3D Augmented Reality Project B.7 Report
Deep Homography Estimation

Paolo Eccher

December 2019

1 Introduction

The goal of this project is to estimate the homography transformation that relates two picture using a Convolutional Neural Network (CNN). The architecture of the CNN is based on the one presented in the paper "Deep Image Homography Estimation" written by DeTone, Malisiewicz and Rabinovich [1].
In this project I'm going to evaluate the performance of this Network architecture in multiple scenarios. This means that I'll train models using both "Vanilla" and noisy data. At test time, I'll evaluate the performance of each model against all types of noise.
This project is done using Python3 and the Pytorch library. This is the first time that I use Pytorch (I had used Tensorflow before) and I consider this project an opportunity to learn Torch. For this reason, there could be some stuff that is not implemented in the most elegant or efficient way.
To train the models I used Google Colab[1] which offers an environment with Nvidia K80 for free.

2 How to reproduce experiments

- Download the data and decompress it in a folder called "data" inside the project directory.

```
wget http://images.cocodataset.org/zips/train2014.zip
unzip -q train2014.zip -d data
rm -rf train2014.zip

wget http://images.cocodataset.org/zips/val2014.zip
unzip -q val2014.zip -d data
rm -rf val2014.zip
```

[1] https://colab.research.google.com/

```
wget http://images.cocodataset.org/zips/test2014.zip
unzip -q test2014.zip -d data
rm -rf test2014.zip
```

- Run the experiments. Use "All" if you want to run the experiment on all the noise types. Otherwise, specify what you want to use ("Vanilla" (Default), "Blur5", "Blur10", "Gaussian", "Compression", "S&P").
  If you have downloaded a different version of MS-COCO you have to specify the year with `--year 201X`.

  ```
  python3 train.py --noise "All"
  ```

3 Homography

An Homography is a transformation that relates two images. Estimation of this transformation starting from 2 images taken from slightly different point of view is a classic problem in Computer Vision. It is typically solved by firstly extracting feature points/corners using features extraction algorithms[2] such as SIFT, ORB or Harris. Afterwards, using the RANSAC algorithm, it is possible to estimate the transformation between the two images with robustness against mismatches.
The novelty of this paper lies in the introduction of a new technique that can, in a single step, estimate the parameters of the homography with good results.

3.1 4-point parameterization

Instead of the common 3x3 matrix H_{matrix} the authors of the paper decided to use a different parameterization for this task. In fact, they adopted the so called 4-point parameterization that is based on a 4x2 matrix.
Letting $\Delta u_i = u'_i - u_i$ be the offset in the u dimension of the i-th corner we have that the 4-point parameterization matrix is constructed as follows:

$$H_{4point} = \begin{pmatrix} \Delta u_1 & \Delta v_1 \\ \Delta u_2 & \Delta v_2 \\ \Delta u_3 & \Delta v_3 \\ \Delta u_4 & \Delta v_4 \end{pmatrix}$$

It's easy to convert the H_{4point} in the H_{matrix}. For example one can use normalized DLT or the OpenCV function `getPerspectiveTransform()`.

4 Noise Effects

The noise effect that I'll consider are:

[2] Interestingly, the same authors presented another paper in which they trained a network called SuperPoint [2], for the feature extraction task with really good results.

- Blurring with an average filter of size 5x5 and 10x10.
- Gaussian Noise with Mean=0 and Var=10.
- Salt and Pepper noise with probability 20% (this means that with p=20% some pixel is set to 0, with p=%20 it is set to 255 and with p=%60 it is left unchanged).
- Compression of jpg at 60% using OpenCV's `imwrite` function.

Figure 1: Vanilla image

Figure 2: Blur 5x5 image

Figure 3: Blur 10x10 image

Figure 4: Salt and Pepper p=20% noise image

Figure 5: Gaussian noise image

Figure 6: Compressed image at 60%

5 Data Generation

To generate the data that is fed to the network I used the procedure depicted in figure 7. The code used for this task is located in the file `dataset.py` and was taken from Github User mez (as suggested by the professor).
The input data is essentially composed by two gray scale image of 128x128 pixels. As said before, the target is a vector containing the 8 offsets.
The only difference that is present with respect to the Github code is that I crop the image starting from a random position (the position **p** of image 7). Instead, in the Github code, they started from a fixed point located at coordinates 32x32. I think that my version is more similar to the one of the original paper.
The data that was used is the MS-COCO dataset. In order to keep the training time small but at the same time a non trivial dataset I'm going to use a smaller subset of the MS-COCO 2014 dataset with a size of about 4GB instead of 13GB (20k images). The validation and test size is 1GB (5k images). The version of MS-COCO used is 2014.
The dataset was downloaded at the official website.

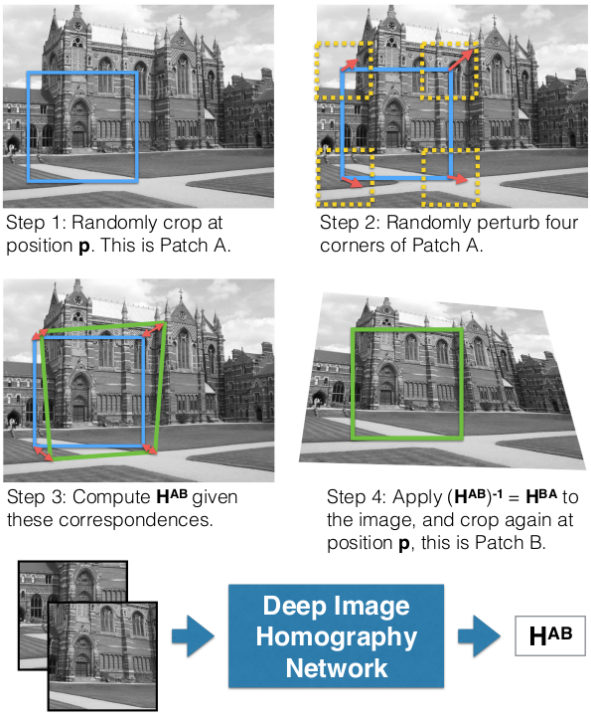


Figure 7: Generation of the data. Image taken from [1].

6 Architecture and Implementation

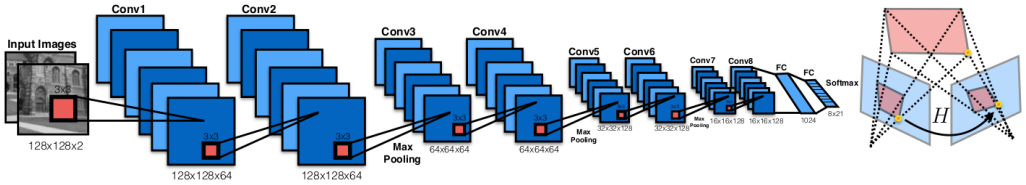


Figure 8: The architecture of the CNN. Image taken from [1].

The architecture of the network is depicted in figure 8. It is contained in the file `model.py`.
As previously said, the goal of the network is to predict horizontal and vertical offsets of the 4 corners of the image, because from this information it is easy to reconstruct the "classical" homography matrix. For this reason the output of

the network is made up of 8 numbers.
The hyper parameters used to train the models are:

- Epochs: 50
- Optimizer: Adam
- Learning Rate: 0.005
- Batch Size: 64
- Betas: 0.9, 0.999
- Weight Decay: 0

6.1 Noise Creation

To create the Blur and Compression Noise I used OpenCV functionalities. For the blurring I used `cv2.blur()` and the `cv2.imwrite()` with proper settings for the compression.
To make the Salt and Pepper noise I took a function from StackOverflow that iterates over all pixels in the image and set to 0 or 255 some pixels with probability = 20%. This function is particularly slow and before training I created a dataset with Noisy image to avoid the recomputation of the noise for every epoch. The code that is submitted calculates the noise at each epoch.
To make the Gaussian Noise I created a matrix following a Gaussian Distribution with Mean=0 and Variance=10 and then I added that matrix to the image, normalizing the result in the [0,255] range.

7 Metrics

In this section i describe the metrics that I will use to evaluate the models. Let T_{pred} and T_{tar} be the tensors that contains the 8 values that corresponds to the vertices offsets. We define:

- **L2 Norm:** $L^2_{Norm}(T_{pred}, T_{tar}) = \sqrt{\sum_{n=1}^{8}(T_{pred}[i] - T_{tar}[i])^2}$.
- **Corner Average Error:** $CAE((T_{pred}, T_{tar}) = \sum \sqrt{\frac{L^2_{Norm}(T_{pred}, T_{tar})}{4}}$ is the L^2_{Norm} divided by 4 and averaged over the whole dataset.

8 Results

↓Train | Test →		Vanilla	Blur 5	Blur 10	Gaussian	S&P	Compression
Vanilla		7,911	9,183	11,742	8,531	11,522	8,583
Blur 5		12,487	8,316	12,937	14,563	25,164	14,673
Blur 10		17,208	10,631	7,551	17,299	34,481	17,232
Gaussian		11,455	11,434	11,178	11,484	12,231	11,524
S&P		19,413	24,481	26,475	19,467	12,893	19,939
Compression		10,215	10,165	9,981	10,155	11,145	10,265

Figure 9: Results obtained on different data sets

In figure 9 I report the results obtained on test set (5k images taken from MS-COCO Test Set 2015) using the Corner Average Error metric. Results obtained by a model trained on a type of data are collected by row. In yellow I highlighted the model that achieved the highest accuracy for that type of test data.
We can see that generally a model trained on some data performs well on that type of data. This is not a surprise. However, in some cases, models trained on a different noise performs better than the ones trained on the same noise. This happens for the Gaussian, S&P and Compression noises.
What follows is a brief discussion on each model summarizing its results.

8.1 Vanilla

This is the best model since it performs well on all the noise effects. It is the best model for Vanilla, Compression and Gaussian data. It also has very good performance on the other types of noises. If we analyze the noisy images from a qualitative point of view we can see that there are small differences between Vanilla, Gaussian and Compression and this explains its high performance.

8.2 Blur 5

This model performs well on all the type of noise that blurs the images (Blur 10, Gauss, Compression). Also on Vanilla it has an acceptable error. However, on S&P, its performance drops significantly. In fact, this kind of Noise alters the image in a different way w.r.t. the other noises because it insert a lot of white and black pixels and, since the Neural Network is trained on data that doesn't contains positive or negative peaks it is easily fooled. Moreover, the S&P used is really strong (perhaps too strong) since it alters 40% of pixels.

8.3 Blur 10

This model behave in the same way of Blur 5. It has high performance (although worse) on data that was warped with blurring effect and low performance on data in which strong contrast were introduced. In fact its performance is of 34 on S&P noise, the worst of any combination of Model-Test. It must be said that the S&P noise considered in this project is really strong, probably too much. A 10% probability, that is more realistic, would have given better results also with Blur models.

8.4 Gaussian

This models is very similar to the vanilla one since it has an acceptable error on all the data types. Interestingly, it is not the best models on the Gaussian test since the Compression and Vanilla models performs better.

8.5 S&P

The Salt and Pepper model is probably the worst of whole collection. It has acceptable error only on the S&P data. Although it is not the best model on S&P data it is not far from the best model that is the Compression one. Probably a probability of 20% is too high to obtain good results on other type of noises.

8.6 Compression

Similarly to the Vanilla and Gaussian models the Compression models is able to generalize well on all the test sets. It has the highest performance on S&P data and the second best on Vanilla, Blur10 and Compression. This is the second best model.

9 Qualitative Results

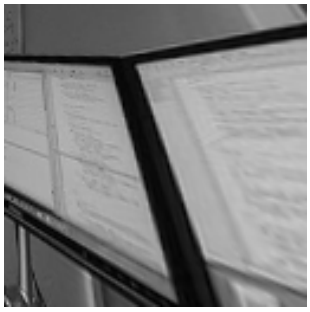
Results obtained using the Vanilla model on data without noise.


$L^2_{Norm} = 5.3595$


$L^2_{Norm} = 14.5854$


$L^2_{Norm} = 7.66384$


$L^2_{Norm} = 12.02581$

$L^2_{Norm} = 27.57509$

References

[1] Andrew Rabinovich Daniel DeTone Tomasz Malisiewicz. "Deep Image Homography Estimation". In: (2016).

[2] Andrew Rabinovich Daniel DeTone Tomasz Malisiewicz. "SuperPoint: Self-Supervised Interest Point Detection and Description". In: (2018).