*Supplemental Material*

# Efficient local correlation volume for unsupervised optical flow estimation on small moving objects in large satellite images

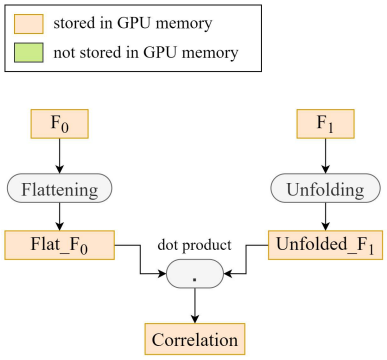

Sarra Khairi
Inria, Rennes, France
sarrakhairi2000@gmail.com

Etienne Meunier
Inria, Rennes, France
etienne.pj.meunier@gmail.com

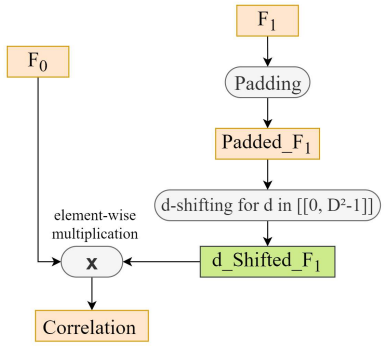
Renaud Fraisse
Airbus Defence&Space, Toulouse, France
renaud.fraisse@airbus.com

Patrick Bouthemy
Inria, Rennes, France
Patrick.Bouthemy@inria.fr


## 1. Efficient correlation computation




(a) Per-pixel computation




(b) Per-shift computation

Figure A1. Efficient correlation computation

Figure A1a introduces the straightforward approach of computing the local correlation volume. For each pixel **p** in $I_0$, we extract the neighbourhood of its corresponding pixel **p'** in $I_1$ and compute the correlation between **p** and all the pixels within the local neighbourhood of **p'**. This computation involves the `torch.nn.Unfold` class, an operation that extracts sliding local blocks from the batched input tensor `fmap1`. For an input tensor of size $N \times C \times H \times W$, where $H$ and $W$ are respectively the height and width of the image, $C$ the number of channels and $N$ the batch size, and a $D \times D$ kernel, the operation outputs a tensor of size $N \times (CD^2) \times L$, where $D^2$ is the total number of values within each block and $L$ is the total number of such blocks ($HW$ in our case). The main drawback of the `Unfold` operation is that it extracts the values in the local blocks by copying them from the large input tensor. This considerably increases the memory usage when dealing with large-size images. An alternative to the *per_pixel computation* is the *per_shift computation* presented in Figure A1b.

A per-shift computation is less memory consuming, since it directly operates on the whole feature maps, and does not require to store the features of the neighbors for every pixel in the source image. Also, instead of building the tensor `unfolded_fmap1` of size $N \times C \times (2R + 1)^2 \times (HW)$, with $D = 2R + 1$, we construct a smaller tensor `padded_fmap1` of shape $N \times C \times (H + 2R) \times (W + 2R)$. Following this approach, we need to pad the target feature on all sides, and compute the matching costs for each shifting position.

## 2. Efficient memory management during inference

Once we have designed an optimized correlation volume and prepared the training settings

(choice of hyperparameters, training set, etc.), the next goal is to see whether or not it is possible to perform inference on the entire satellite images without the need to crop them. Unfortunately, unless we change the way some tasks are performed, it is not possible to test the model on large full-frame images. Indeed, the inference on these images lead to a *"GPU out of memory"* error, which means that there is not enough memory available on the chosen GPU to complete the task. To identify the source of this issue, we suggest isolating the blocks of the architecture that are more prone to generating memory leaks.

In the following observations, it is assumed that the image size is equal to $[N, C, H, W] = [1, 3, 2400, 1400]$ to get as close as possible to the size of satellite images. Tests on both the feature encoder and the update block are performed after gradient computation is disabled (using *torch.no_grad()*).

## 2.1. Feature encoder

To demonstrate that there is no memory leak in the feature encoder, we isolate this block and construct M consecutive pairs of feature maps by calling the feature encoder multiple times. We will then show that the high memory consumption of the encoder is due to its complexity, rather than a misuse of GPU memory. Feeding the images into the feature encoder multiple times does not increase the memory, which demonstrates that GPU memory is released after every iteration and that there is no memory leak. Therefore, our intuition is that the feature extraction block requires significant memory due to its complexity. As we decrease the number of output channels from 128 to 96, the global free GPU memory after feature extraction increases from 22.7% to 51.7%. We will see in the following sections that such a design does not affect the performance of our model on satellite images.

Once we have built the correlation pyramid, we can remove the feature maps from the GPU as they are not useful for future tasks. This significantly increases the total memory available.

## 2.2. Update block

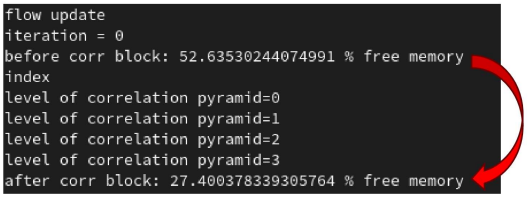
The correlation pyramid occupies 35% of the GPU memory, which significantly reduces the memory available for upcoming tasks. Before updating the flow, we will therefore move this pyramid to the CPU to save GPU memory and send one correlation volume to CUDA only when it is needed.

Each flow update is preceded by a correlation lookup that concatenates the similarity costs at different pyramid levels to form the correlation feature map CF. There is no memory leak within the pyramid indexing function, since the number of stored variables does not increase as we move to upper levels.

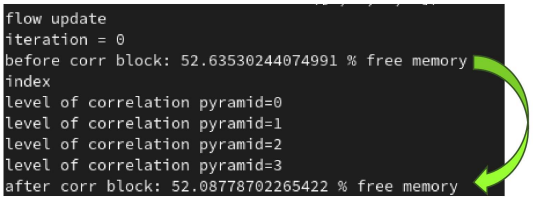
PyTorch uses a caching memory allocator to speed up memory allocations and deallocations (see memory management in Pytorch website). In some cases, however, there is still a delay before memory is released, creating a memory peak for upcoming tasks. To optimize its use, we will manually release local variables within the indexing function using `del variable`.

Once we calculate the correlation features and before outputting the result, we will manually remove unnecessary local variables, namely `sampling_coords`, `indexed_corr_volume`, `centroids_coords` and the last correlation volume that is stored in the GPU.

If we do not remove local variables in the pyramid indexing, the variables generated in the index function occupies 25.24% of the memory. The remaining memory (27.40%) is not enough to compute the next step, namely the flow update. Removing these variables enable the process to go on, as 52.08% of the memory is still free as shown in Fig.A2.




(a) before




(b) after

Figure A2. Global free GPU memory before and after removing local variables

In order to avoid memory peak errors while running inference on satellite images, the last precaution that we must take concerns the `index_pyramid` function. Instead of indexing the restricted correlation volume all at once (for all

the pixels in the source image), we compute this indexing per chunks. The number of chunks depends on the number of pixels contained in the source image. The more pixels, the greater the number of chunks (denoted `nb_batches` in the function.

Once we have optimized the memory within the indexing function, we can focus our attention on how the GPU memory behaves from one iteration to the next. There is no memory leak within the update function, since the number of stored variables does not increase as we move from one iteration to another.

Due to the memory consumption caused by the gradient graph construction, training our model on large-size satellite images is impossible in the initial setting. This section has shown that, when introducing significant improvements, it is possible to run inference on the entire images without having to crop them.

## 3. Dataset Description

The dataset consists of sequences of aerial images that have been acquired in order to be representative of future satellite video acquisitions at sub-metric resolution. It contains stabilized sequences of 50 to 100 frames. This set can be divided into two categories: sequences representing cities (Table A1) and sequences representing highways (Table A2). Let us add that there are several image sequences acquired over the Placa Tarraco in the test set.

In order to get a rough approximation of the hyperparameter $R$, which represents the search radius of the local correlation volume, we need to quantify the displacement range in both image categories. For this purpose, we will compute the absolute difference between two consecutive frames after averaging their three color channels. Figures A4 and A5 contain regions of interest in four different sequences, namely *A7C4_Sagrada_50cm*, *PlacaTarraco_Lion_5hz*, *A002C001E8_CenterCo_50cm* and *A4C1R9_Peage_50cm*.

We observe that the range of motion of vehicles in cities is about 3 pixels. On highways, this range is about 4 to 6 pixels. Therefore, the search radius R must be of the same order as these quantitative estimations. In order to have a search domain that is large enough to capture all the possible motions, this hyperparameter is set to 12. These regions of interest also provide us with information about the dimensions of moving vehicles: size of cars is between 4 and 10 pixels.

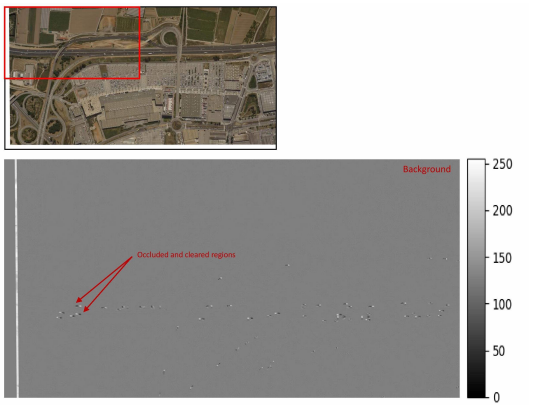


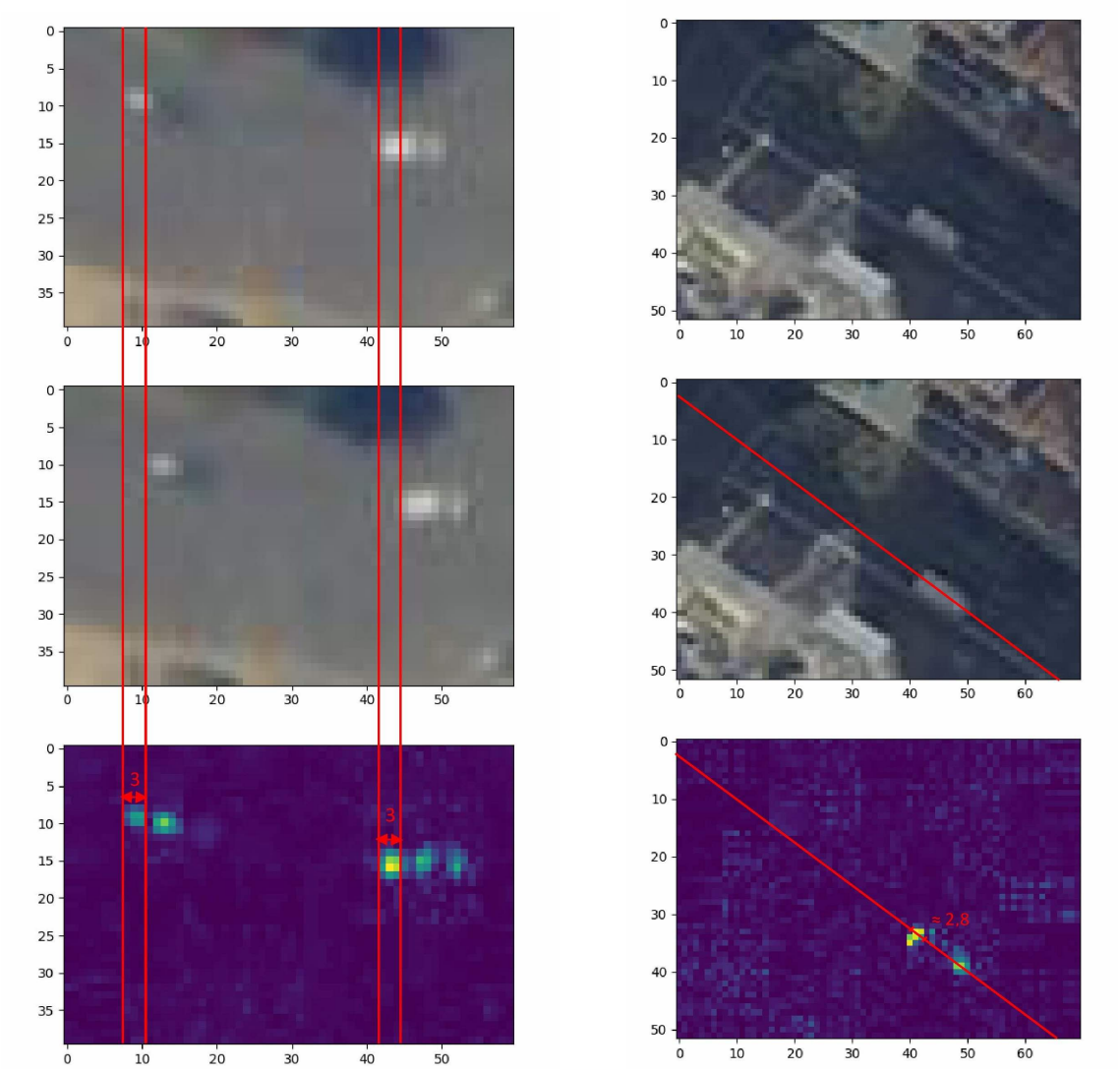

Figure A3. Stabilized satellite images. Top row: one of the two consecutive images. Bottow row: The temporal image difference computed in the image crop delineated in red above.

The difference between two consecutive images also proves that the sequences are actually stabilized. As shown in Figure A3, this difference is indeed equal to zero in regions associated with the background (gray regions). The non-zero pixels generally correspond to occluded and disoccluded regions, and more precisely to the front and rear parts of vehicles. We can also see that some pixels corresponding to truck roofs are equal to zero. In fact, if the size of an object of uniform intensity or color is greater than its displacement, there will be an overlap region. Therefore, the temporal difference between two successive images in this specific region is very close to zero.

This difference is obtained by averaging the three color channels of each RGB image, calculating the difference between them, adding 255 to all pixels and dividing the result by two so that their values are bounded between 0 and 255 (128 corresponding to a difference value of 0, 0 to 127 to negative values, 129 to 255 to positive values).

## 4. Mosaic and crop experiments regarding RAFT

In order to have a manageable memory consumption, RAFT constructs feature maps downsampled by a 1/8 factor. However, a low-size feature map cannot fully represent the fine structures of an image, which makes the model unsuitable for our application. In order to confirm our intuition, we conducted two experiences with the RAFT model. Flows are displayed using the usual HSV color code as illustrated in Fig.A6.




(a) ROI in PlacaTarraco_Lion_5hz

(b) ROI in A7C4_Sagrada_50cm

Figure A4. Displacement range in sequences representing cities

### 4.1. Mosaic experiment

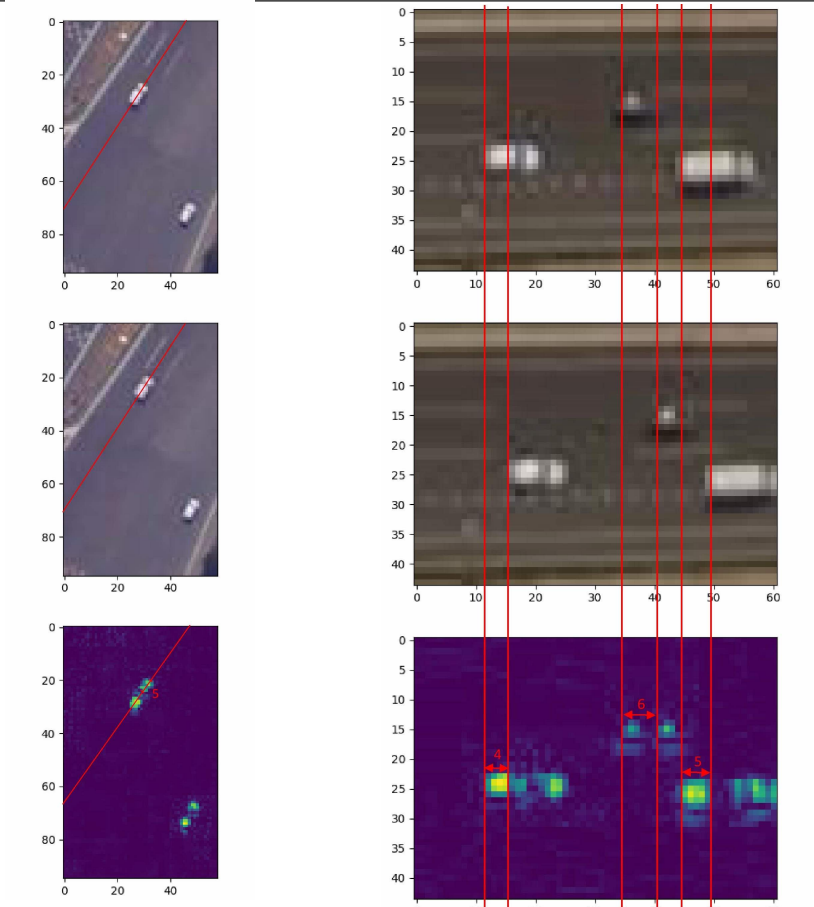
The first experiment is intended to demonstrate that small objects tend to disappear when shrunk. The inference is performed first on a mosaic of DAVIS2016 images[1], and then on each DAVIS2016 image individually. The mosaic contains a total of 49 images arranged in 7 rows and 7 columns. The mosaic is the result of combining the DAVIS2016 images into one large image. For a given position in the mosaic, the first frame represents a DAVIS2016 image at time $t_0$, while the second frame represents the same DAVIS2016 image at time $t_0 + 1$. The mosaic is further downsampled so that objects become smaller. Figure A8 shows that small objects disappear, such as the man parachuting in the image at position $(4, 2)$ and the car in the image at position $(7, 3)$. Some of the objects that are missing in the flow predicted on the lower-resolution mosaic are circled in green. In order to make a meaningful comparison with individual flow predictions, the flow was normalized per instance and not over the whole mosaic for the color representation.

This experiment shows that objects that were initially captured by RAFT (Fig.A8c) disappear when shrunken (Fig.A8d). The downsampling in the feature extraction module has certainly discarded these objects.

### 4.2. Crop experiment

The goal of the second experiment is to demonstrate that moving objects whose flow is initially

[1] https://davischallenge.org/index.html




(a) ROI in A4C1R9_Peage_50cm

(b) ROI in A002C001E8_CenterCo_50cm

Figure A5. Displacement range in sequences representing highways

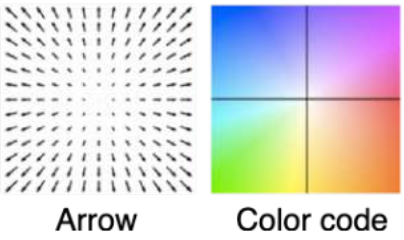



Figure A6. Usual flow field visualization using the HSV color code

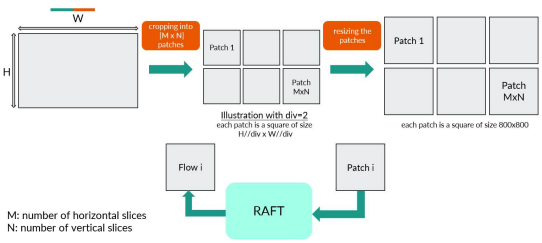


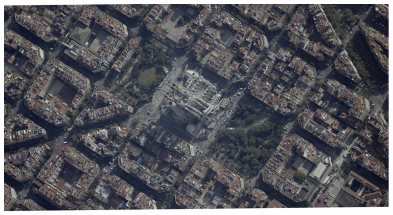

Figure A7. Principle of the crop experiment (the term "patch" used in the figure is equivalent here to "crop").

omitted by RAFT, may be detected if we increase their size. To do so, large-size aerial images are first cropped into small sub-images, each of which is upscaled to an $800 \times 800$ frame (note that the dimensions are divisible by 8). Flow estimates are then calculated for each crop. In order to increase the size of moving objects, we use bilinear upsampling. We will compare flows obtained with different upscaling factors (denoted *div* in Fig.A7a). Let us note that the size of the cropped images determine the upscaling factor. The flow will be predicted using the same trained RAFT model as in the previous experiments.

In Fig.A9a, we observe that combining the flow of all crops leads to flow discontinuities in overlapping regions. To get rid of these sharp edges between crops, we have implemented the tiling technique described in FlowFormer[2]. For each pixel that is covered by several tiles, we compute its output flow $\mathbf{w}(\mathbf{p})$ by blending the predicted flows

[2]Z. Huang et al. FlowFormer: A transformer architecture for optical flow. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, New Orleans, 2022.

Table A1. Examples of sequences depicting cities

| sequence | description |
|---|---|
|  PlacaTarraco_Lion_5hz **size**: $1800 \times 950$ | This sequence is centered on a roundabout. It mostly contains cars. Their speed is about 3 pixels per frame. It certainly corresponds to the speed allowed for road vehicles within towns. |
|  A6C15_Glories_50cm **size**: $1423 \times 775$ | This sequence represents a large square in Barcelona. We can see a skyscraper that undergoes an apparent motion due to its height. This apparent displacement of objects when the camera is moving is called motion parallax. This phenomenon was not compensated by the stabilization algorithm. It could affect the estimation of optical flow, since it is usually greater than the motion of small moving vehicles. |
|  A7C4_Sagrada_50cm **size**: $1370 \times 732$ | This sequence represents the surrounding of Sagrada Família in Barcelona. Similar to the previous sequence, motion parallax arises since the church tours are high. This is also an example of image where the brightness constancy assumption (BCA) attached to moving pixels is violated, since vehicles move under the shadow cast by the church. |
|  A7C5_Sants_50cm **size**: $1381 \times 754$ | This sequence depicts a neighbourhood in the southern part of Barcelona. Like previous sequences, it mainly contains cars that move at 3 pixels/frame. |

$\mathbf{w}_i(\mathbf{p})$ in the different tiles with weighted averaging:

$$\mathbf{w}(\mathbf{p}) = \frac{\sum_i w_i \mathbf{w}_i(\mathbf{p})}{\sum_i w_i}, \tag{A1}$$

where $w_i$ is the weight of the $i$-th tile for pixel $\mathbf{p}$. Here, tiles are the crops, and we explain below how weights are determined.

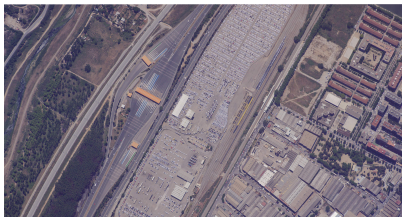
We compute the $H \times W$ weight map according to normalized distances $d_{x,y}$ of pixel $\mathbf{p} = (x, y)$ to the crop center:

$$d_{x,y} = \| \left(\frac{x}{H} - 0.5, \frac{y}{W} - 0.5\right) \|_2,$$
$$\text{and } w_{x,y} = \exp\left(-\frac{d_{x,y}^2}{2\sigma^2}\right), \tag{A2}$$

with $\sigma = 0.05$. Then, we build $K$ tensors $wm_k$ with dimensions $H \times W$, each of them representing the weight assigned to a pixel $\mathbf{p} = (x, y)$ in the $k$-th crop. Mathematically, the weighting map $wm_k$ of the $k$-th crop can be expressed as follows:

- $wm_k(x, y) = 0$, if $(x, y)$ does not belong to the

Table A2. Examples of sequences depicting highways

| sequence | description |
|---|---|
|  A001C002YE_Gare_50cm **size**: $2702 \times 1406$ | This sequence depicts a train station. It therefore contains carriages that can be assimilated to large objects. It also contains small moving vehicles (standard cars). |
|  A002C001E8_CentreCo_50cm **size**: $2761 \times 1444$ | This sequence presents a parking area of a shopping center as well as nearby highways. It comprises both small and larger displacements as some drivers are parking their cars, while other cars move on the road. |
|  A4C1R9_Peage_50cm **size**: $2708 \times 1445$ | This sequence shows a toll system. Similar to the previous sequence, it incorporates larger displacements (drivers that are leaving the tollbooth) and smaller displacements (drivers that arrive at the tollbooth and therefore need to decelerate). |
|  A004C002OI_Gare2_50cm **size**: $2708 \times 1391$ | This sequence depicts another train station. |

$k^{th}$ crop,

- $wm_k(x, y) = 1$, if $(x, y)$ is only contained in the $k^{th}$ crop,
- $wm_k(x, y) = \omega_{x,y}$, if $(x, y)$ is covered by several crops including the $k^{th}$ crop.

The final weighting map corresponds to the sum of the $K$ maps previously built. Fig.A9b highlights that using the weighting map leads to smoother edges between overlapping crops.

In Figure A10, we show that increasing the upscaling factor improves the optical flow results on satellite images. Indeed, as we increase the size of objects in the patches, more motion is predicted by RAFT. The flow is illustrated using smoothly blended results. Fig.A10a shows that the predicted flow between frames 32 and 33 of PlacaTarraco_Lion_5hz is better when upsampling each crop by a factor of 2. The result is even better when upsampling the crops by a factor of 4.

Cropping images has several drawbacks. First, estimating a per-crop flow significantly increases the inference time, because we feed the model one crop at a time. Also, the crop process can degrade the flow estimates, since it is impossible to predict

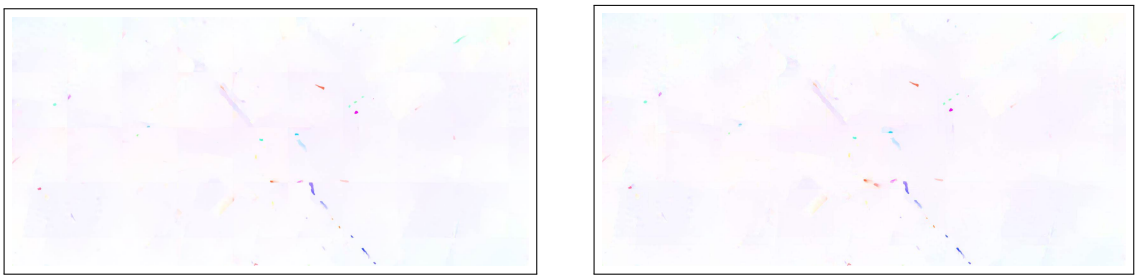
(a) Mosaic at time $t_0$

(b) Mosaic at time $t_0 + 1$

(c) Flow estimated on the downsampled mosaic

(d) Flow estimated on each individual image

Figure A8. Comparison of the flow of the mosaic and the mosaic of the flows



(a) Sharp boundaries between crops

(b) Smoothly blended boundaries between crops

Figure A9. Flow estimated between frames 2 and 3 of the cropped PlacaTarraco_Lion_5hz (upscaling factor = 4)

out-of-frame motion, i.e., objects that move out of the crop between two consecutive frames (see Figure A11). The number of such objects increases as we divide the image into smaller crops.

Both experiments confirm our intuition that the downsampling task performed by the feature encoder reduces the accuracy of RAFT. They also motivate our decision to develop a method that achieves a good tradeoff between memory and accuracy for satellite images, i.e., a method that is able to predict the flow of small vehicles while using the available GPU memory in an efficient way.

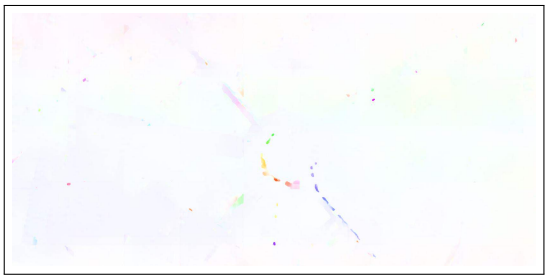
# 5. Visual results

## 5.1. Visual results on the training set

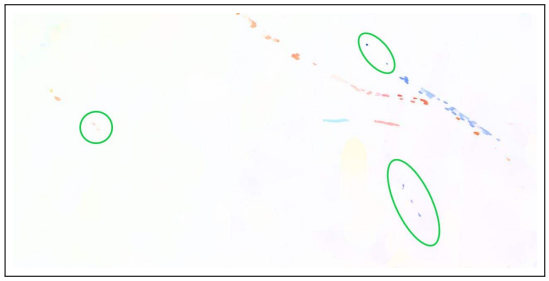
Training was performed on six aerial image sequences. We illustrate this training set by showing in Fig.A12 one sequence of this training set along with flows computed with our SMOFlow method and by the DIP and SCV methods. SCV fails to correctly estimate the optical flow over the small moving objects. DIP is prone to blur out the flows and then to blend close vehicles, as highlighted in the area cercled in red. Our SMOFlow method provides better defined and separated flows for vehicles close to each other, and recovers more small moving vehicles as in the area framed in blue.

(a) Scaling $\times 2$: predicted flow between frames 32 and 33 of PlacaTarraco_Lion_5hz

(b) Scaling $\times 2$: predicted flow between frames 4 and 5 of A6C15_Glories_50cm

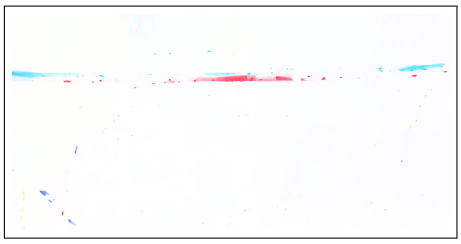


(c) Scaling $\times 4$: predicted flow between frames 32 and 33 of PlacaTarraco_Lion_5hz

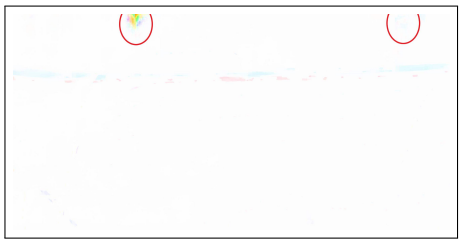


(d) Scaling $\times 4$: predicted flow between frames 4 and 5 of A6C15_Glories_50cm

Figure A10. Impact of the size of moving objects on the accuracy of RAFT



(a) **scaling** $\times 2$



(b) **scaling** $\times 4$

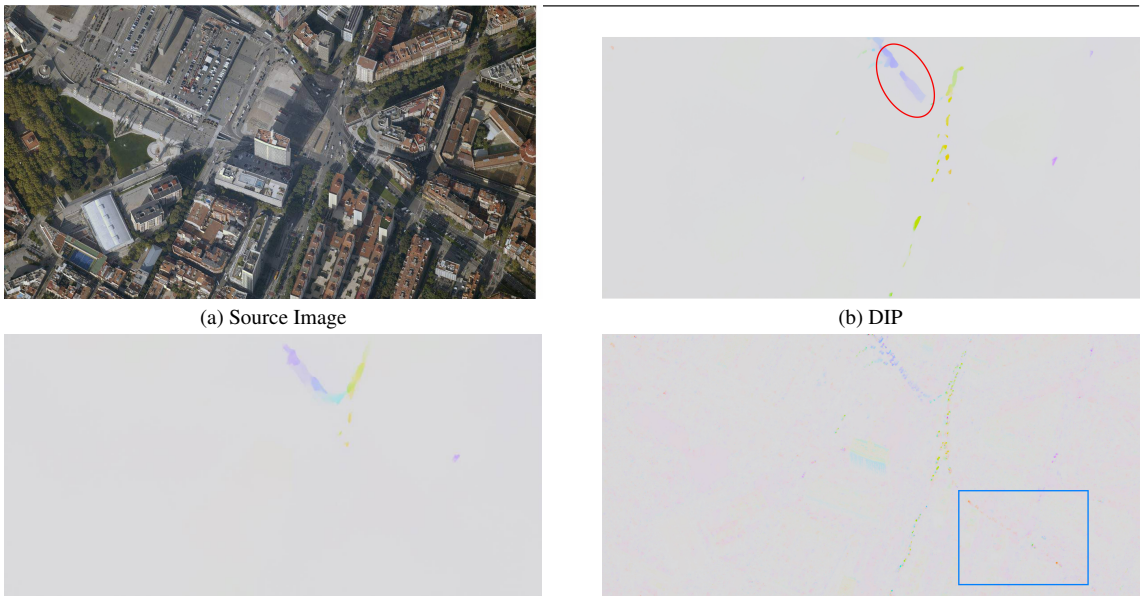
Figure A11. Predicted flow between frames 79 and 80 in A002C001E8_CentreCo_50cm

## 5.2. Visual results on the test set

In Figure A13, we provide a set of visual results depicting flow fields computed with our SMOFlow method on several image sequences of the test set. We can observe that SMOFlow is able to correctly estimate the flow of very small vehicles (best viewed by zooming in the pdf).

The first two examples at the top of Fig.A13 correspond to distant instants of the same sequence. There are vehicles moving on both sides of the tollgate (tollgate of orange color, located in the middle left of the image), as well as on the road to the right of the tollgate. Our SMOFlow method captures their motions in both directions (recognizable by the two different main colors). The two next examples depict cities. Again, SMOFlow correctly estimates the flow of cars moving on the straight avenues and around the roundabout. In these four examples, close vehicles are well-separated and clearly discernible.

In the last example at the bottom of the figure, we can observe a failure case that nevertheless remains moderate. On the access road to the tollgate (light gray tollgate in the center of the image), there are trucks whose computed flows are somewhat fragmented. This problem is due to the fact that these vehicles are elongated and uniform in color. Consequently, there is an overlap between the projections of these vehicles in the two successive images, with almost no temporal difference. The model tends to favor zero flow over this overlap area, if the smoothing effect cannot spread as far. One way of overcoming this problem would be to compute flows corresponding to several time intervals, and then, to combine them. A simpler and



(a) Source Image (b) DIP

(c) SCV (d) SMOFlow

Figure A12. From top to bottom and from left to right: One image of the training sequence Sants. Sample flows computed with DIP (b), SCV (c) and SMOFlow (d). Best viewed in color and by zooming in the pdf.

less time consuming alternative would be to compute the optical flow between more distant frames, for instance, between frames at time instants $t$ and $t + 3$.

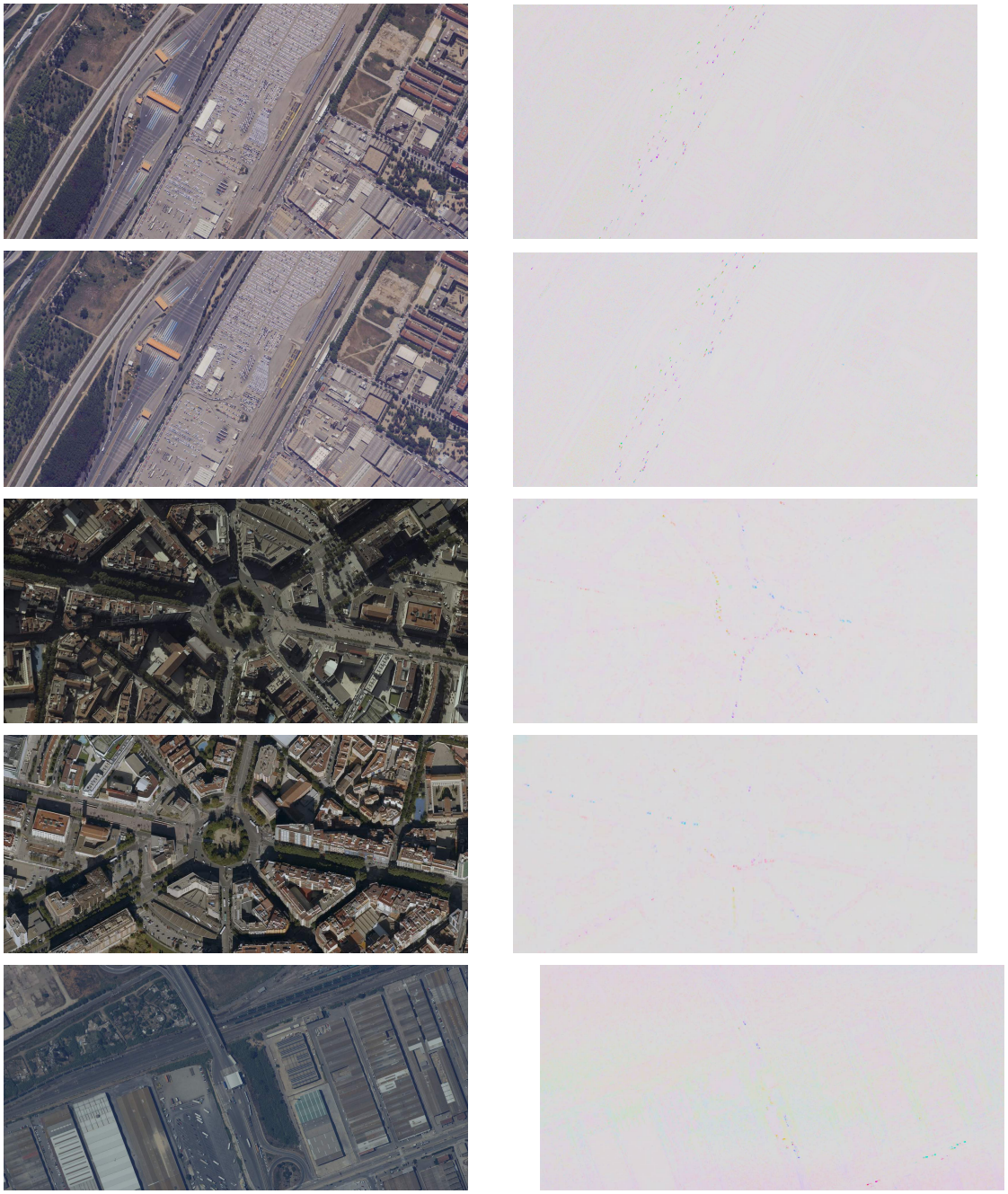


Figure A13. Sample results of our method SMOFlow on several aerial image sequences of the test set. Left: one image of the sequence. Right: corresponding flow computed with SMOFlow. Flows are represented using the usual HSV color code, with here a postprocessing for a better visibility as done in the main text. Best viewed in color and by zooming in the pdf.