Computer Vision
Project: Relative Pose Estimation

Hodaya Koslowski and Ganit Kupershmidt

August 31, 2020

Contents

1 Introduction

The task of relative pose estimation is key in many computer vision tasks. It is usually solved using the 8 point algorithm that was introduced in [1]. This method uses the calibrated points to find the essential matrix, and then decompose it to one of two options for both the relative rotation and translation - four in total. In order to choose the correct combination out of the four, the points are triangulated. The chosen combination is the one that has the maximal number of points that in front of both views.
In recent years, the power of deep learning was demonstrated beautifully, overcoming difficult tasks that were a distant fantasy only ten years ago. Therefore, it was no surprise that research groups were attempting to solve the relative pose estimation this way.
In this exercise, we attempt solving the relative pose estimation using a neural network. Specifically, we build a Siamese neural network, inspired by the work done by [3]. The network consists of two branches of a pre-trained model. This model was trained on the places datasets by [2]. We tried different loss functions and architectures, and the results are reported next.

2 Evaluation

When evaluating the different models we used a fixed division to training and validation sets (Other than experiments that were done on specific smaller datasets). The two main evaluations we used on trained models (in addition to the loss function values over epochs) were epipolar lines that were drawn on the images, and angular distance.
The **epipolar lines** measure is mainly a sanity check, and it is evaluated manually - it allows us to grasp the power of the model visually. For each trained model, we took an example from the validation set and predicted the relative pose between the images. The relative pose lets us compute the predicted fundamental matrix F:

$$F_{ij} = K_i^{-T}[t_{ij}]_\times R_{ij} K_j^{-1} \tag{1}$$

Where t_{ij} and R_{ij} are the relative translation and rotation matrix, and K_i, K_j are the calibration matrices of the two views. When knowing the relative pose between two views, a point in one image corresponds to a line in the second image - due to ambiguity in the distance of the point to the two views. The fundamental matrix is a transformation that takes as input a point in one image and outputs the line on the other.

$$\begin{aligned} l_j &= F_{ij} x_i \\ l_i &= F_{ij}^T x_j \end{aligned} \tag{2}$$

If we have the ground truth matches between the points, we can easily see how well the model predicted the relative pose by comparing the lines that are produced by the ground truth F with the lines from the estimated F.

The second evaluation metric is the **angular distance** (in degrees) that was given to us as part of the exercise:

$$
\begin{aligned}
E_R &= \|(\text{vector}(R_{gt}^T R_e))\| \cdot \frac{180}{\pi} \\
E_t &= \cos^{-1}\left(\frac{t_{gt} \cdot t_e}{\|t_{gt}\|\|t_e\|}\right) \cdot \frac{180}{\pi}
\end{aligned}
\tag{3}
$$

3 Trivial Solution - 8 point algorithm

Before experimenting with different architectures, we evaluated the trivial solution given to us in Matlab. We used the **same validation examples** as in our python deep networks experiments. In figure 1 we can see that there are major translation errors for some examples. We were curious

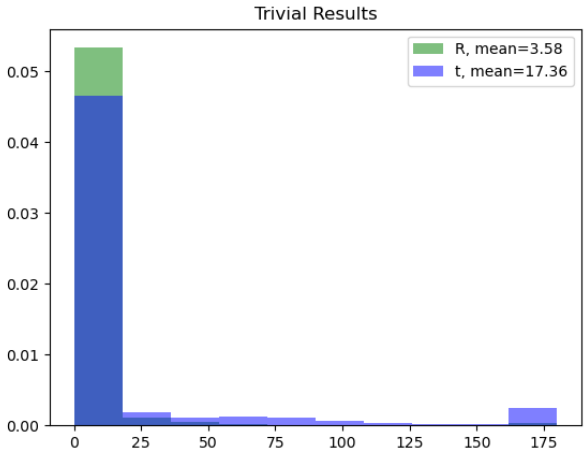


Figure 1: Results of the 8-point algorithm on the 5190 examples of the validation set.

to see what these examples look like, and eventually compare the performance of our best model on them.
We sampled one such "bad" data instance and got 'lund cath large/7619' (Figure 2). This input had a rotation error of 14.5707 and translation error of 78.3345. We will come back to it later.

4 Models

4.1 Baseline model

4.1.1 Model Architecture

As a baseline model we used Siamese network architecture. The model is built from two similar branches, each branch is a pre-trained model of Resnet50 (without the last fully connected layer), followed by spatial pyramid pooling (SPP) layer. The two branches were connected together by the

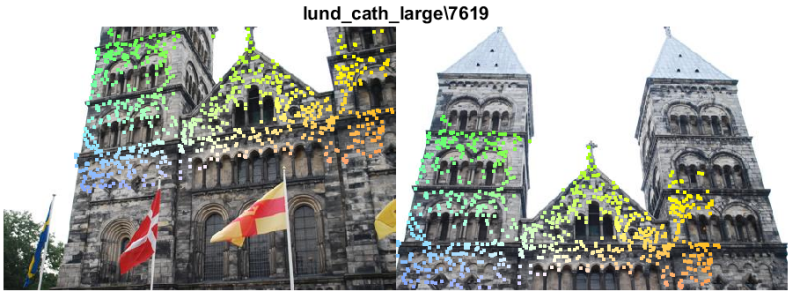


Figure 2: An example of an input that was hard for the 8-point algorithm.

head structure, which contains two consecutive fully connected (FC) layers. The first FC layer has 128 out features, and the second one has 7. The architecture is presented in figure 3.

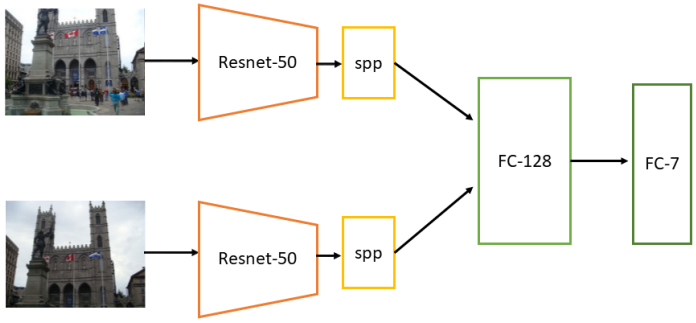


Figure 3: baseline architecture

4.1.2 Loss Function

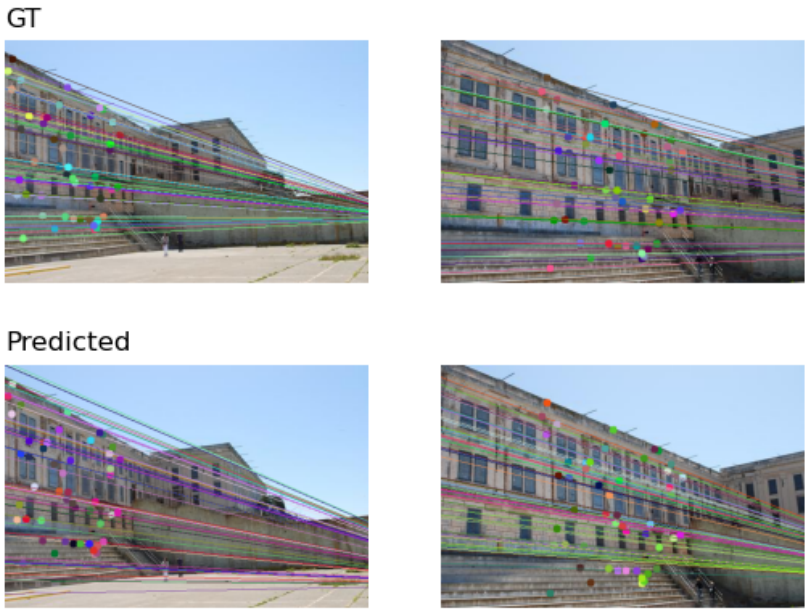
In the baseline model we used only the pose loss, as presented in equation (4).

$$L_{pose} = \frac{1}{m}\sum_{i=1}^{m} \beta \cdot \|q_{gt}^{(i)} - q_e^{(i)}\| + (1-\beta) \cdot \|t_{gt}^{(i)} - t_e^{(i)}\| \tag{4}$$

When $q_{gt}^{(i)}$, $t_{gt}^{(i)}$, $q_e^{(i)}$ and $t_e^{(i)}$ are the ground truth quaternion and translation of the i'th example, and the estimated quaternion and translation of the i'th example respectively. m is the batch size, and β is an hyper-parameter. For the baseline model we chose $\beta = 0.5$.

4.1.3 Results

In Figure 4 we can see an example for estimated epipolar lines. The training and validation pose difference loss of this model stands on around 0.5 along all 10 epochs of training. Moreover, the mean angular distance on the validation set for the rotation matrix is 21.21, and for the translation is 58.08.



(a) epipolar lines

Figure 4: Baseline models' results

4.2 Epipolar loss

4.2.1 Model architecture

In this section we used the same architecture as in section 4.1.

4.2.2 Loss function

In this section we introduce another loss, the epipolar loss, presented in equation (6).

$$L^{(i)}_{epipolar} = \frac{1}{n}\sum_{j=1}^{n} |(x^{(i)}_{j,2})^T \cdot F \cdot x^{(i)}_{j,1}| \tag{5}$$

$$L_{epipolar} = \frac{1}{m} \sum_{i=1}^{m} L_{epipolar}^{(i)} \tag{6}$$

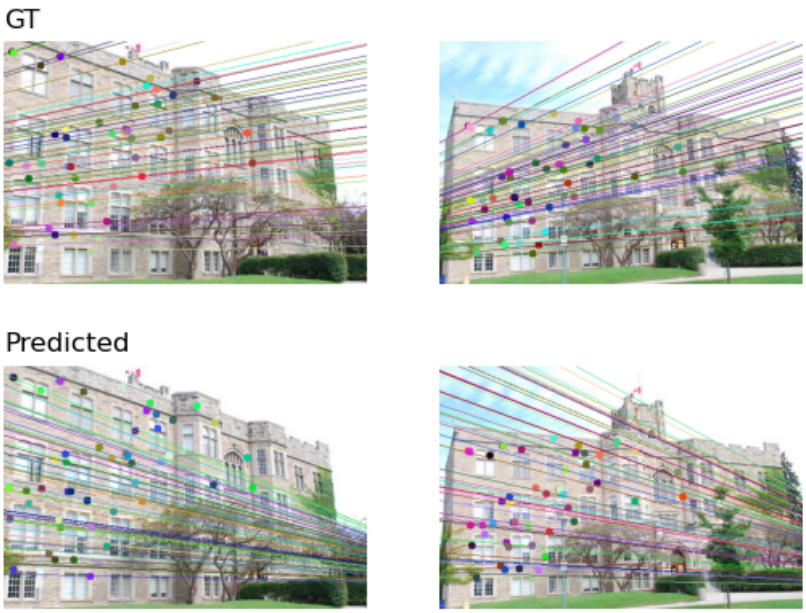
In this loss, we used the given correspondence points. In equation (5) - $x_{j,1}^{(i)}$, $x_{j,2}^{(i)}$ are the j'th pair of corresponding points, of the i'th example. As we saw in class, the fundamental matrix F should satisfy $x_2^T F x_1 = 0$, for each pair of corresponding points. Thus, for each example we calculated the mean of the distance from the epipolar constrain as the epipolar loss for this example. For each example we used a fixed number of corresponding points (in this section we used $n = 50$). The epipolar loss is a mean over all the examples in the batch. The overall loss in this section was calculated by equation (7).

$$L = (1 - \gamma) \cdot L_{pose} + \gamma \cdot L_{epipolar} \tag{7}$$

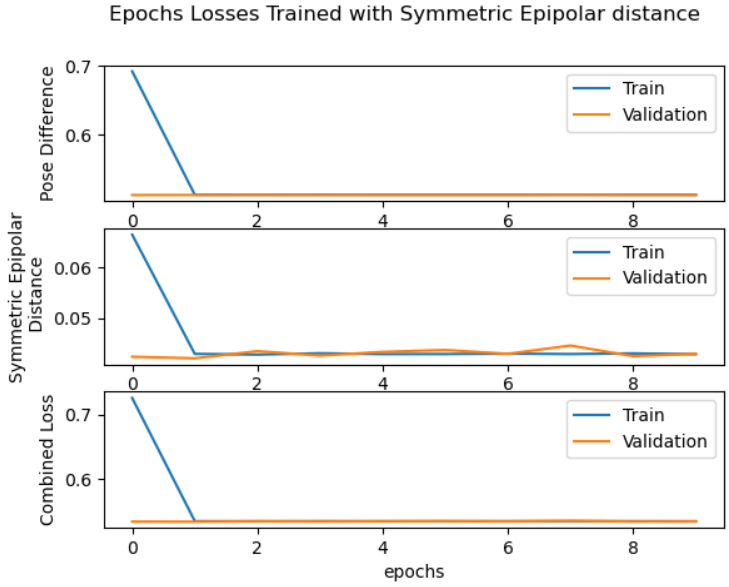
When γ is an hyper-parameter, in this case we chose $\gamma = 0.5$.

4.2.3 Results

In Figure 5 we can see the results and visualization of the model trained with epipolar loss. Figure 5a shows an example for estimated epipolar lines. We can see that the estimated epipolar lines look less similar to the GT epipolar lines, comparing to the baseline model. Figure 5b shows the training and validation errors over 10 epochs. Moreover, the mean angular distance on the validation set for the rotation matrix is 21.2, and for the translation is 58.07. Thus, there isn't any improvement from the baseline model.



(a) epipolar lines



(b) angular error

Figure 5: Epipolar loss models' results

4.3 Head extension

4.3.1 Model architecture

In this section we extend the head structure. We added one FC layer, with 256 output features. We also added intermediate batch normalization layers. The full architecture is presented in Figure 6.

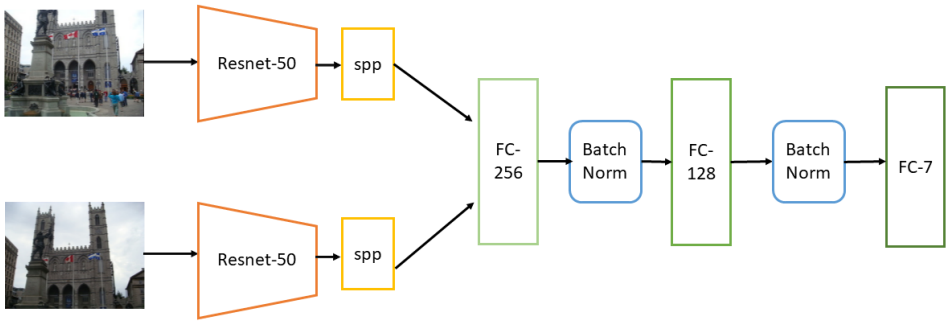


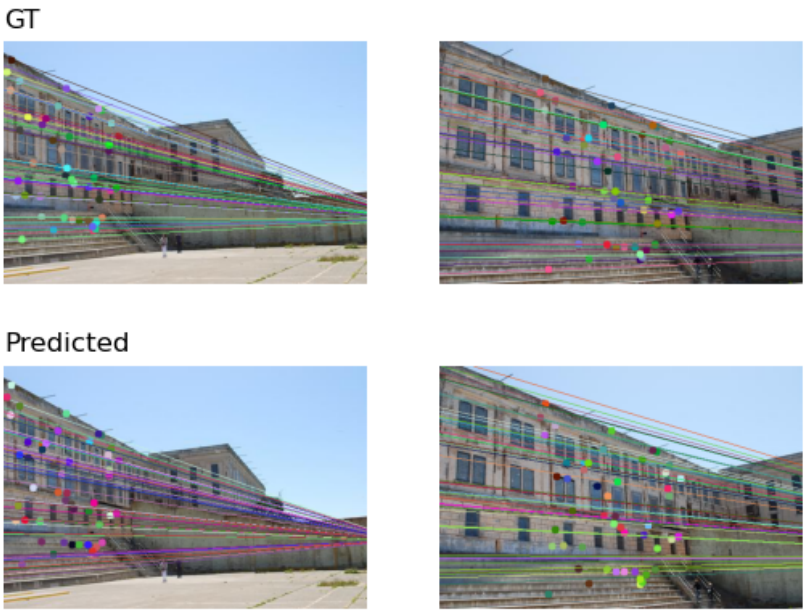
Figure 6: head extension architecture

4.3.2 Loss function

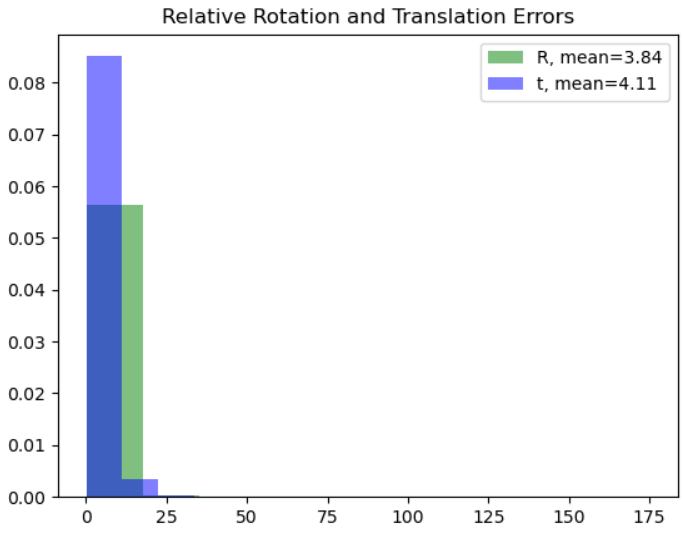
In this section we used the same loss function as in equation (7).

4.3.3 Results

In Figure 7 we can see the results and visualization of the head extended model. Figure 7a shows an example for estimated epipolar lines. Figure 7b shows an histogram of the translation angular error and the rotation matrix error. Figure 7c shows the training and validation errors over 10 epochs. In this experiment, we can see improvement in the results. In particular, we can see that the epipolar lines are more similar to the GT. Moreover, the training and validation loss and the mean angular error is a lot smaller than previous experiments.

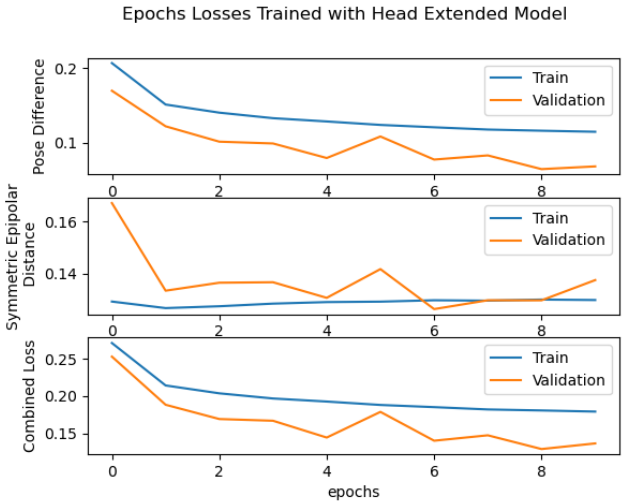


(a) epipolar lines



(b) angular error

Figure 7: Head extension models' results



(c) train and validation loss

Figure 7: Head extension models' results

4.4 Wider model

4.4.1 Model architecture

As we have see, the head extended model worked best on the validation set so far. Thus, we did another experiment, with another FC layer. The full architecture is presented in Figure 8.

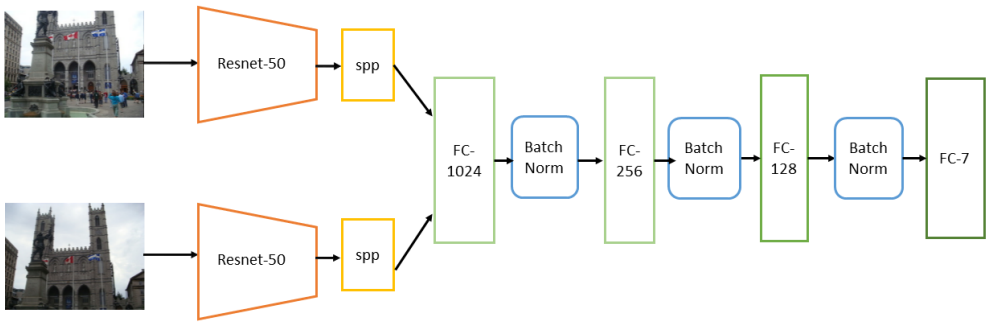


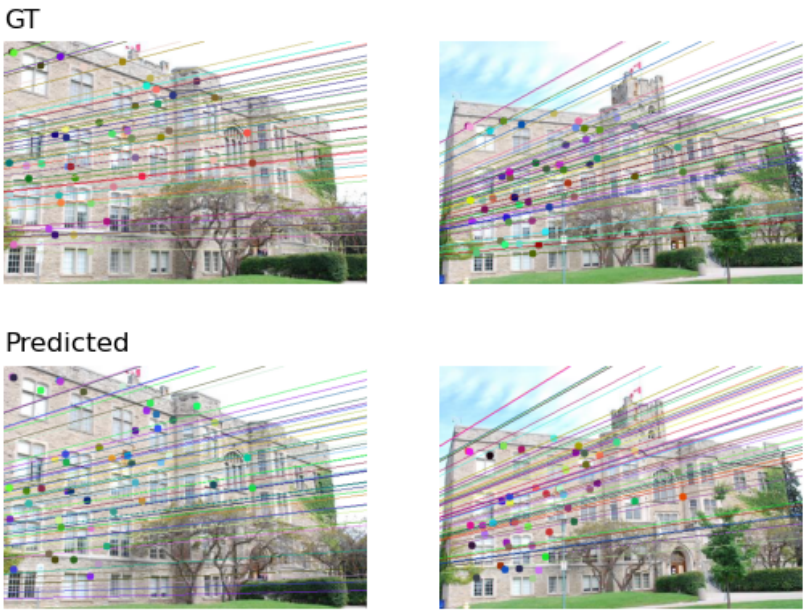
Figure 8: Wide model architecture

4.4.2 Loss function

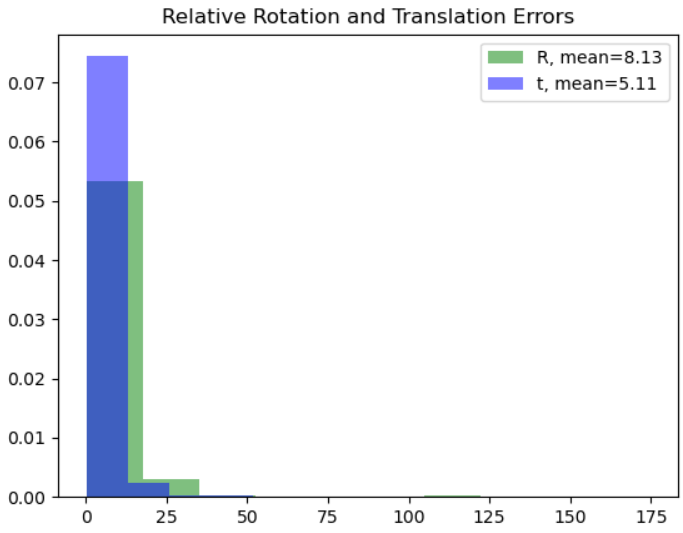
In this section we used the same loss function as in equation (7).

4.4.3 Results

In Figure 9 we can see the results and visualization of the wider model. Figure 9a shows an example for estimated epipolar lines. Figure 9b shows the training and validation errors over 10 epochs. The mean angular distance on the validation set for the rotation matrix is 8.13, and for the translation is 5.11. We can see that he wider network did not improve the results over the validation set, compared to the head extended network.

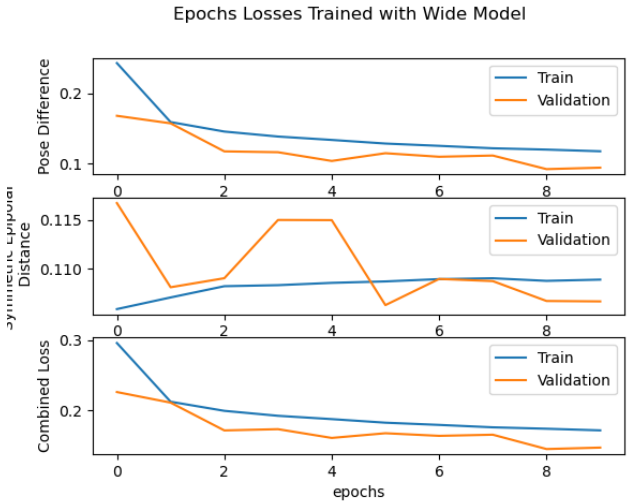


(a) epipolar lines



(b) angular error

Figure 9: Wide models' results



(c) train and validation loss

Figure 9: Wide models' results

4.5 Angular Loss

4.5.1 Model architecture

Since we saw that the wider model did not improved the results over the validation set, we used the architecture of the head extended model (Figure 6).

4.5.2 Loss function

In order to minimize the angular distance measure, we tried using a loss function that is based on it:

$$\begin{aligned} L_R &= \|(\text{vector}(R_{gt}^T R_e))\| \\ L_t &= 1 - \text{cosine-similarity}(t_{gt}, t_e) \end{aligned} \tag{8}$$

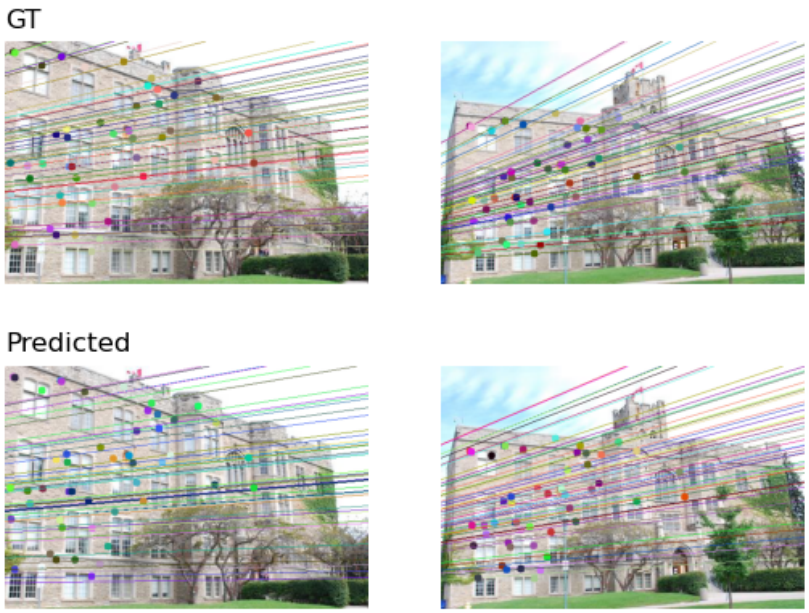
The L_t term will help us maximize the cosine similarity between the vectors of the two locations, and the L_R term will minimize the angular distance between the two rotation matrices (We had to transform the quaternion to a rotation matrix before applying the loss). The overall loss was calculated by:

$$L = L_R + L_t + L_{pose}$$

4.5.3 Results

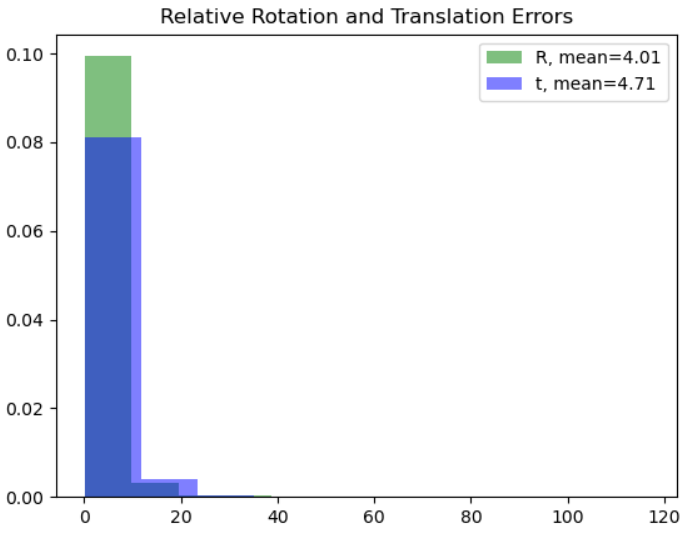
In Figure 10 we can see the results and visualization of the model trained with pose and angular loss. Figure 10a shows an example for estimated epipolar lines. Figure 10b shows an histogram of the translation angular error and the rotation matrix error. Figure 10c shows the training and

validation errors over 10 epochs. In this experiment, we can see that the performance over the validation set is almost similar to the head extended model.

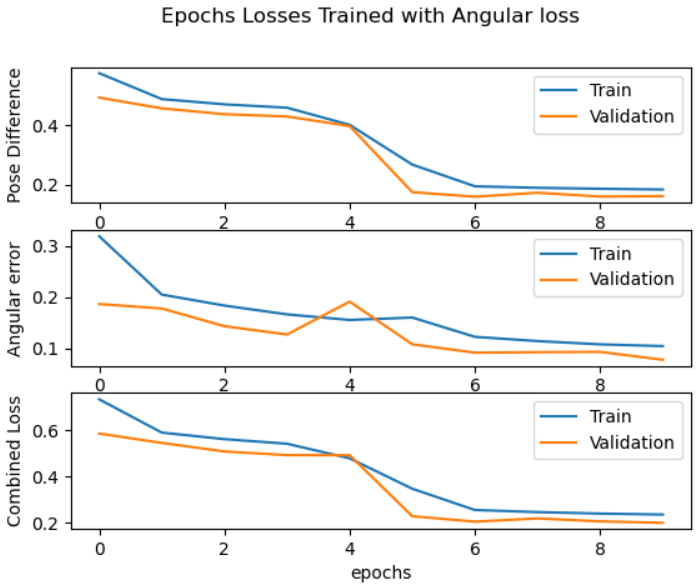


(a) epipolar lines

Figure 10: Angular loss results



(b) angular error



(c) train and validation loss

Figure 10: Angular loss results

5 Generalization

In order to determine which model works best, we examined the head extended model and the model trained on angular loss generalization abilities. This was done by training on a single scene (fine arts palace), and validating on the same scene and another one (kings college front). In this way we can compare the validation results over a known to model scene, and a scene that the model never saw, thus we can examine the generalization abilities of the model. Although both were inferior dramatically for the novel scene, the angular loss model outperformed in both same scene and new scene validation.

Validation scene	Error	Angular loss	Extended head & batchnorm
same	translation	**3.64**	4.77
	rotation	**2.27**	5.70
novel	translation	**22.26**	28.48
	rotation	**24.33**	29.26

6 The advantage of deep over the 8-point algorithm over learned scenes

In the beginning of the report we mentioned the poor performance of the 8-point algorithm on part of the validation examples. Our model had performance that are almost identical on all inputs, while the classic method seems to fail for a large subset of the inputs. We took the **600 validation inputs that had a translation error of over 50** with the classical method and tested them on our best model. As a concrete example, we can get back to input 'Lund cath large 7619' and see our performance on it. We can see in figure 11 how the model was able to overcome the hard pose.

Data	Error	8-points	Our model
600 hard examples	mean translation	12.8609	**4.96**
	mean rotation	118.7967	**4.41**
Lund/7619	translation	78.3345	**3.6004**
	rotation	14.5707	**5.7321**

It seems that our model is far less sensitive to difficult poses on scenes that it has seen in training.

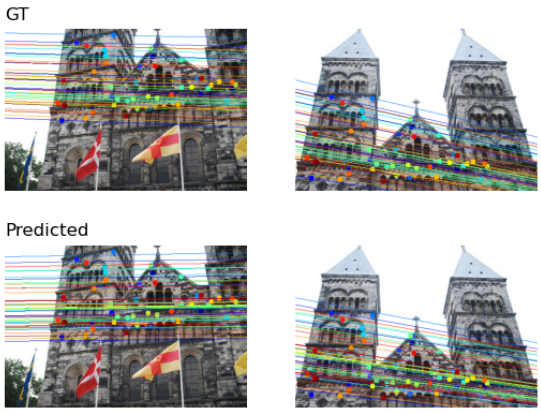


Figure 11: Our model's prediction on a validation example that was hard for the 8-point algorithm.

7 Test-set Evaluation

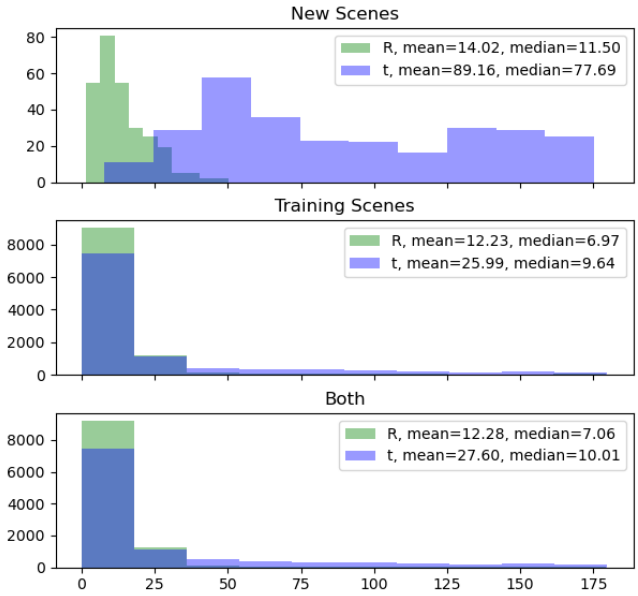


Figure 12: Our best model's performance on the test set

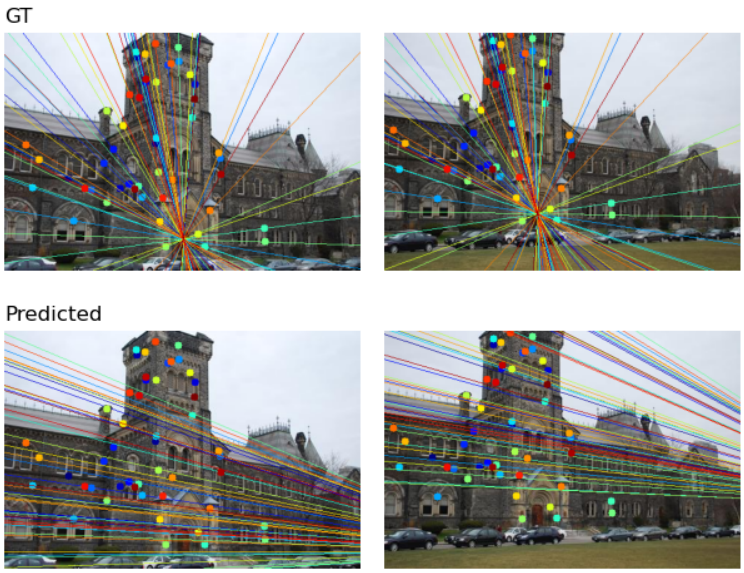


Figure 13: Poor performance of our model on a test example from a new scene.

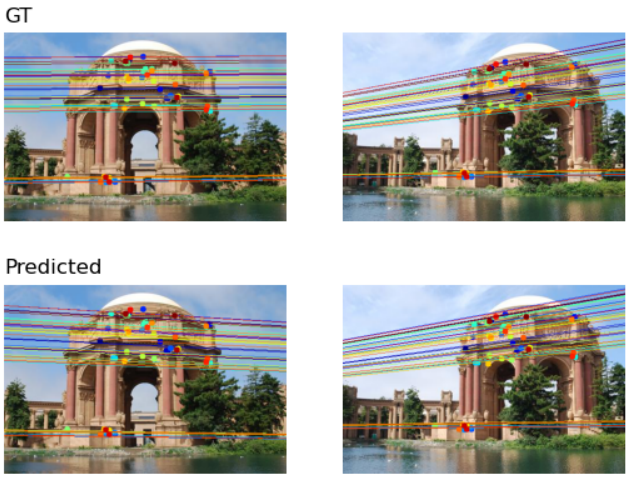


Figure 14: Good performance of our model on a test example from a known scene.

The performance on the new scenes is quit poor, but combined with the scenes that were used for training we got fair results.

Note: out of curiosity we tried the second best model - extended head with batchnorm and got better results on the test set. However, we chose the model based on the validation set and could not have guessed it would be worse.

8 Summary and future ideas

In summary - our model performed beautifuly on the validation set (it was built to perform best on it) but not as much to the unseen test examples - especially the new scenes.
We have some more ideas that we did not have time to try for this project (maybe next year) but we'll mention some of them here:

1. Evaluating generalization by training on all the scenes except one, and then using the left out scene as validation. This should be done in a "cross validation" manner, where we go over all the scenes and each time one is left out as validation.

2. Investigating the large variation in performance over different scenes - what makes a scene easier or harder for pose estimation with classical/deep methods?

3. Training on all three losses - we have tried training on the pose loss alone, or the pose with either angular or reprojection (epipolar) losses. It could be interesting to check weather the three will provide better results.

4. Separating the two branches and training each of them on paired examples for which we know which image is the left and which is on the right. (could be done using the 8-point algorithm and extracting relative translation)

5. Using convolution layers for the head of the model rather than just the branches.

References

[1] H. C. Longuet-Higgins. "A computer algorithm for reconstructing a scene from two projections". In: *Nature* 293.5828 (Sept. 1981), pp. 133–135. ISSN: 1476-4687. DOI: 10.1038/293133a0. URL: https://doi.org/10.1038/293133a0.

[2] Bolei Zhou et al. "Learning Deep Features for Scene Recognition using Places Database". In: *Advances in Neural Information Processing Systems 27*. Ed. by Z. Ghahramani et al. Curran Associates, Inc., 2014, pp. 487–495. URL: http://papers.nips.cc/paper/5349-learning-deep-features-for-scene-recognition-using-places-database.pdf.

[3] Iaroslav Melekhov et al. "Relative Camera Pose Estimation Using Convolutional Neural Networks". In: (2017). arXiv: 1702.01381 [cs.CV].