Δv values.

$$L1 - loss = |H_4^{original} - H_4^{predicted}| \tag{4}$$

Output of spatial transformation layer is image and it can view in Fig 18. L1 loss shown in above equation was used to compute the loss. We trained our model for 20 epochs starting with 0.0001 learning rate. After every 8 epochs, we reduced the learning rate by the factor of 10. Plots for loss and error can be seen in Fig 20.

III. Results

Both the models gave average error of 2 pixels in 4 point homography calculations. We tried running these models on test set provided and results are discussed here.

From Fig 21, we can see that model has predicted the 4 point homography, with very less errors. One of the possible reason for this could be the presence of good number of features. After that it was necessary to what kind of prediction model would make in the absence of features.

Images shown in Fig 22, has real dearth of features. Patches taken from these images have monotonous color and almost no corners. Our model prediction are not satisfactory in such situations. Homography points are shifted by big distances and it shows real bad homography prediction. We

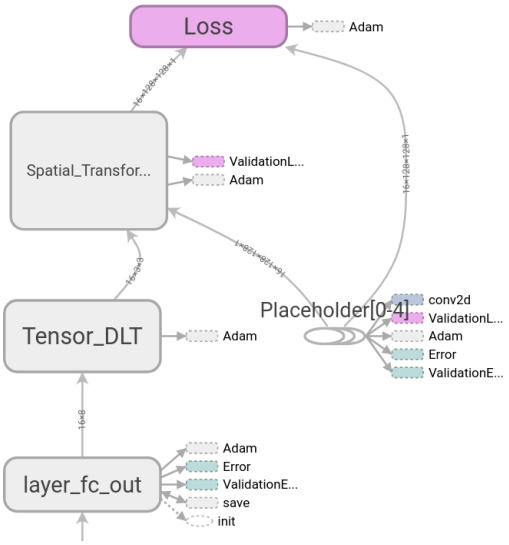


Fig. 17: Unsupervised model

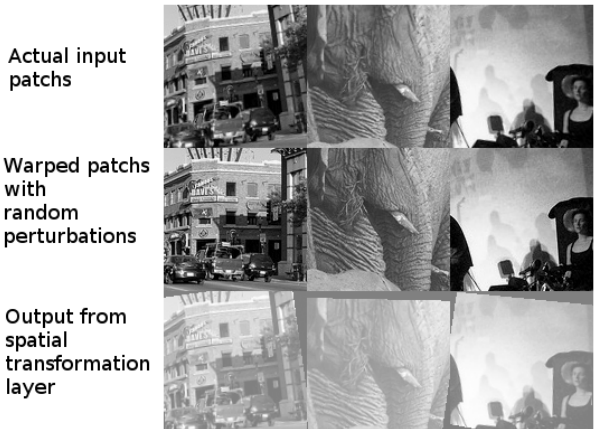
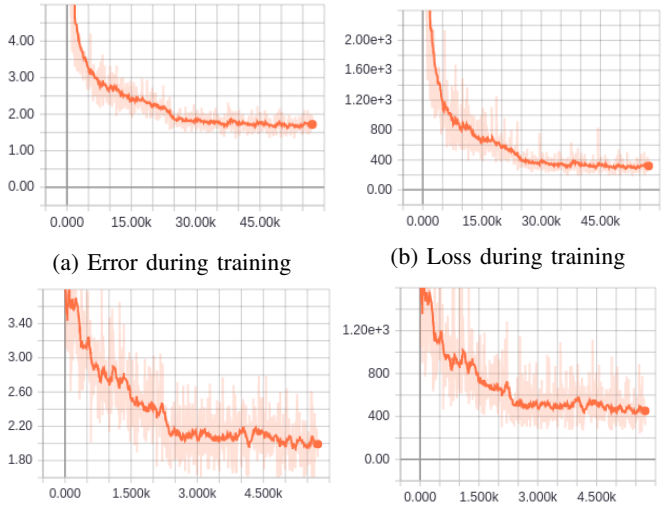


Fig. 18: Output of spatial transformation layer



(a) Error during training (b) Loss during training

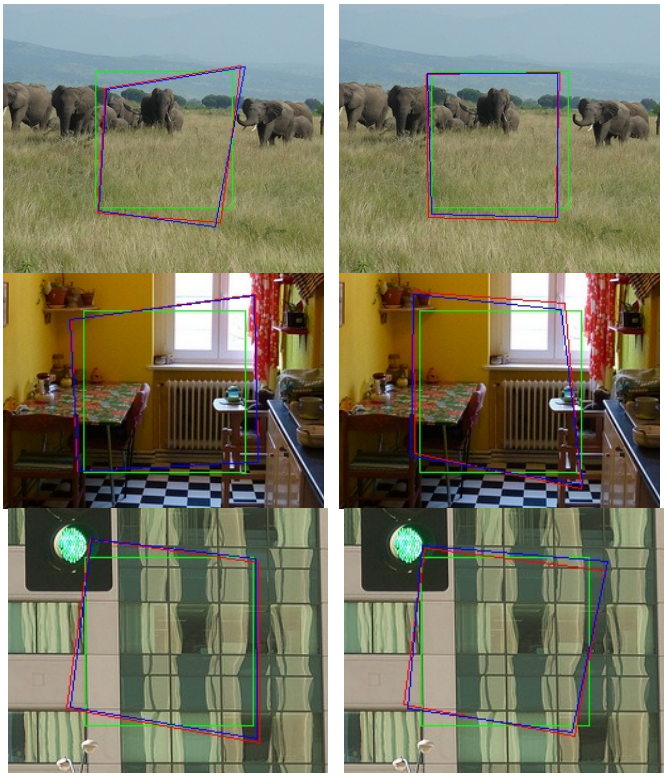
(c) Error during validation (d) Loss during validation

Fig. 19: Training the Homography net with supervised approach

(a) Error during training (b) Loss during training

(c) Error during validation (d) Loss during validation

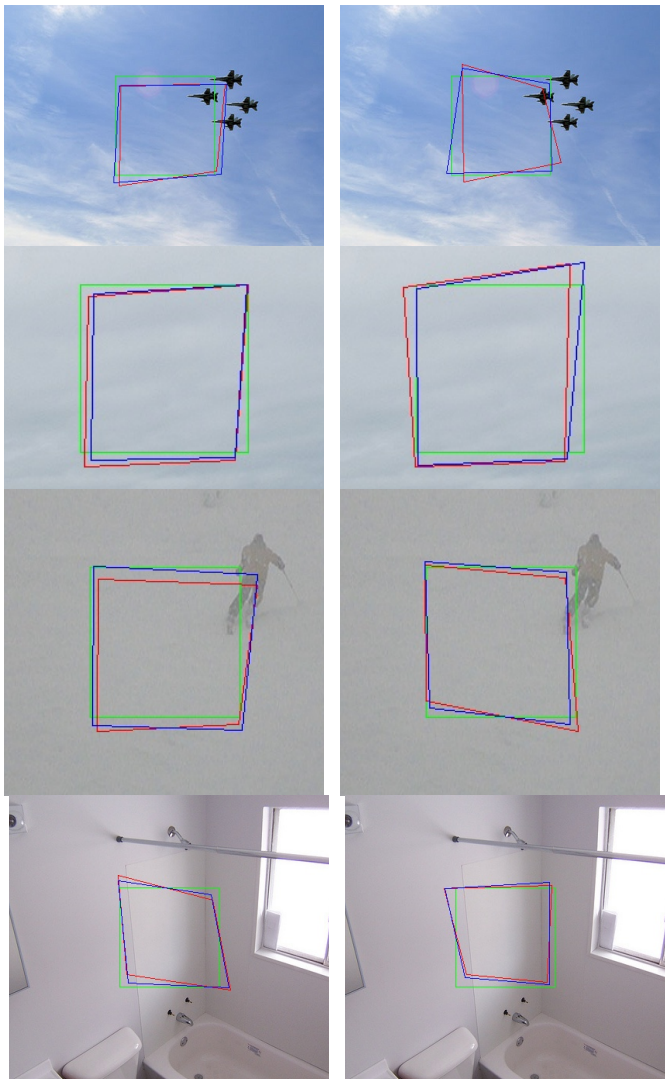
Fig. 20: Training the Homography net with upsupervised approach



(a) (b)

Fig. 21: Homography net has detected impressive homography, probably because of availability of features in provided patch. Green box indicates the patch randomly selected from an image. Red box indicates new patch generated with random perturbations to create a warped image. Blue box indicates the predicted perturbations with trained homography model. (a) Predicted with supervised homography net. (b) Predicted with unsupervised homography net.

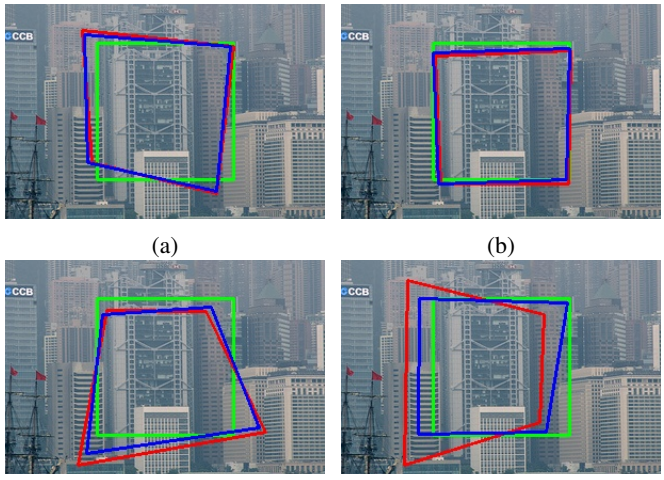
also wanted to see the predictions with the images having different brightness and contrast, but because of less time availability, that analysis has not been presented here.



(a) (b)

Fig. 22: Homography net unable to detect correct homography because of scarcity of features in patch. Green box indicates the patch randomly selected from an image. Red box indicates new patch generated with random perturbations to create a warped image. Blue box indicates the predicted perturbations with trained homography model. (a) Predicted with supervised homography net. (b) Predicted with unsupervised homography net

Network has been trained with the data generated with the random perturbations of $\rho = 32$ pixels. We tried to compare its predictions on image patches, generated with the random perturbations of 64. Results can be seen in Fig 23. Network was predicting less erroneous 4 point homography with for the patches with perturbations less than 32, but for the perturbations of 64, predictions were totally random.



(a) (b)

(c) (d)

Fig. 23: Predictions made with the model, trained on images which had homographies because of random perturbations of 32 pixels only. (a) Random perturbation of 32 with supervised homography net. (b) Random perturbation of 32 with unsupervised homography net. (c) Random perturbation of 64 with supervised homography net. (d) Random perturbation of 64 with unsupervised homography net

REFERENCES

[1] http://cocodataset.org/home

[2] Daniel DeTone, Tomasz Malisiewicz, Andrew Rabinovich "Deep Image Homography Estimation"

[3] Ty Nguyen, Steven W. Chen, Shreyas S. Shivakumar, Camillo J. Taylor, Vijay Kumar, "Unsupervised Deep Homography: A Fast and Robust Homography Estimation Model",12 Sep 2017 (v1), last revised 21 Feb 2018 (this version, v3)