Supplementary materials for:

Multimodal image alignment through a multiscale chain of neural networks with application to remote sensing

Armand Zampieri[1], Guillaume Charpiat[2], Nicolas Girard[1], and Yuliya Tarabalka[1]

[1] TITANE team, INRIA, Université Côte d'Azur, France
[2] TAU team, INRIA, LRI, Université Paris-Sud, France

1 Neural network architecture details

See Figures 4 and 5 for further details about the architecture and meta-parameters.

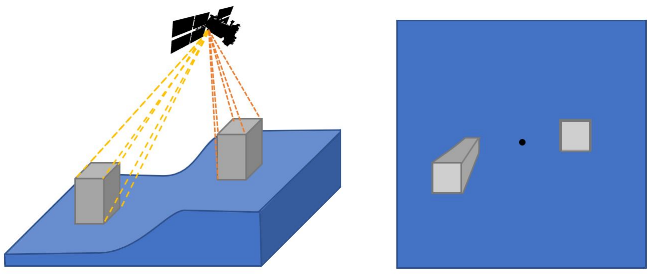
2 Keypoint matching

The keypoint matching experiment shown in the figure 7 of the main paper is shown full resolution here in Figure 6.

Supplementary matching examples are shown in Figure 7.

3 Causes and amplitude of deformations

The causes of deformations in remote sensing are mainly: the relief together with the often-high angle of the satellite/plane view (houses on top of a hill do not appear at the same place as the ones on the bottom) and the numerous mistakes done by the Humans creating the cadastral maps. Also, the top of buildings is significantly shifted with respect to their floor. The problem of registration (which has to be done w.r.t. floor location) becomes all the more harder. Moreover, trees can cover roads and roofs, shadows create strong non-relevant shapes... making registration with the cadastre difficult. Note that the typical amplitude of displacements expected is much bigger than *e.g.* for brain imaging.



4 Example of deformation

As explained in the article, to augment the dataset size for training, we generate random deformations, as a mixture of Gaussian functions with random shifts. An example of such a deformation (amplified 4 times for better visualisation purposes), and the result of its application to the original image, are shown in Figure 1.

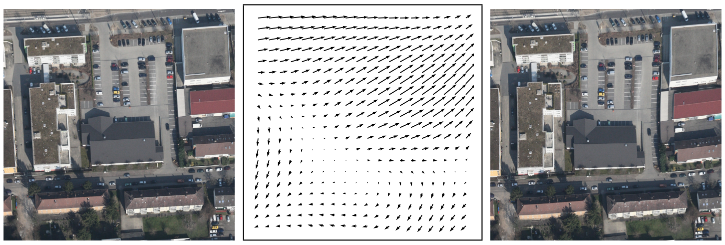


Fig. 1. Example of deformation. Left: an image I; middle: a deformation ϕ, *i.e.* a $\mathbb{R}^2$ vector field; right: the associated deformed image $I \circ \phi$.

5 Stereovision

In the paper we suggest to try the same framework on a different task, the one of stereovision. The result, shown in Figure 2 without any tuning, is very promising and confirms the generalization ability of our approach.

6 Training details

6.1 Tricks for better training

To reduce the training time and possible memory issues, patch of images (256×256 pixels) were given to the network for the training instead of the whole image, reducing the amount of computations needed per mini-batch. This is also important in terms of memory usage of the network as original images contain 5000×5000 pixels. Furthermore, neural network computations and data generation (a random transformation is generated for each image at each training step in order to augment the training set size) are parallelized in order to improve the training speed of the algorithm.

Another issue encountered was to reach the local minimum corresponding to outputting always a null deformation, thus preventing the neural network parameters to evolve towards a better optimum. To solve this issue we used several

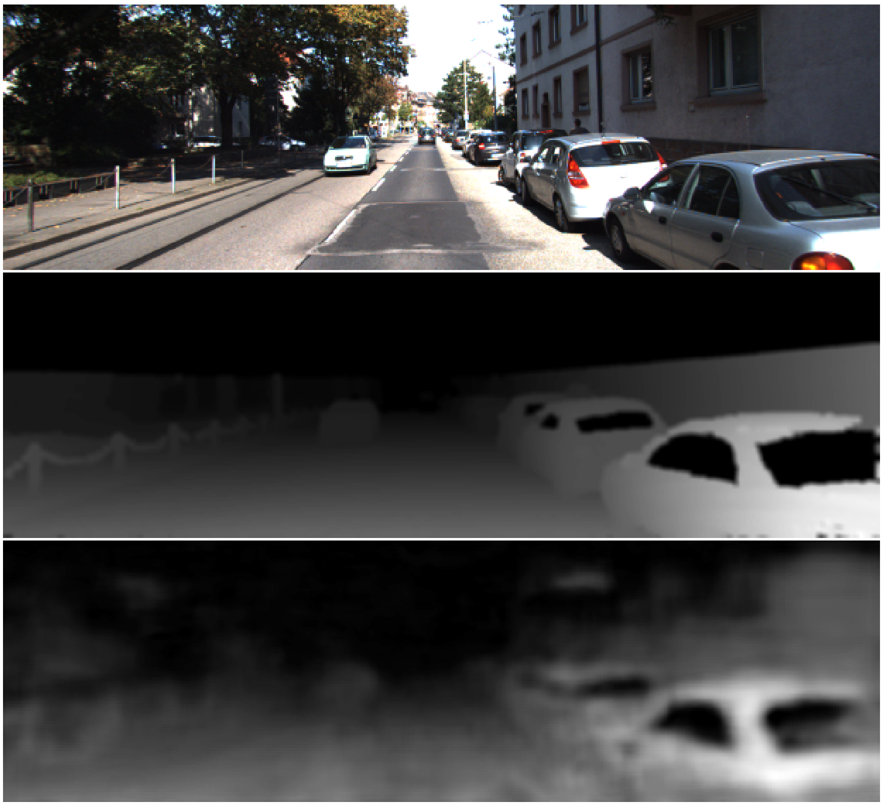


Fig. 2. Stereovision test. Top: right image; middle: ground truth depth map for right image; down: predicted depth map.

methods to facilitate the training of the network and reduce its probability to reach this local minima.
The first technique used was to force the network to overfit a very small sample of the dataset (400 iterations on 4 images with random transformations). This proved to be particularly efficient at avoiding the null local minimum.

The second technique is specific to the dataset used, *i.e.* the data from the cadastral images is particularly sparse. The first step is then to check, before selecting any training example, that data needed for alignment is present on the cadastral image, *i.e.* the cadastral image does contain buildings, *e.g.* This was done simply by calculating the ratio between labeled and unlabeled pixels (cadastral/non-cadastral) on each candidate image patch, and by setting a range of accepted values (*e.g.*, the minor class of an image should represent at least 5% of the labels of this image). Patches not respecting this rule were not selected when forming mini-batches. Thus mini-batches contained only relevant examples.

Another issue linked to the sparsity of the cadastre arises when the network is training on parts of a patch where not enough data is present to determine the transformation needed (*e.g.*, only one class within an area, as in a garden for example), even to a human eye: the estimation of the offset was not possible due to the lack of information. To solve this problem, we increase the weight of the loss function on the boundaries of buildings (parts where the transformation can be well estimated). For this, we first detect building boundaries based on the cadastre, as shown in 3, and add an multiplying factor to the loss at such locations (which is equivalent to sampling more often there). This insures that the deformation is findable and that the training is useful. This last trick is specific to our dataset, which is binary labelled, but we show in experiments that this is not needed when the dataset is not sparse or binary.

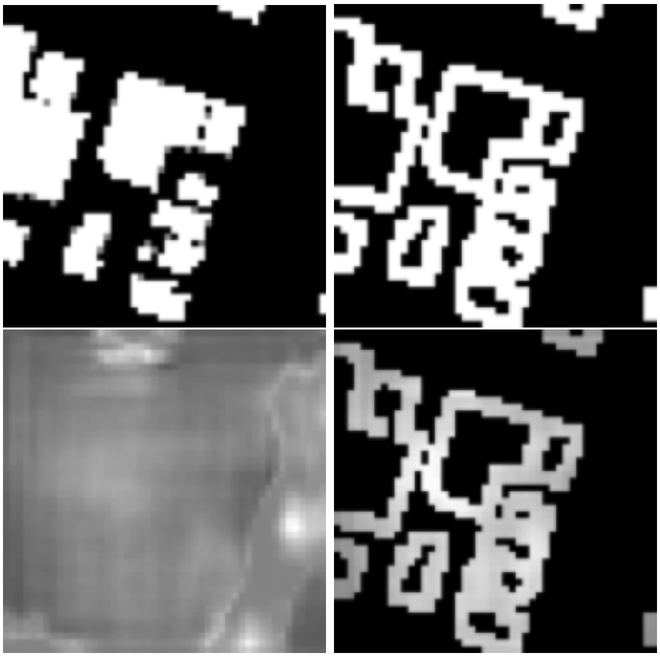


Fig. 3. Building boundaries used to re-weight the loss function. Top left: cadastral image; top right: building boundary mask used; bottom left: the original map of the loss function; bottom right: masked map of the loss function on boundaries. The loss will be multiplied with a constant factor in these areas.

Lastly, we observed minor aligning issues when dealing with rows of houses with common walls, due to local translation invariance of the images, which adds locally a degree of freedom for alignment along the row axis. We decided to give supplementary information corresponding to the location of all corners extracted from the OpenStreetMap vectorial image (as each corner of each house is indicated), hoping to help to guide the alignment along such translation-invariant line in the cadastre. This step is however not critical as the results improvement is small and specific to certain building geometries (row of identical houses with common walls).

6.2 Training information

Number of iterations : 60 000
Batch size : 16
Time to train : 16 hours
Number of images : 108 original images (with a random transformation generated at each iteration for each image)
Original image size : 5000×5000
Patch size : 256×256
Total number of layers : 26
Memory used with tensorflow : 9.7 GB
GPU : GeForce GTX 1080
Processor : Dual-Xeon E5-2630
RAM : 64 GB

7 Alignment framework

The whole processing framework for the alignment of OpenStreetMap cadastral information with aerial images is summarized in the chart shown in Figure 8.

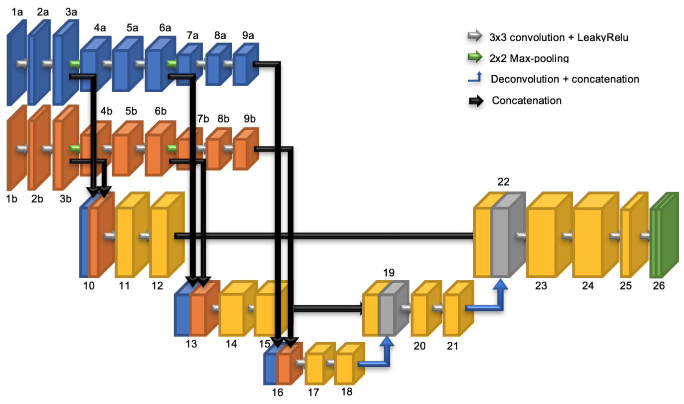


Fig. 4. Network architecture. The two input images I_1 and I_2 are fed to layers 1a and 1b respectively. The output is a 2 dimensional vector map (layer 26 with 2 channels). This architecture allows to merge information from both sources at all scales, to extract high-level information, and to remember fine details from the input resolution to output a precise full-resolution deformation. Details on Figure 5.

Start layer	End Layer	Name	Kernel size	Number of filters	padding	stride
1	2	convolution-1	5	16	2	
2	3	convolution-2	5	32	2	
3	4	pooling-1	2			2
4	5	convolution-3	3	32	1	
5	6	convolution-4	3	32	1	
6	7	pooling-2	2			2
7	8	convolution-5	3	64	1	
8	9	convolution-6	3	64	1	
3	10	concatenation-1				
6	13	concatenation-2				
9	16	concatenation-3				
10	11	convolution-7	3	32	1	
11	12	convolution-8	3	32	1	
13	14	convolution-9	3	64	1	
14	15	convolution-10	3	64	1	
16	17	convolution-11	3	64	1	
18	19	convolution-12	3	64	1	
18	18'	deconvolution-1	3	64		
15-18'	19	concatenation-4				
19	20	convolution-11	3	64	1	
20	21	convolution-12	3	64	1	
21	21'	deconvolution-2	3	32		
12-21'	22	concatenation-5				
22	23	convolution-13	3	64	1	
23	24	convolution-14	3	64	1	
24	25	convolution-15	3	32	1	
25	26	convolution-16	3	2	1	

Fig. 5. Details for each layer of the (scale-specific) neural network displayed on Figure 4. "Kernel size 3" for a convolutional layer means "3 × 3" convolution.

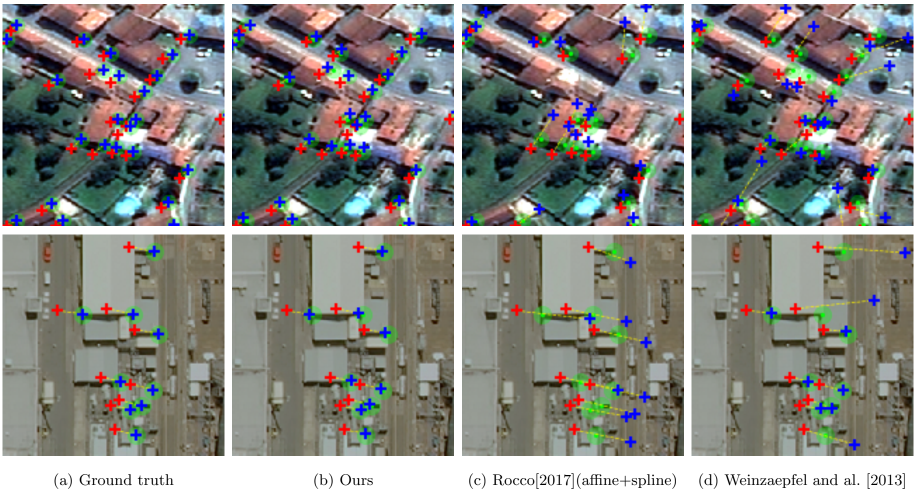

(a) Ground truth (b) Ours (c) Rocco[2017](affine+spline) (d) Weinzaepfel and al. [2013]

Fig. 6. Multimodal keypoint matching comparison for different methods and two datasets. Top: Forez dataset; bottom: Kitsap. Blue: predicted, green: ground truth (centers of the green circles), red: original location of the corner (from the OpenStreetMap cadastral image, which is mis-geolocalized with respect to the RGB image).

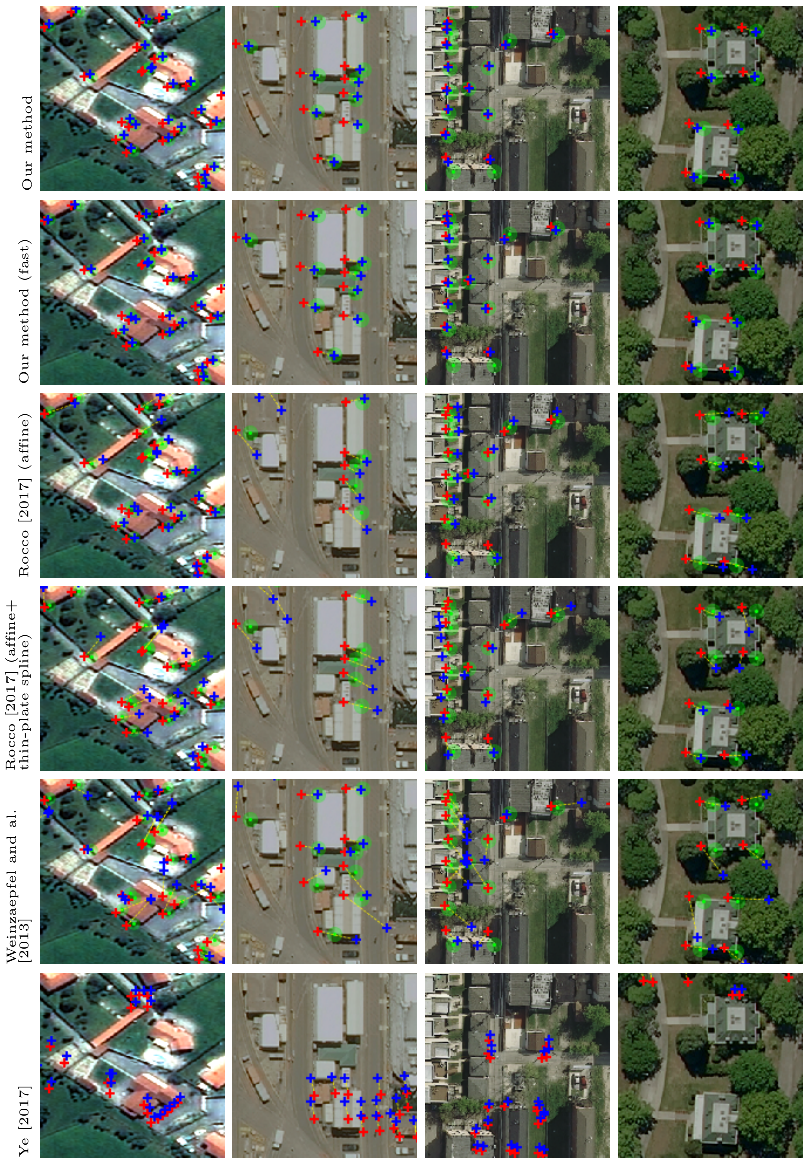

Our method
Our method (fast)
Rocco [2017] (affine)
Rocco [2017] (affine+ thin-plate spline)
Weinzaepfel and al. [2013]
Ye [2017]

Fig. 7. Additional multimodal keypoint matching examples. Same setup as in Figure 6.

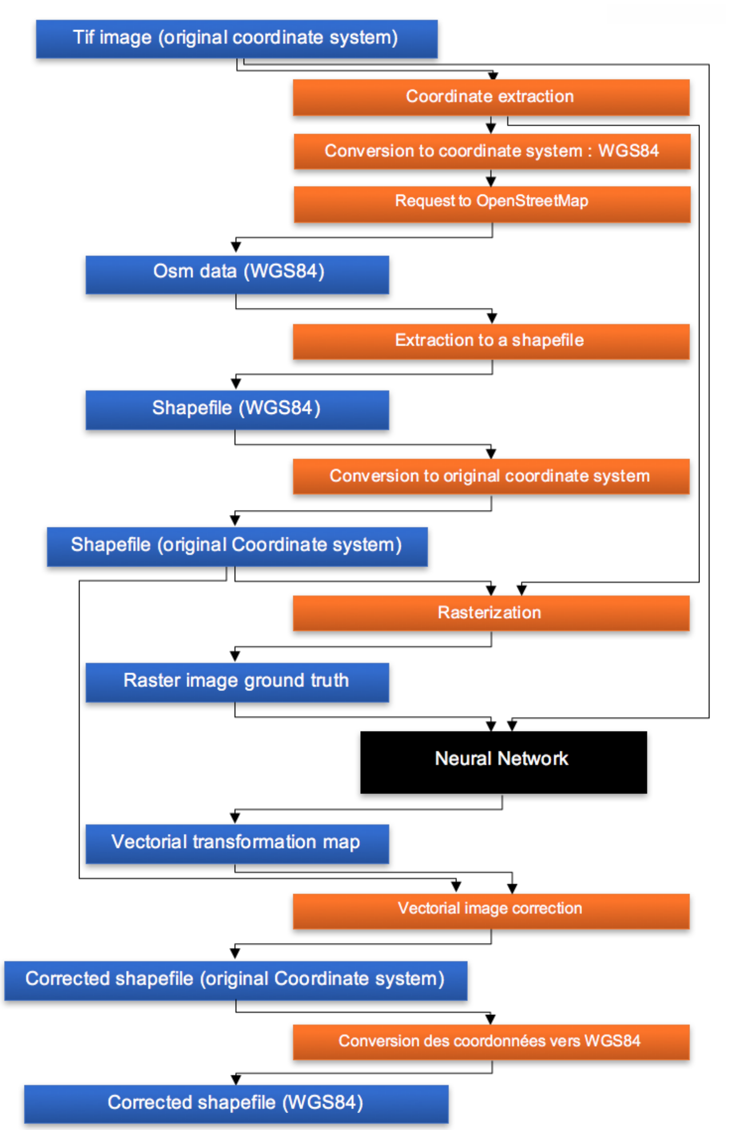

Tif image (original coordinate system)
Coordinate extraction
Conversion to coordinate system : WGS84
Request to OpenStreetMap
Osm data (WGS84)
Extraction to a shapefile
Shapefile (WGS84)
Conversion to original coordinate system
Shapefile (original Coordinate system)
Rasterization
Raster image ground truth
Neural Network
Vectorial transformation map
Vectorial image correction
Corrected shapefile (original Coordinate system)
Conversion des coordonnées vers WGS84
Corrected shapefile (WGS84)

Fig. 8. Global framework for OpenStreetMap data correction.