CMSC733: Project 1 - MyAutoPano

Gaurav Raut
University of Maryland
Email: gauraut@umd.edu

Sonaal Kant
University of Maryland
Email: sonaal@umd.edu

I. Phase 1: Traditional Approach

The traditional approach follows a sequence of operations given a set of input images to get the panoramic image. These operations include loading the set of input images, detecting corners, applying adaptive non-maximal suppression (ANMS), extracting the feature vectors, matching feature vectors, applying RANSAC to remove outliers, estimating homographies between images using the matched features after RANSAC and finally, warping and blending the final panoramic image. We will start serially with the explanation and outputs of these steps and finally discuss the pros and cons of our approach in the conclusion.

A. Corner Detection

In order to detect corners in the input image, we made use of the OpenCV's cv2.cornerHarris() function. This functions takes the grayscale equivalent of the input image and outputs a corner score matrix. The higher the value in the corner score matrix, the more the probability of the pixel being a corner.

B. Adaptive non-maximal suppression

Now that we have the corner score matrix, we need to find the pixel co-ordinates of the corners. To do so, we take this matrix as an input to our ANMS algorithm. Since the corner score matrix is a probability matrix, we get a lot of redundant corners. The purpose of ANMS is to filter out this redundancy and form spaced corner points. Firstly, we isolate the local maximas and get their co-ordinates. Then we sort the Eucledian distances and choose the N_{best} = 500 points which are above a certain threshold distance away from each other. The pseudo code of the ANMS algorithm is provided Fig. 2.

C. Feature Extraction

The process of feature extraction compares the feature vectors from two images and matches the features between two images. This is implemented using two main steps,
i. Describe feature vectors in individual images
ii. Compare two adjacent images and perform feature matching

1) Feature Descriptor: In this step, we take our N_{best} corner points from ANMS's output and associate a feature vector to each of these points. To do so, we take each point and extract a 41x41 patch around it with the point as the centre. We later apply Gaussian blur on this patch and resize the output into an 8x8 dimension. Now, we flatten this resized patch and standardize the vector to remove bias and achieve illumination invariance.

Fig. 1: Corner Detection outputs. Here, red colored spots are the corner probabilities

$$StandardizedFV = \frac{fv - mean(fv)}{std(fv)} \quad (1)$$

where,
fv is the feature vector
The output we get is a 64x1 vector for each of the N_{best} points which is nothing but the feature vector associated with

Input : Corner score Image (C_{img} obtained using `cornermetric`), N_{best} (Number of best corners needed)
Output: (x_i, y_i) for $i = 1 : N_{best}$

Find all local maxima using `imregionalmax` on C_{img};
Find (x, y) co-ordinates of all local maxima;
$((x, y)$ for a local maxima are inverted row and column indices i.e., If we have local maxima at $[i, j]$ then $x = j$ and $y = i$ for that local maxima);
Initialize $r_i = \infty$ for $i = [1 : N_{strong}]$

```
for i = [1 : N_strong] do
    for j = [1 : N_strong] do
        if (C_img(y_j, x_j) > C_img(y_i, x_i)) then
            ED = (x_j − x_i)^2 + (y_j − y_i)^2
        end
        if ED < r_i then
            r_i = ED
        end
    end
end
```

Sort r_i in descending order and pick top N_{best} points

Fig. 2: ANMS

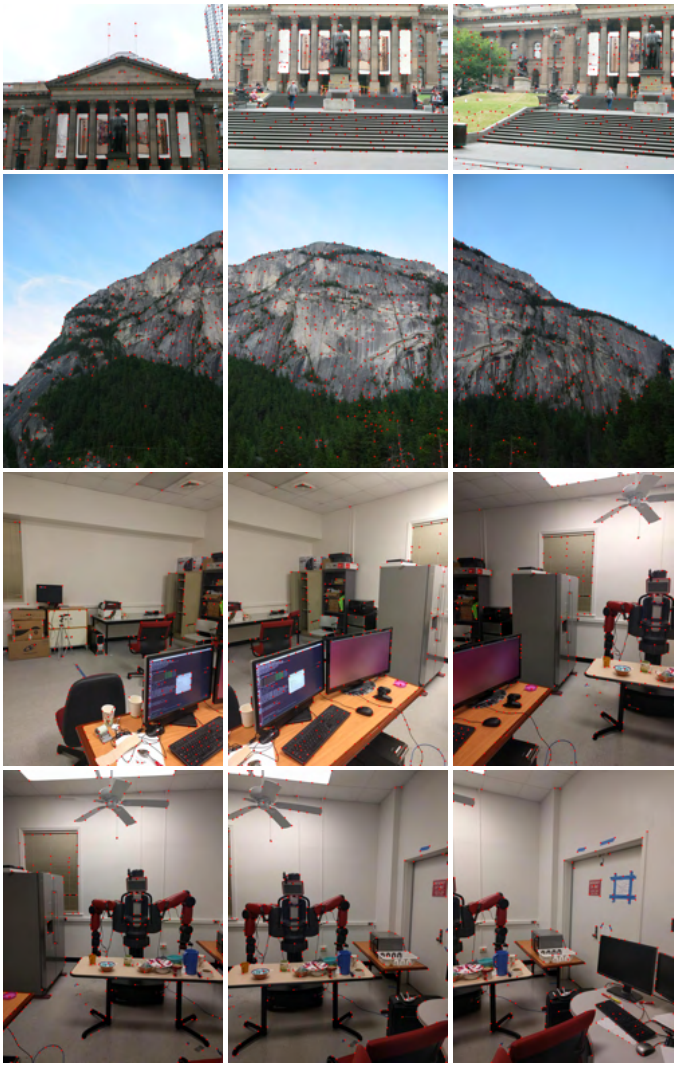

Fig. 3: Corner Detection outputs. Here, red colored spots are the corner probabilities

that particular point.
There are instances when the points are so close to the edge of the image that we are unable to form a 41x41 patch around the point. In such cases, we can simply skip the point and move on to the next point.

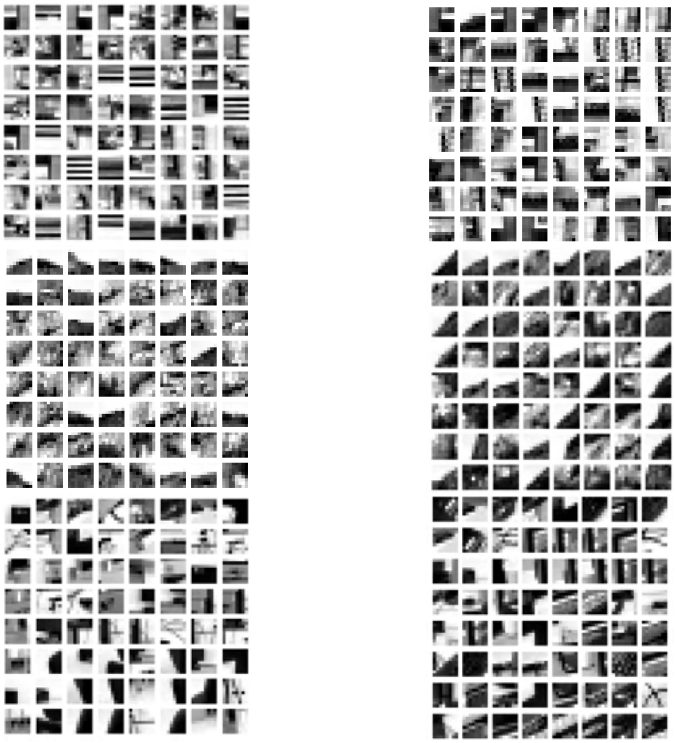

Fig. 4: Examples of random 128 extracted features from Sets 1, 2 and 3

2) Feature Matching: In the previous step, we associated a feature vector with each of our corner points skipping only those which were close to the edge. Now, we move forward with comparing and matching these vectors between two adjacent images. To do so, we loop over every point's feature vector in one of the image and compute the SSD (Sum of squared distances) with all the feature vectors in the other image. Getting the SSD is a computationally heavy task and hence, to improvise the runtime, we simply computed the the sum of the absolute differences. Now, we select the point for which the distance is lowest. Although, we just can't accept the lowest pair since this could be a false match. Hence, we take the ratio of the lowest distance and the second lowest distance and we accept the point pair as matched points if and only if the ratio is below a certain threshold. We have chosen this threshold as 0.85. After feature matching, we return the pairs of matched points. The outputs of matched features for various sets are provided in figures below.

D. RANSAC for outlier rejection and to estimate Robust Homography

When we look at the result of our feature matching, we see that there are some wrong matches along with many correct matches. These wrong matches are the outliers and are to be rejected. Else, we won't be able to get a good

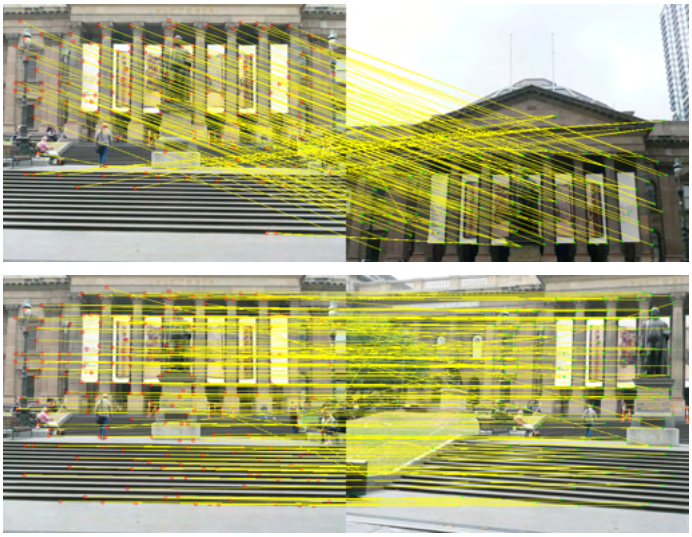

Fig. 5: Matched Features for Set 1

Homographic relation between the images. In order to reject these outliers, we apply the RANSAC (Random Sampling Consensus) algorithm. Our RANSAC algorithm takes input the two arrays of matched feature points and outputs the Homography matrix for transforming the second image with respect to the first image.
Now to ask the question, what does RANSAC exactly do? The algorithm runs for 1000 iterations and estimates the homography matrix by randomly selecting four matched point pairs. It then selects the homography matrix with the most inliers, thus rejecting the outliers. We use the function cv2.findHomography() to compute the homography matrix. Now, based on these inliers, the RANSAC algorithm calculates the best estimate of the homography and returns it. The matched features after RANSAC are also shown in figures below.

E. Blending Images

With our Homography estimates, we are finally ready for blending images and creating one panoramic image. But, if we recall, we have calculated the Homographies only for adjacent images. These homographic relations only apply to the adjacent image and not to the other images in the set. In order to solve this conundrum, we select an anchor image and project all homographies with respect to this anchor. The anchor image is chosen as the output index of the floor division of the set size by 2.
As said before, we need to calculate the homographies of all images with respect to the anchor image. To do so, we take the product of consecutive homographies till we reach the direct neighbors of the anchor image. More specifically, the relation between the anchor, let's call it the k^{th} element in the image set, and the n^{th} element can be given as,
if $n < k$,

$$H_{n,k} = \prod_{n=0}^{k-2} H_{n,n+1} \tag{2}$$

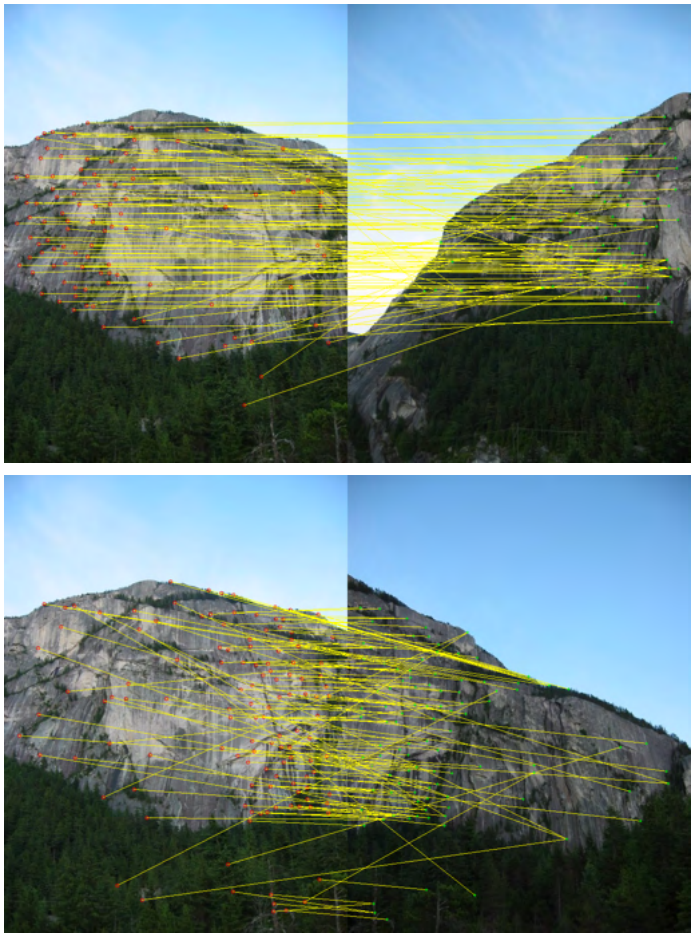

Fig. 6: Matched Features for Set 2

else,

$$H_{n,k} = \prod_{n=t}^{k-1} H_{n,n-1} \tag{3}$$

where, t = Total images - 1
After the homography matrices are ready, we simply apply the function cv2.warpPerspective() on each image with it's corresponding homographic matrix. To blend the warped images into a big panorama, the panoramic image's size is chosen to be equal to the $X_{max} - X_{min}$ and height as $Y_{max} - Y_{min}$.
Another hurdle that we face is that the anchor image starts from the origin of this panorama. This causes all the images to the top and left of the anchor to form negative origins and hence the images get cut from the final panorama. To improvise here, we simply translate all the images based on the minimum X and Y values. This method makes sure that all images fit properly within the bounding box. To do so, we manipulate the first two elements of the last column of every homography matrix. We add $-X_{min}$ and $-Y_{min}$ in the first two elements respectively and our images get translated.
Now, we just overlay all the images on the anchor image and we get our final panoramic output.

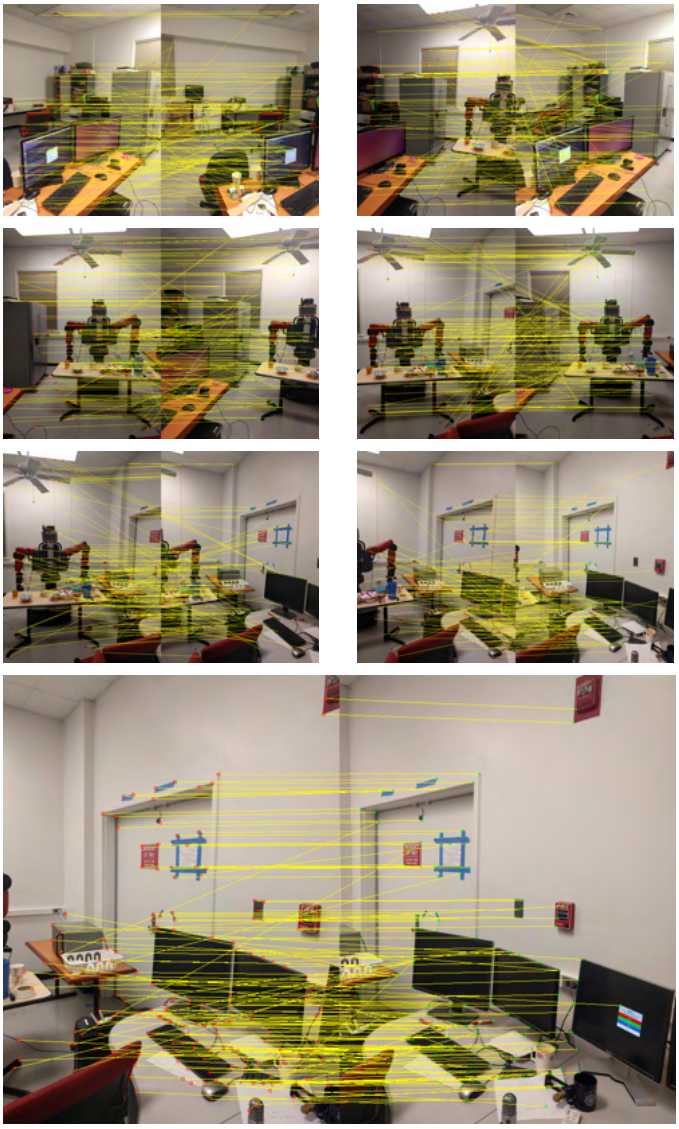

Fig. 7: Matched Features for Set 3

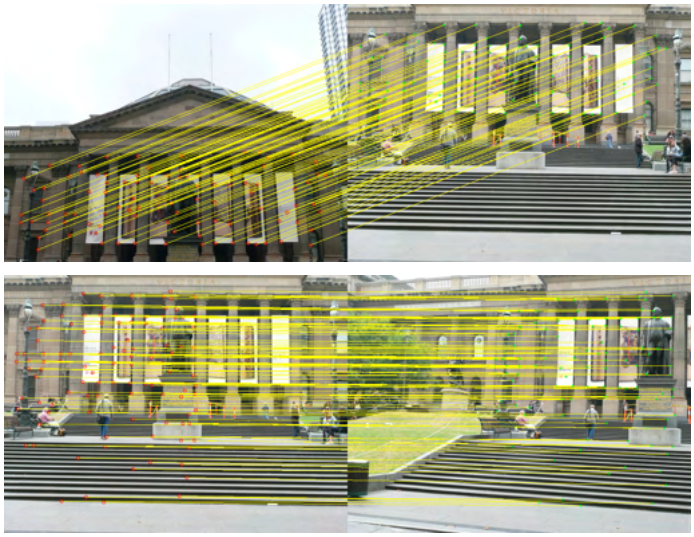

Fig. 8: Matched Features after RANSAC for Set 1

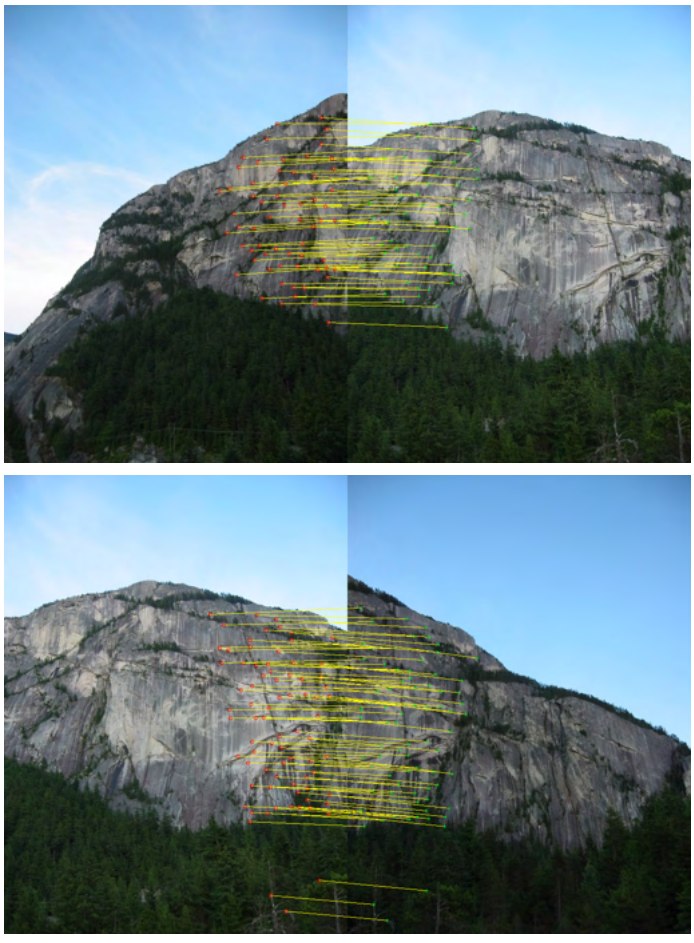

Fig. 9: Matched Features after RANSAC for Set 2

F. Conclusion

From what we see in the outputs, the algorithm is erroneous when applied on sets with more than 4-5 images. This is because of the Homographic matrices multiplication in order to convert every adjacent image's homography with respect to the anchor image. The errors between these matrices will add up over the range and hence, we won't be able to achieve a perfectly warped image. These errors increase as we get farther from the anchor image. Although, if we have accurate estimates of the homography, this method won't cause a huge distortion.

Another approach to overcome this issue will be to run warping in a recursion algorithm. Meaning, we can take the perspective warp between two adjacent image, then concatenate it and pass this image to feature matching with the next image in the list. The drawback of this approach is that it's computationally heavy and hence, a slower runtime. Although, this can surely be optimized but, we didn't have enough time to implement this.

To look at the silver lining, our algorithm runs faster in comparison to the recursion approach and also achieves accurate

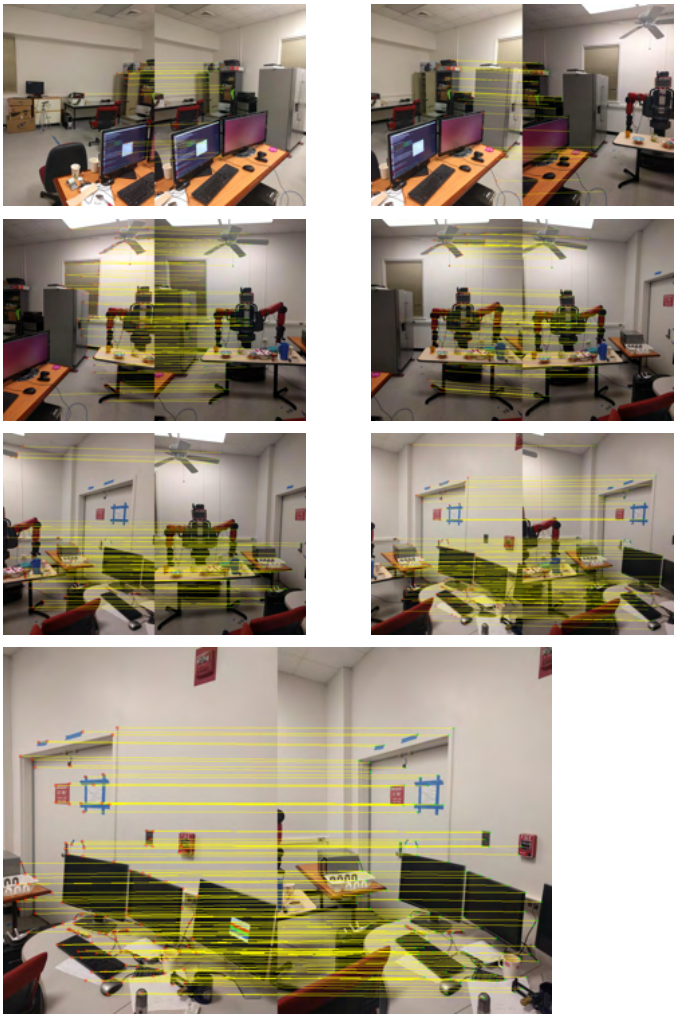

Fig. 10: Matched Features after RANSAC for Set 3

results for sets with less than 5 images.

Fig. 11: Panorama of Set 1 (3 images)

Fig. 12: Panorama of Set 2 (3 images)

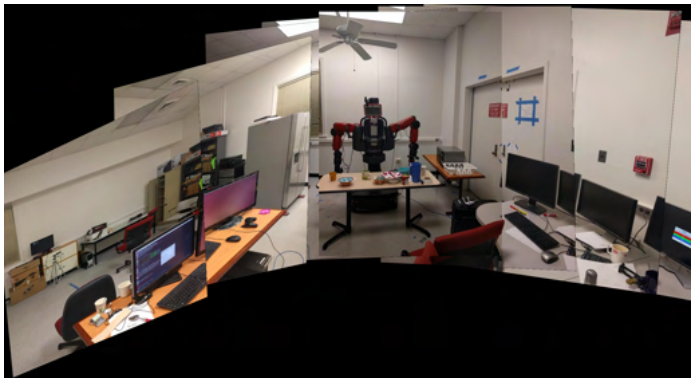

Fig. 13: Panorama of Set 3 (8 images)

Fig. 14: Panorama of Custom Set 1 (4 images)

Fig. 15: Panorama of Custom Set 2 (3 images)

Fig. 16: Corner Detection and ANMS outputs for Test Set 1

Fig. 17: Corner Detection and ANMS outputs for Test Set 2

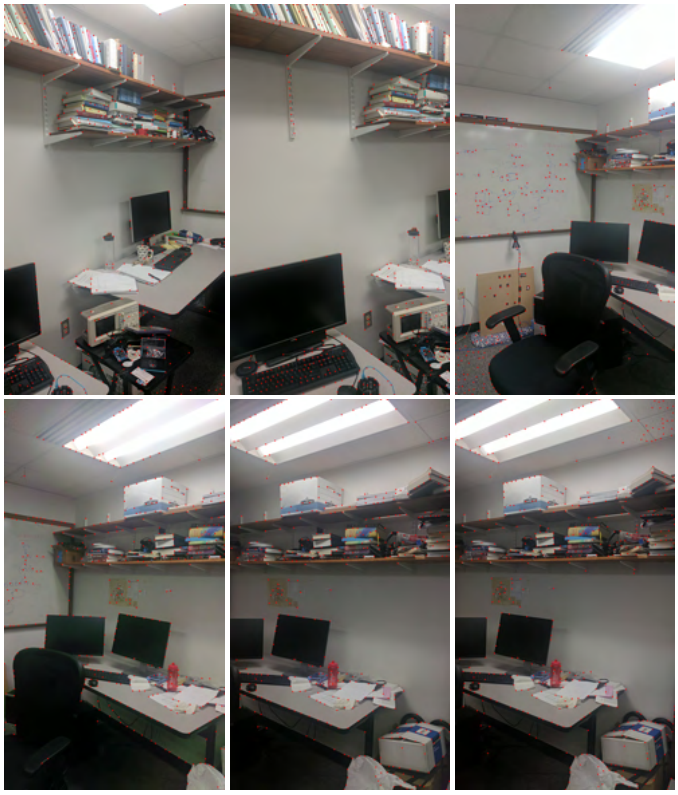

Fig. 18: ANMS outputs for Test Set 2

Fig. 19: Corner Detection and ANMS outputs for Test Set 3

Fig. 20: Corner Detection and ANMS outputs for Test Set 4

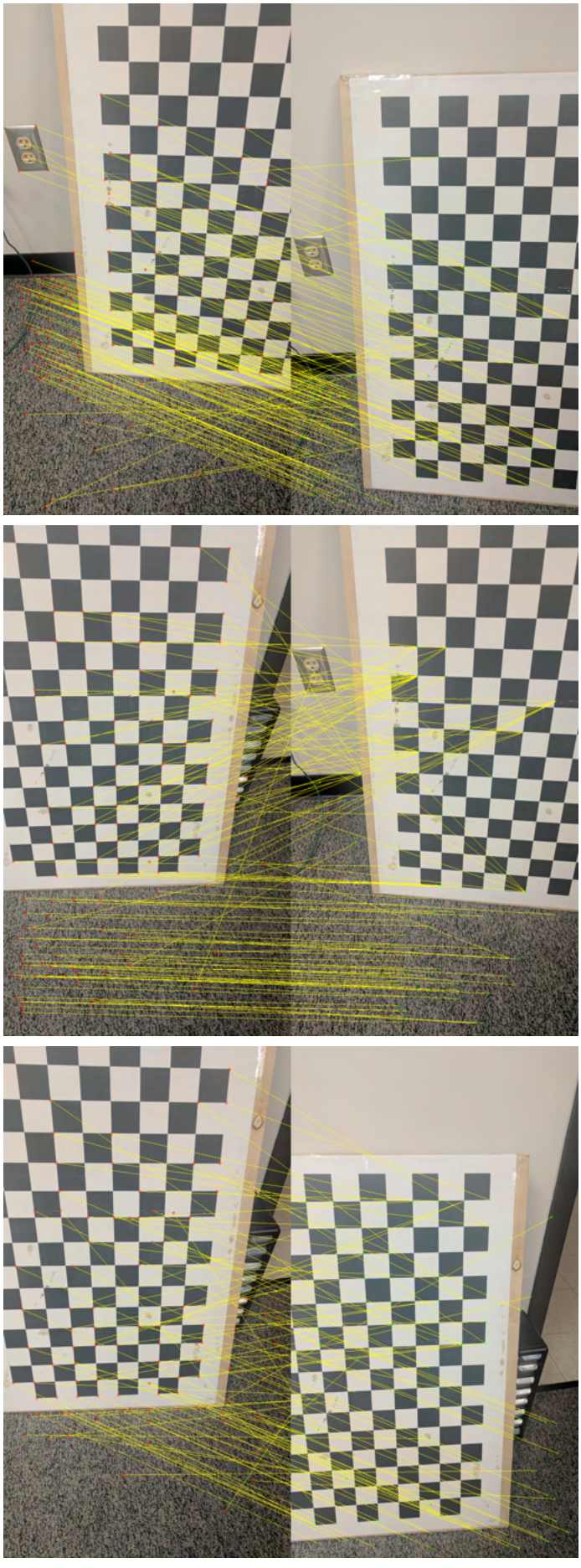

Fig. 21: Matched Features for Test Set 1

Fig. 22: Matched Features for Test Set 2

Fig. 23: Matched Features for Test Set 3

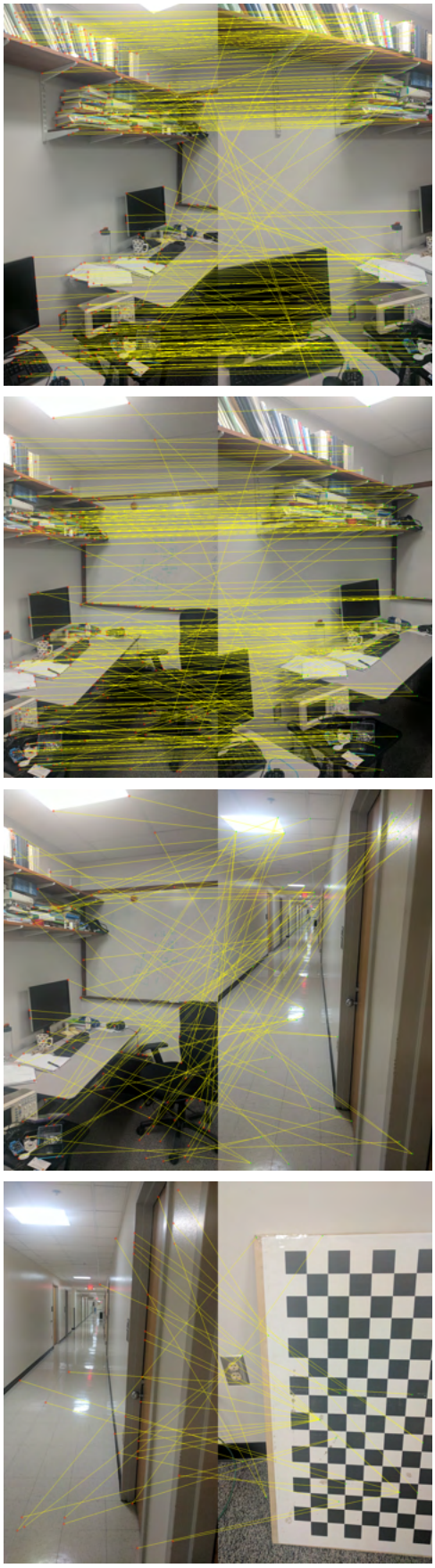

Fig. 24: Matched Features for Test Set 4

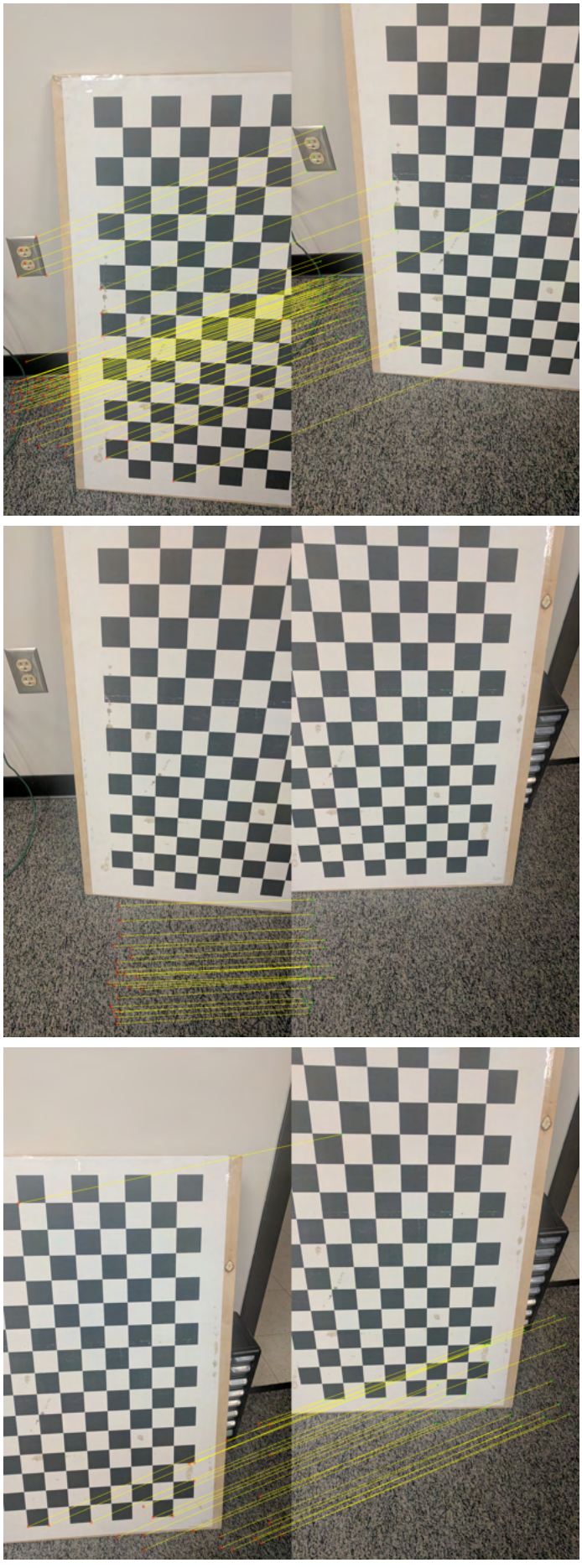

Fig. 25: Matched Features after RANSAC for Test Set 1

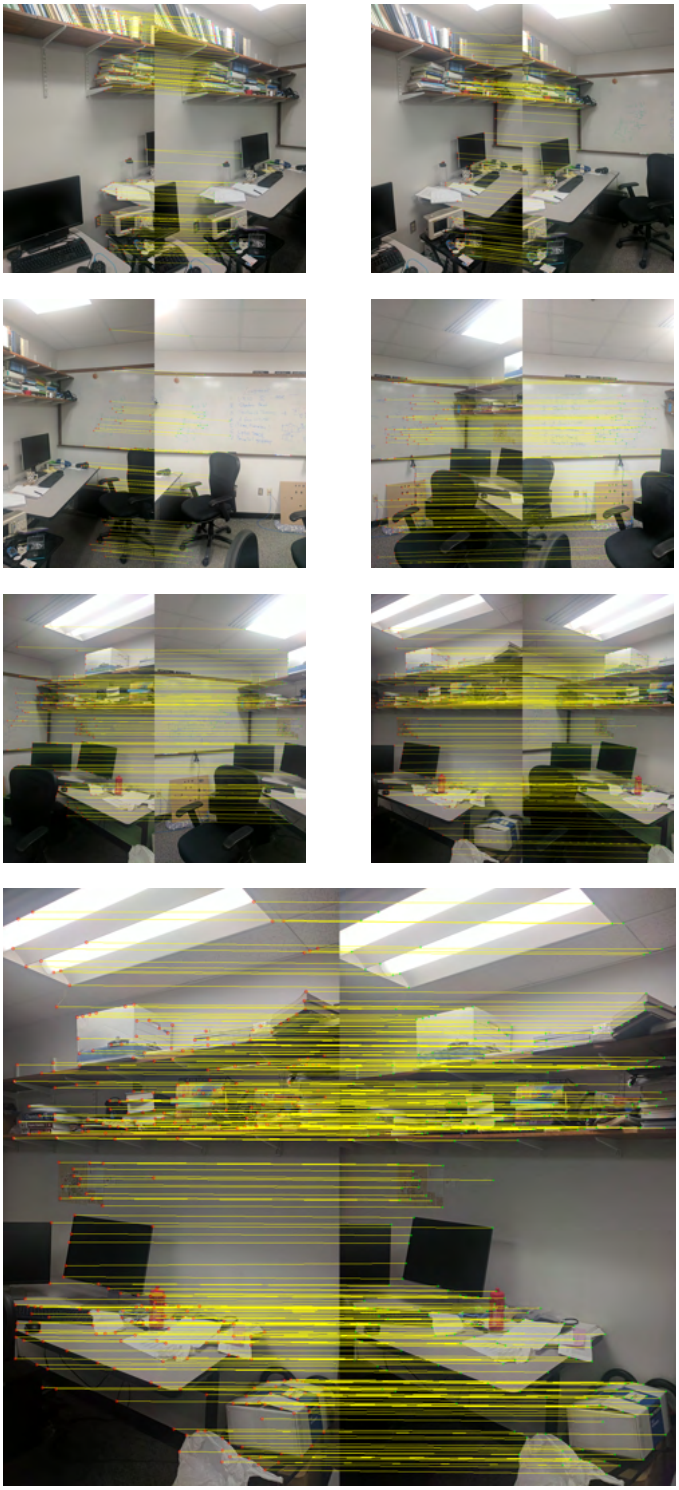

Fig. 26: Matched Features after RANSAC for Test Set 2

Fig. 27: Matched Features after RANSAC for Test Set 3

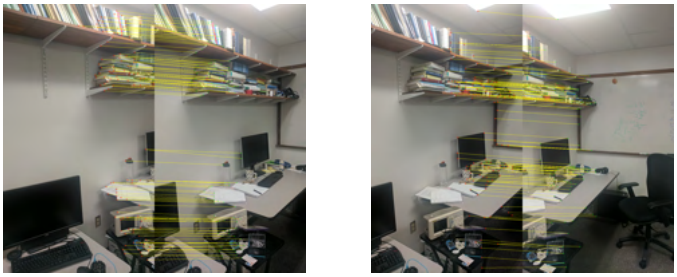

Fig. 28: Matched Features after RANSAC for Test Set 4

Fig. 29: Panorama of Set 1 (4 images)

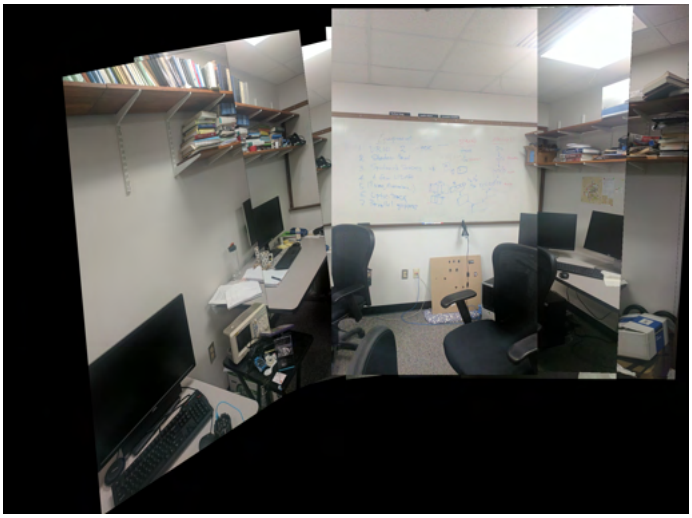

Fig. 30: Panorama of Set 2 (9 images)

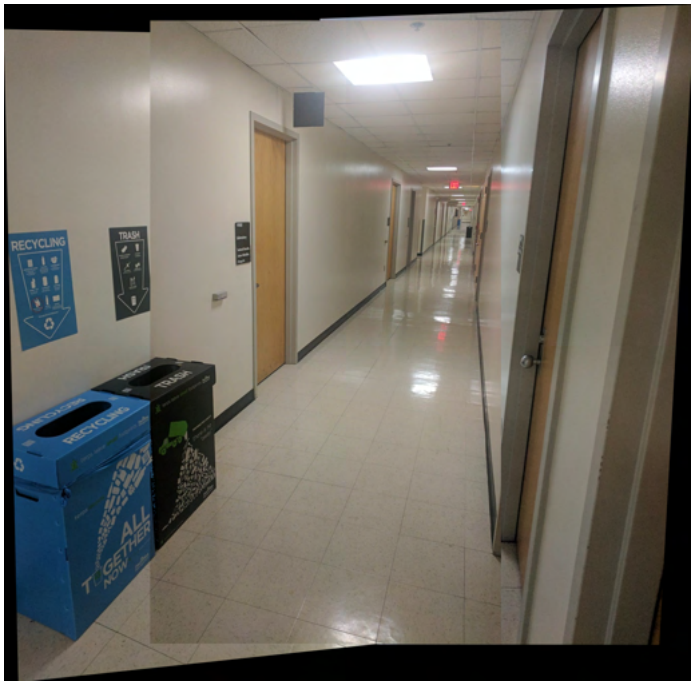

Fig. 31: Panorama of Set 3 (3 images)

Fig. 32: Panorama of Custom Set 4 (4 images)

II. Phase 2: Deep Learning Approach

Now, we will explore our implementation of the deep learning approach. In the following sections we will discuss the data generation technique for supervised and unsupervised

approach, followed by the details on network architecture of supervised and unsupervised methods.

A. Data Generation

The dataset with training and validation images are first resized to 320 x 240 and then converted to grayscale. We then follow the below steps for every image in the training and validation set for creating the dataset.

- Randomly get 4 coordinates of a patch of size (128 x 128) keeping the boundary conditions in mind.
- Perturb those 4 coordinates using some random error in range (-32, 32). This perturbation is used as a ground truth to train our supervised model.
- Generate the homography matrix H_{AB} using cv2.getPerspectiveTransform function.
- Use the inverse of that matrix to warp the training image along with the coordinates.
- Crop the rectangular patch from the warped image of size (128 x 128)
- Concatenate the training Image and the warped image along channel and use this as the input to your model.

We create 500000 such training images by randomly sampling from our train set similarly we create 5000 validation images.

B. Network Architecture

We use the basic VGG like architecture for both our supervised and unsupervised model. Input of both the networks is the concatenation of patches from train and warped image (patchA, patchB). The output from the network is the perturbation in the points. We normalize our perturbation in -1,1 while training our network.

1) Supervised Model: In this, we use SGD optimizer with learning rate 1e-3 and a batch size of 512 to train the network. We use higher number of training images so as to prevent overfitting. We also show that training on smaller dataset (5000 images) overfits easily. We present a plot of training and validation loss in fig 33. Also we test the trained model on test images and the prediction on unseen images can be seen in fig 34-40. We train our model using the MSE loss and due to time constraints our model is still not saturated and can reach a better local minima.

2) Unsupervised Model: In this, we use Adam optimizer with learning rate 1e-4 and a batch size of 128. The hyperparameters for this method are slightly different because this method is slightly difficult to train and not stable. We train our network with photometric loss which is similar to L1 loss, we also show the plot of our training loss and validation loss for comparison. The unsupervised model was trained for very few epochs as compared to the supervised model.

C. Performance on Val/Test set

We compare both our models on the test set and calculate the mean corner pixel error of our model on validation and test set. As we can see results in Table 1., Our supervised model works much better than the unsupervised model and we suspect couple of reasons behind that observation.

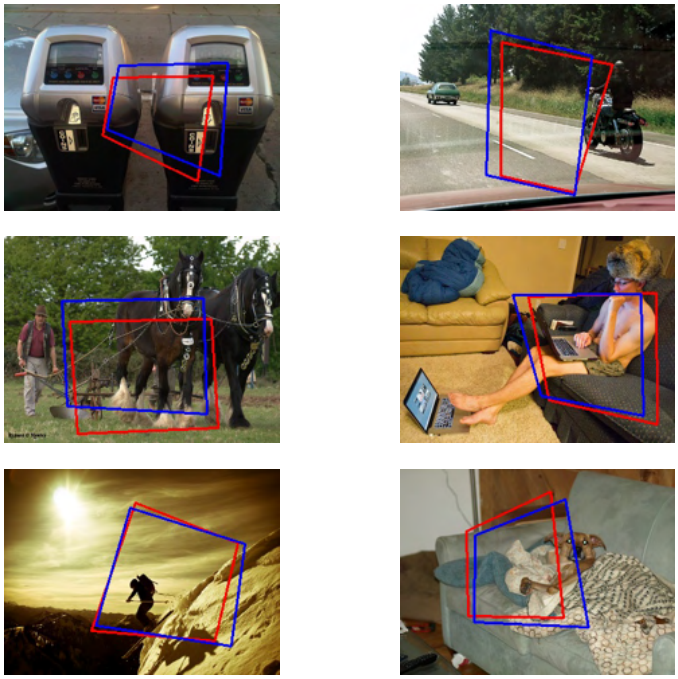

Fig. 33: Training outputs

- The unsupervised model is trained for very few epochs as compared to supervised model due to time constraints.
- The A matrix generated in TensorDLT layer for calculating the homography matrix results in same singular values. This becomes an issue because the gradient of the SVD is 1 over difference square of singular values this results in nan or very high gradients that results in unstable training.

Model Name	VAL EPE	TEST EPE
SUPERVISED	10.11px	10.46px
UNSUPERVISED	16.93px	16.94px

D. Conclusion

We create a homography estimation model using classic machine learning methods and the deep learning methods but deep learning methods are more dependent on high quality dataset generation process. While the performance of the deep learning model is high on synthetic dataset it seems very limited on real world images. Hence, the homography estimation and stitching were not as good as the classic machine learning methods.

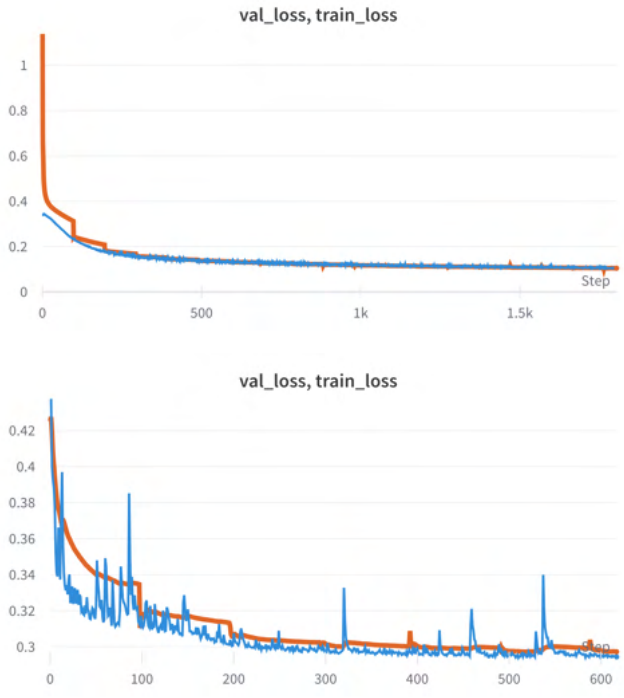

Fig. 34: Red: Training loss, Blue: Validation loss

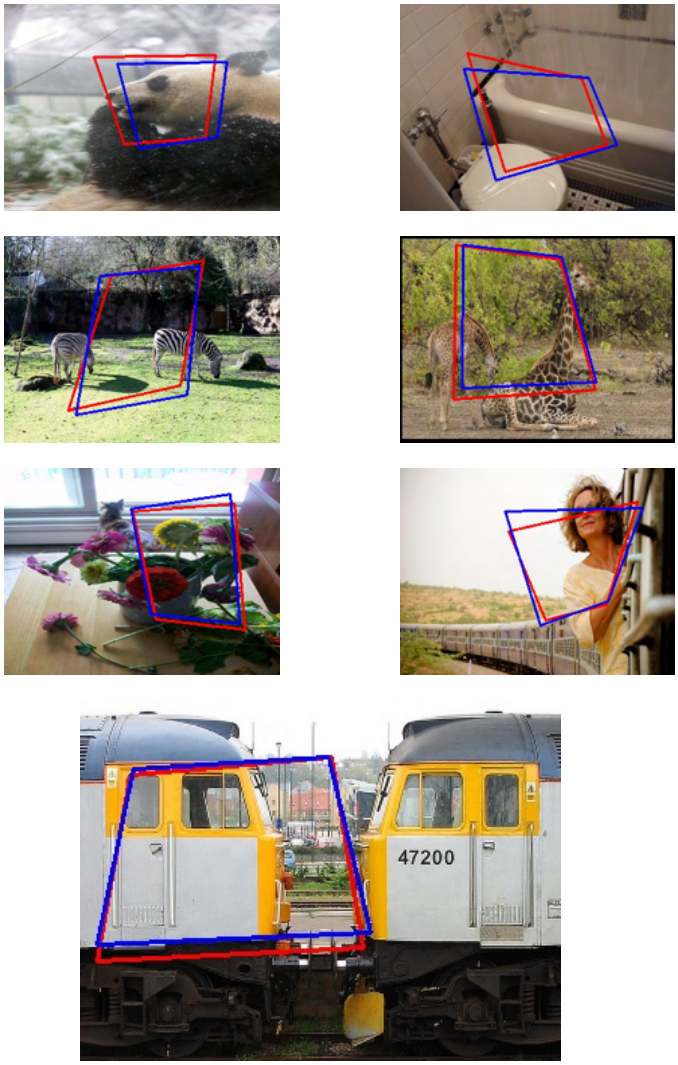

Fig. 35: Validation outputs