Towards video inpainting: joint inpainting of motion and dynamic shapes with deep learning techniques

Pierrick Chatillon, Coloma Ballester, Gloria Haro, Pablo Arias, Lara Raad

october 2019 - july 2020

Contents

1 Introduction

This project falls into a more general goal of obtaining the semantic decomposition of a video into the different complete objects that were on the recorded 3D scene, and their trajectories. Some objects may appear as occluded or partially occluded in some frames because of differences in depth. In those cases we would like to disocclude (inpaint or complete) the dynamic objects by leveraging their visible parts at other time instants. For that purpose, the use of the optical flow has been shown to be very useful (e.g. [18, 9, 23, 5]). However, the use of the optical flow introduces an additional problem because the motion itself has to be completed in the occluded areas. The approach proposed in [23] is a video inpainting technique where the optical flow is completed in a first stage with a deep network and then it is used to guide the propagation of the known RGB values along the unknown region to fill in. It produces acceptable results but it fails in some cases, where the lack of sharp motion discontinuities in the inpainted flow can create visible artifacts. Furthermore, an unresolved ambiguity is created when two objects meet behind the occlusion.

In this project we planned to explore the impact of "dynamic shapes" to address these issues. A dynamic shape is a mask of an object in a scene, and evolves through time tracking the object. However, faced with the difficulty of reproducing the results of [23], we worked on simpler concepts like edges. The starting hypothesis was that both pieces of information, motion and shape or shape boundary, help each other in their completion. In particular, the shape can be used as a guide for the completion of the flow and thus be able to get better motion discontinuities aligned with the shape boundaries. This project follows the previous works of the team on dynamic shape completion [14] and motion inpainting [15] by treating both problems at the same time and mutually reinforcing in a joint formulation. Furthermore, we added the use of temporal consistency in network trainings, inspired by the methods proposed in [16, 20]. In order to decompose the video in different objects we use state-of-the-art video segmentation methods (e.g. [1, 19, 10]) or even ground truth segmentations of the YouTube-VOS dataset [22].

We did not reach the second stage planned: the use of inpainted optical flow to propagate the color information, which could be performed on its own with off-the-shelf propagation techniques: by solving a propagation/diffusion PDE [18], or by integrating optical flow trajectories from the interior of the inpainting domain to its boundary. This is the approach followed by [23].

This report is organized in two main parts, following the two big movements of this investigation year:

- A first part which rose after failing to reproduce [23]: essentially trying to achieve good inpainting results, with our own networks, losses and adversarial approaches. This made us reflect on architecture issues, warping in neural networks, use of edges, efficiency of several losses.

- A second part oriented towards a more traditional refinement procedure respecting spatio-temporal consistencies enhanced with deep learning techniques.

2 Around deep flow-guided video inpainting and experiments

The article [23] proposes to inpaint the optical flow of a sequence by feeding the concatenated masked flow of the sequence, jointly with the inpainting holes, to a cascade of huge residual networks [4], with an up-sampling after each network in a coarse-to-fine approach. Once the flow has been inpainted, they use a classical image inpainting method, and spread color information along flow lines.

Using the code provided by the authors of [23], we only tried to replicate the first inpainting step (only one ResNet [4], not training the two others for refinement purposes). However, training from scratch could not converge, fine tuning gave some results, but experimenting with such an architecture proved to be heavy and difficult (no change in architecture was possible since we needed the pre-trained weights).

2.1 Implementation details

2.1.1 Architectures

We mainly used two kind of networks for our experiments: ResNet [4] and U-Net [17]. We couldn't reproduce the results obtained by the ResNet architecture used in [23], so we tried our own take.

2.1.2 Dataset

We use videos from the Youtube Video Object Segmentation dataset [22] for training, and sometimes evaluate methods on the DAVIS dataset [2]. The flow ground truth is computed with FlowNet2 [6].

To prepare the input volume, we tried to apply the frame-wise smooth filling proposed in [23], which is indeed improving performance (Fig 9).

Some of our experiments needed the addition of edge information (discussed in Section 2.6).

2.1.3 Data bias

In order to create inpainting holes in interesting regions in each flow image of the dataset, the dataset is biased in the following way: for each sequence, we compute the Jacobian norm of the first flow frame. We sample the inpainting hole in a region whose center is picked according to an exponential law: $center = c_i, i \sim Exp(\beta = \frac{N_{samples}}{10})$ where c_i are the positions in the image sorted from biggest to lowest gradient strength.

2.1.4 Input data

We essentially dealt with two setups for the input data:

1. 6 consecutive frames, with a single random mask (as explained in 3.1.2) applied only on the two middle frames
2. 6 consecutive frames, with a random spacial mask applied to all of them, the same mask for all frames, created as explained in 3.1.2.
3. 'sandwich' setup: 6 consecutive frames, all of them masked, except the first and last one.

2.2 Losses

2.2.1 $\mathcal{L}^1$ loss

As usual, given two optical flows $p_1, p_2 : \Omega \subset R^2 \to R^2$, the $\mathcal{L}^1$ error between them is defined as:

$$\mathcal{L}^1(p_1, p_2) = \|p_1 - p_2\|_1$$
$$\mathcal{L}^2(p_1, p_2) = \|p_1 - p_2\|_2$$

After testing, we concluded that $\mathcal{L}^1$ error was a better supervision than $\mathcal{L}^2$.

2.2.2 Forward-Backward Check

The forward backward check consists in following each pixel from one frame to the next according to the flow and then back again according to the backward flow of the pixel reached. We then compute the distance ($\mathcal{L}^1$ or $\mathcal{L}^2$) between the original pixel, and the one reached through this process.

If f^t is the forward flow at frame t and b^{t+1} the backward flow at frame $t+1$,

$$\mathcal{L}_{fb}\left(f^t, b^{t+1}\right) = \int_{\boldsymbol{x}} \left\| f^t(\boldsymbol{x}) + b^{t+1}(\boldsymbol{x} + f^t(\boldsymbol{x}) \right\|_2$$

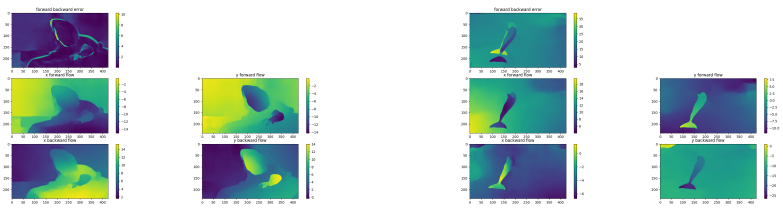


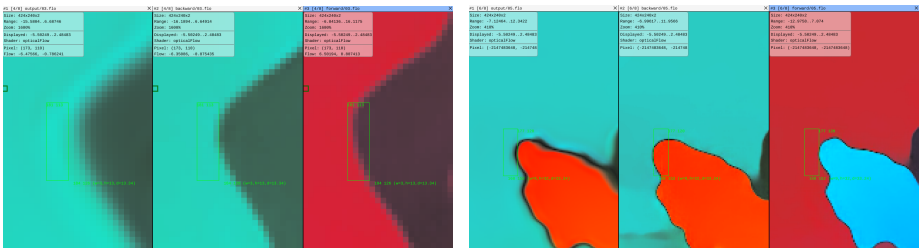
Figure 1: sanity check of the forward backward consistency loss (top is forward/backward error, the for images below are the x and y coordinates of the two consecutive frames used

Figure 1 shows this pixel-wise forward/backward error in a good case (error close to zero except in occlusion zones) at the left, and in a bad case to the right: since the forward and backward flows are independently computed, background regions may not have a consistent optical flow, hence the big error.

2.2.3 Forward backward consistency supervision: a sanity check

It appeared like a good starting point to ask for forward backward consistency for flow inpainting.

We still conducted a sanity check: to use a network to predict the backward flow of an image while using the forward flow of the previous one frame. The network is trained to minimize the forward/backward errors.



(a) Good example (b) Bad example

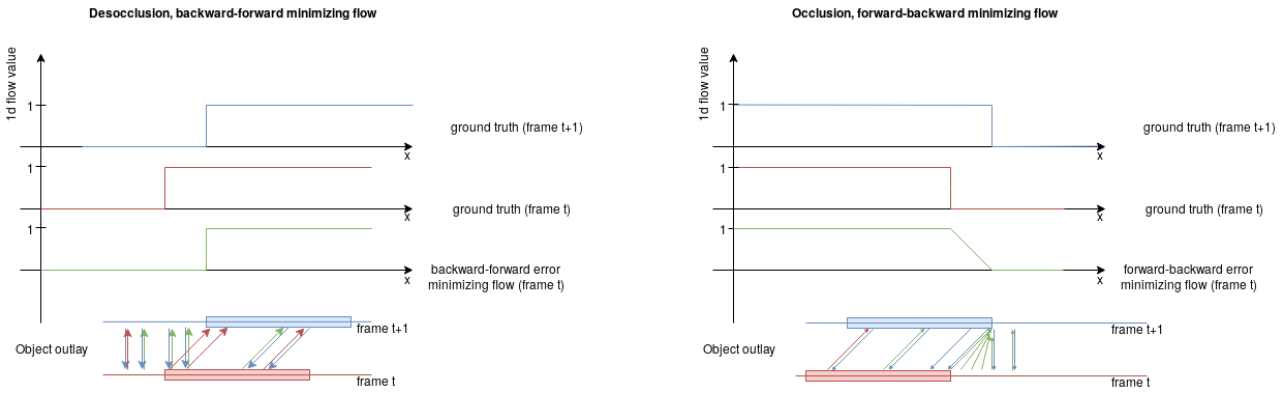
Figure 2: Sanity check examples for a *good* example (a) and a *bad* one; left: network output, middle: target backward ground truth flow, right: input forward ground truth flow

We can see a working example on the left of Figure 2a, where, although blurry, the edge of the object has been moved according to the optical flow. On the right however (Fig 2b), were the values of the flow are much bigger, the network fails to correctly warp the object, and instead just copies the silhouette. This highlights both a warping problem with the network, and an issue with the forward/backward loss itself.

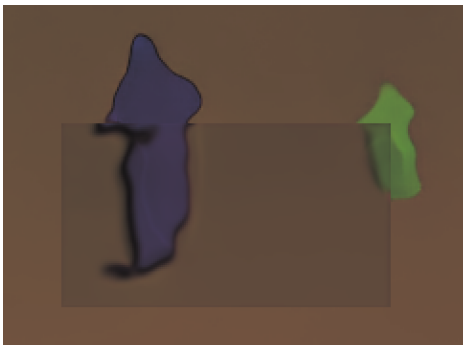
2.2.4 Theoretical justification

Let us now detail how asking to minimize forward/backward and backward/forward errors is not giving an optimal solution in occlusion areas:

- in case of disocclusion from frame t to t+1, minimizing the backward-forward error with respect to the flow at frame t leads to the copying of the shape of the object at frame t+1.
- Similarly, in case of occlusion, the optimal minimizer of the forward-backward error will linearly interpolate the values of the flows (of the object and of the background) in the occlusion area



(a) Simple occlusion and disocclusion cases; red and blue being the flow ground truth at frames t and $t+1$, green being the forward flow at frame t obtained by minimizing $\mathcal{L}_{fb}$ between itself and the backward flow at frame $t+1$.



(b) example of inpainted flow using these errors as supervision for a network

Figure 3: Occlusions and forward-backward

in the right side of Fig 3a, we study the case of occlusion and recovery of forward flow by minimizing the forward/backward error. In other words, we have an object moving at speed 1 to the right with edge at position 0 at frame t (flow plotted in red) and in the next frame $(t+1)$, we have its backward flow (the negative of the forward flow at frame $t+1$ plotted in blue) with edge at position 1. We want to find the forward flow f at frame t that minimizes the forward/backward error with frame $t+1$:

$$f = \underset{g}{\operatorname{argmin}} \|g + (-\mathbb{1}_{g+id\leq 1})\|^2$$

with $id : x \to x$; $x \to -\mathbb{1}_{x\leq 1}$ being the backward flow at frame $t+1$; and $-\mathbb{1}_{g(x)+x\leq 1}$ being the backward flow taken at the arrival point when following the forward flow from point x on the previous frame.

$$f(x) = \underset{y}{\operatorname{argmin}} \mid y - \mathbb{1}_{y+x \leq 1} \mid$$

One can verify that f takes these values:

x	$-\infty$	0	1	$+\infty$
$f(x)$	1	1-x	0	

Such an optimization leads to a smooth interpolation between flows of occluding object and background. In Fig 3b is an example of result obtained with the sanity check; the motorbike is moving towards the left, so we have an occlusion on the left and a disocclusion on the right of the biker (motion depicted in purple). We can see a black area on its left, which is the result of the linear interpolation between the background flow and the motorcycle flow. This check was made after numerous experiments were conducted with the forward/backward error, so this loss will still appear in some results.

2.2.5 Lateral dependency loss

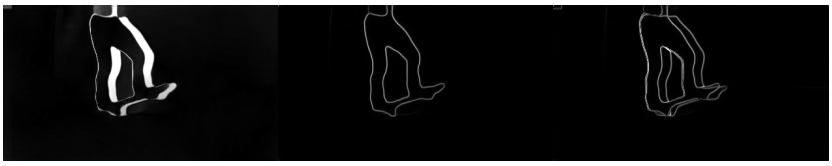
The lateral dependency loss is taken from [25], and was tried out in some experiments:

$$\mathcal{L}_{ld} = \frac{1}{n} \sum_{i,j} \left| EPE\left(p_2(i,j), p_2(i-1,j)\right) - EPE\left(p_1(i,j), p_1(i-1,j)\right) \right| + \left| EPE\left(p_2(i,j), p_2(i,j-1)\right) - EPE\left(p_1(i,j), p_1(i,j-1)\right) \right|$$

where $EPE\left(p_1(\boldsymbol{x}), p_2(\boldsymbol{x})\right) = \|p_1(\boldsymbol{x}) - p_2(\boldsymbol{x})\|_2$ that is, the Euclidean distance between two vectors $p_1, p_2 \in R^2$.

2.2.6 Flow-based loss intent

We studied the spacial derivative of the convective derivative of the flow e.g. $\|f^{t+1}(\cdot + f^t) - f^t\|$.
It turns out it gives the boundaries of the occluded regions of frame $t+1$, just like taking the norm of the gradient of the forward-backward error would.
This is caused by the forward-backward error is based on the term $\|b^{t+1}(\cdot + f^t) + f^t\|$; and$f^{t+1}$ and b^{t+1} carry the same 'shape' information since they both describe the movement of pixels of the same frame $t+1$. As we can see in Fig **??**, this error computes the edges of moving objects, and of the occlusion area. This error was left behind for the rest of the work.



(a) forward backward (b) flow gradient norm (c) spacio-temporal error

Figure 4: Spacio-temporal error

2.2.7 PatchGan

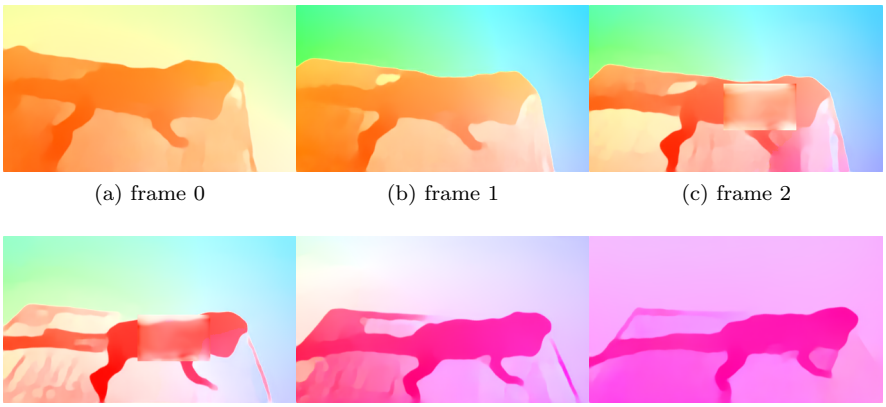
In addition to usual losses, we tried to add an adversarial loss in the spirit of [7]. In other words, we submit the inpainting network to the critic of a discriminator that looks at overlapping patches in flow images, inpainted or not, and is optimized to distinguish between them. The inpainting network is trying to fool the discriminator. In all our experiments, the Discriminator performs so well that the inpainting network cannot compete as well as a generator would. Indeed, a generator takes as input random noise, whereas our network is deterministic since its only input is the masked flow, paired with the mask itself.

In the end, the loss to be minimized reads:

$$\mathcal{L} = \lambda_{\mathcal{L}^1}\mathcal{L}^1 + \lambda_{fb}\mathcal{L}_{fb} + \lambda_{gan}\mathcal{L}_{PatchGAN}$$

We conducted tests without the GAN part: $\lambda_{\mathcal{L}^1} = \lambda_{fb} = 1, \lambda_{gan} = 0$.
only forward flows are shown in Figure 5.
the network predicts f_2, f_3, b_2, b_3 The forward backward checks are made between f_1^{gt} and b_2, f_2 and b_3, and finally between f_3 and b_4^{gt}.

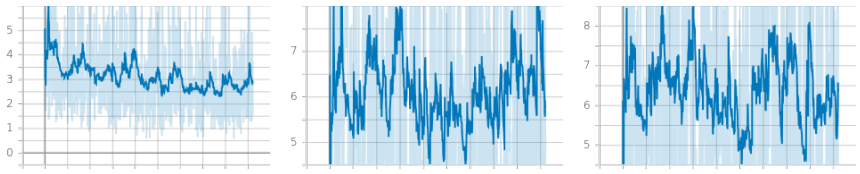
The losses are computed between whole images (and not only on the masked areas), that is why the loss curves do not go down in Figure 6 (due to variation in the ground truth, independent to the inpainting). However, the loss is still back-propagated correctly to the network.



(a) frame 0 (b) frame 1 (c) frame 2

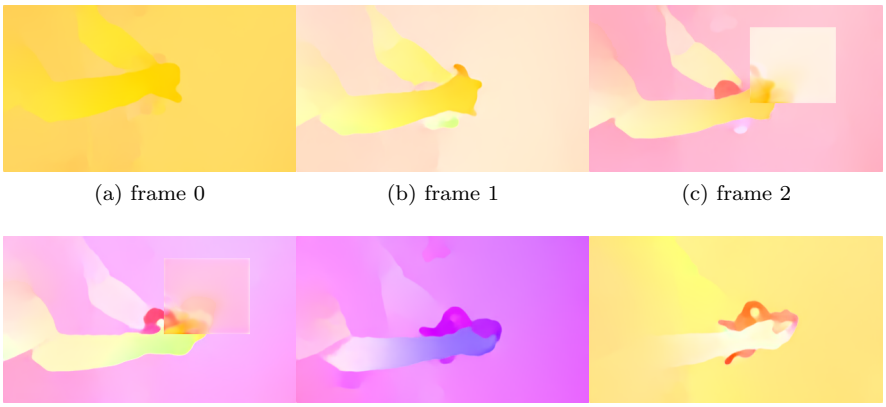
(d) frame 3 (e) frame 4 (f) frame 5

Figure 5: PatchGAN set up ($\lambda_{\mathcal{L}^1} = 1, \lambda_{fb} = 1, \lambda_{gan} = 0$). Inpainting is made on frames 2 and 3



(a) L1 (b) forward backward f_2 / b_3 (c) forward backward f_1^{gt} / b_2, and f_3 / b_4^{gt}

Figure 6: loss curves for Figure 5



(a) frame 0 (b) frame 1 (c) frame 2

(d) frame 3 (e) frame 4 (f) frame 5

Figure 7: PatchGAN set up ($\lambda_{\mathcal{L}^1} = 1, \lambda_{fb} = 1, \lambda_{gan} = .1$). Inpainting is made on frames 2 and 3

2.3 Results

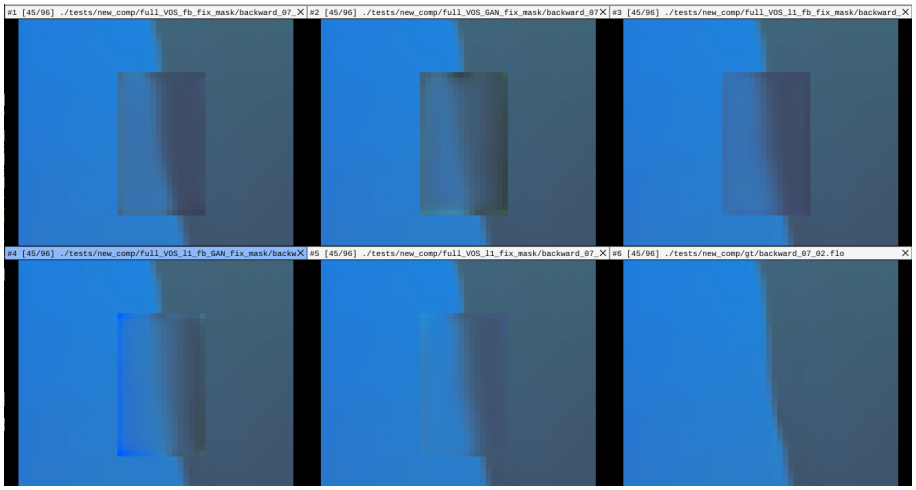


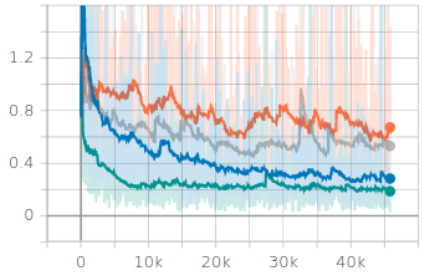
Figure 8: Comparison of performances for the same inpainting task. Only the two middle frames are inpainted. top row, left to right: $\lambda_{\mathcal{L}^1} = 0, \lambda_{fb} = 1, \lambda_{gan} = 0$; $\lambda_{\mathcal{L}^1} = 0, \lambda_{fb} = 0, \lambda_{gan} = .1$; $\lambda_{\mathcal{L}^1} = 1, \lambda_{fb} = 1, \lambda_{gan} = 0$; bottom row: $\lambda_{\mathcal{L}^1} = 1, \lambda_{fb} = 1, \lambda_{gan} = .1$; $\lambda_{\mathcal{L}^1} = 1, \lambda_{fb} = 0, \lambda_{gan} = 0$; ground truth

2.4 Architecture discussion

One problem that rose in the inpainting task was that sometimes the network would place edges at the very position from another frame (where the edge is not masked).

After reflexion, we realized that U-Net architecture was a bad choice since it is designed for segmentation tasks. The copying that happens is likely due to the architecture itself (which directly sends low level information to the end of the network).

This is confirmed by the comparison of $\mathcal{L}^1$ loss curves (Fig 9).



(a) L1 loss

Figure 9: Different displays of L1 loss evolution during training for different settings in $\mathcal{L}^1$ supervised flow inpainting for 6 frames and a 6 frame rectangular hole. Orange curve: U-Net and 0 flow values in the hole; grey: U-Net and smooth initial interpolation in the hole Blue: ResNet and 0 flow values in the hole; green: ResNet and smooth initial interpolation in the hole

2.5 Warping

To face the struggle for warping of the networks, we investigated ways to help the network on this specific task:

2.5.1 Discussion

- The article **CoordConv** [12] proposes to add 2 channels to the input volume, containing the x and y coordinates of each pixel. The article claims to solve the following problem: draw an image with a lone point for given coordinates. We first thought this could be applied to warping, since it consists in filling information at a specific location, given the starting position and the flow offset. After reflexion, we believe their method doesn't generalize to the case of warping since they propose is very problem-oriented solution:
  The input is a 4 channel volume taking the spacial size of the desired output: the first two channels are filled respectively with x and y (the query coordinates of the point to be filled on the image), the two others are the coordinates of the locations of the feature map. However, their architecture consists in a cascade of 1*1 convolutions; which means the network basically learns to apply the simple pixel-wise logic gate $(x_{target} == x_{pos}) and (y_{target} == y_{pos})$. Such a Network could learn to warp 1 pixel, but it doesn't generalize to warping hole images since we would basically need to add 2 channels filled with the coordinate of every pixel we wish to warp.

- **Spatial Transformer Networks** The article [8] can be tailored for our method, using simple transformations like translations to help warping. The idea is to take the features after the first convolutions, predict an offset for each channel, and used these warped features instead for the rest of the network computations.

2.5.2 Warping UNnet

Following the application of Spatial Transformer Network [8]:
Features f are extracted from the input, from the input volume, we predict translation offsets for each channel of f with a small network (strided convolution and fully connected). The concatenation of f and the translated version of f are fed to the rest of the network (see Fig 10).

This combined with the same U-Net used previously produced poor results as shown in Fig 11

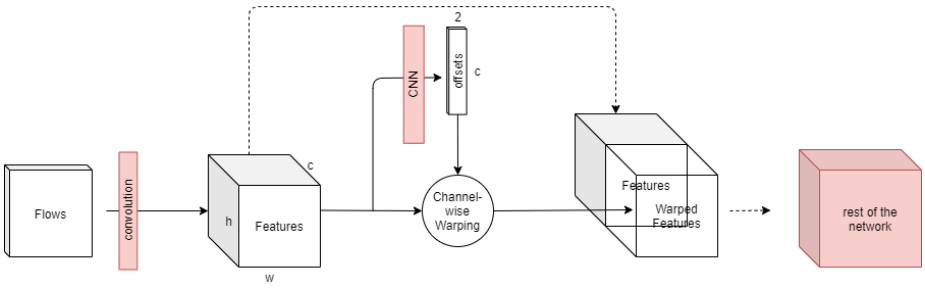


Figure 10: warping module

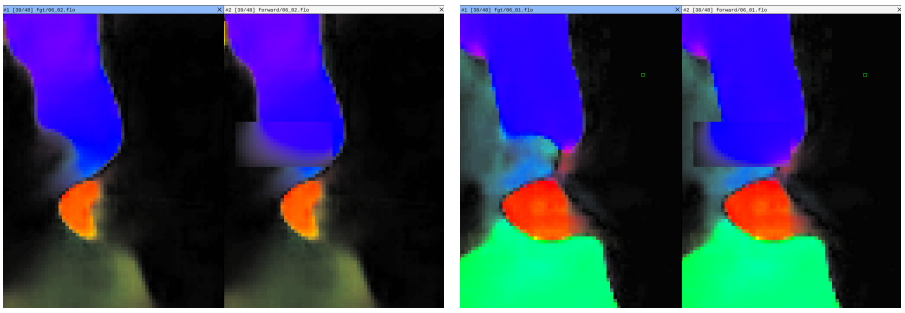


Figure 11: U-Net + Warping module (left: ground truth, right: inpainting result)

2.6 Flow and edges

Since optical flow data is very smooth with sudden discontinuities, one can investigate how to use this edge information to obtain better results.
We thought of two ways to incorporate edges to our methods:

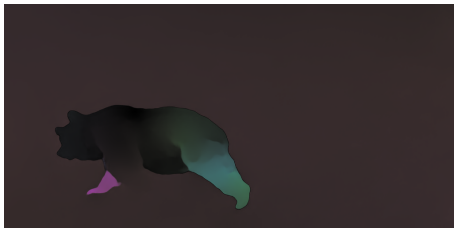
- We can ask the network to predict both flow and edges, in a supervised or unsupervised way.
- We can try to inpaint flow edges and feed them to a network previously trained to fill flow information given edges. This idea was inspired by the article EdgeConnect [13] which does image inpainting by completing the edges first.

But first, which edges are we talking about?

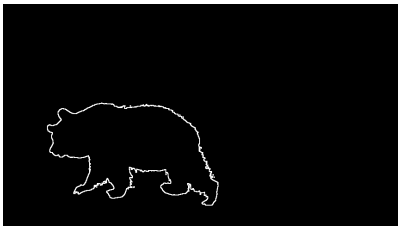
2.6.1 Edge computation

The first edges we used were computed using the gradient of the object segmentation images that came with the datasets, but they sometimes contain non moving, some moving objects are not segmented, and the boundaries of the flow we initially computed do not always match well with the segmentation result.

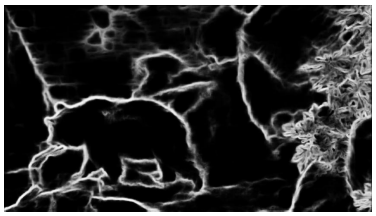
We considered computing edges directly on the image level, using an 'off-the-shelf' segmentation algorithm ([21]), but this gives poor results since discontinuous background information gets detected.
To address the lack of consistency between our computed optical flow and edges we finally decided to compute them on the image flow directly. We can achieve that with a simple Canny algorithm.



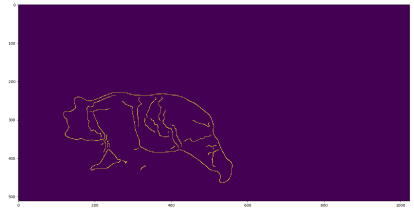
(a) flow



(b) edges of object segmentation ground truth



(c) HED applied at image level

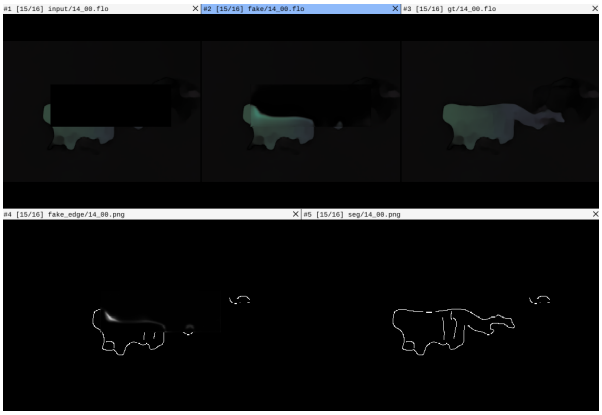


(d) Canny edges on flow image

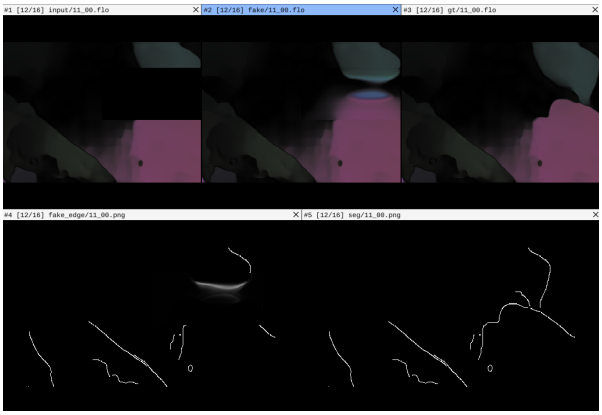
Figure 12: Edges computation

2.6.2 Supervised edges

In an adversarial way, we train a network to inpaint both flow and edges conditionally to flow and edges outside the inpainting domain. In other words, in addition to an $\mathcal{L}^1$ loss for the flow, we use a discriminator to look at the combination of flow with edges, to help the network improve its results, and learn how flow discontinuities should be located at the edges. We provide the masked edges to both the generator and discriminator for them to learn the how it affects the inside of it.
This method yields much sharper edges (Fig 13).



(a)



(b)

Figure 13: two examples of joint inpainting of flow and edges: first row: masked flow, inpainted flow, ground truth; second row: inpainted edges, ground truth edges

2.6.3 Unsupervised edges

The idea proposed in this section were not put into practice as we decided to stick with the supervised learning of edges. We thought about forcing the correlation

between the edge map and the gradient norm (hence the unsupervision):
if $\boldsymbol{e}, \boldsymbol{n}$ are the edge map and the gradient norm map, we want to have a loss $\mathcal{L}_{edge}(\boldsymbol{e}, \boldsymbol{n})$ that satisfies:

- $\forall \boldsymbol{n},\ \arg\min_{\boldsymbol{e}} \mathcal{L}_{edge}(\boldsymbol{e}, \boldsymbol{n}) \in \{0, 1\}$
- $\mathcal{L}_{edge}(0, 0) = 0$
- $\lim_{\boldsymbol{n} \to \infty} \mathcal{L}_{edge}(1, \boldsymbol{n}) = 0$

$\mathcal{L}_{edge}(\boldsymbol{e}, \boldsymbol{n}) = (1 - \boldsymbol{e}) \times (1 - e^{-\boldsymbol{n}/N}) + \boldsymbol{e} \times e^{-\boldsymbol{n}/N}$ With N being a fixed threshold. The loss plotted as a function of $x = \boldsymbol{e}$ and $y = \boldsymbol{n}$ in figure 14.

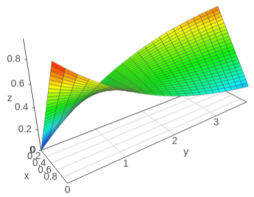


Figure 14: $\mathcal{L}_{edge}(\boldsymbol{e}, \boldsymbol{n})$

2.6.4 Edge inpainting

We trained a WGAN-GP ([3]), to fill the Canny edges inside the mask, conditionally to the edges outside the mask .
This provided good results edge-wise (see the continuity of the inpainted edge in Fig 15, left). We did not however continue in that direction because this method is applied to single frame flow inpainting (one lead is to further study sequence generation).This method expectedly tends to hallucinate shapes as shown in the left of Fig 15.

Figure 15: Two examples of edge inpainting with conditioned GAN (on each image, left: ground truth, right: inpainted edges plotted inside the mask

3 Flow-based iterative refinement for inpainting

3.1 Dataset

We changed which data is loaded, because of bad flow consistency in previous data (see Fig. 2 on the right).
Before, we only used 1 frame out of 5 in the video because only 1 out of 5 was annotated for the segmentation (which we used at some point). We computed again all the flows with FlowNet2 [6], and changed the way they were stored (.flo files are huge compared to images, so we saved flows as images, while saving the range of the flows values.

3.2 Idea

3.2.1 Formulation

The idea is to maintain the frames separated instead of mixing them, as in the previous approaches. We want to embed local flow information into a feature vector for every frame, and iteratively refine them in a way that respects flow consistency. For each frame and iterative step we have the following information at every pixel $(\boldsymbol{x}, t)$:

- feature vectors $F_t^f(\boldsymbol{x})$ encoding both forward and backward flows that stream from frame f at step t. They encode the forward and backward optical flows, but in a local neighborhood of $\boldsymbol{x}$.
- forward and backward flows $v_t^{f,f+1}(\boldsymbol{x}), v_t^{f,f-1}(\boldsymbol{x})$
- confidence mask on the flows $\kappa_t^f(\boldsymbol{x})$

The intuition is that $F_t^f(\boldsymbol{x})$ are a rich representation of the optical flows so that the network has enough capacity. The optical flows together with their confidences are will be used to determine the connectivity in the network, which is unknown and to control the propagation of information from the boundary inwards.

These variables should be related: F_t should give, after decoding, the optical flows used for confidence spreading.

3.2.2 Update

We need three networks, one encoder, one decoder, and finally an update network, using partial convolutions [11] (chosen upon gated convolutions [24]), where the input features are multiplied by the confidence κ, and the output is then normalized. These three networks are denoted by $Flow2Feature$, $Feature2Flow$ and $Update$

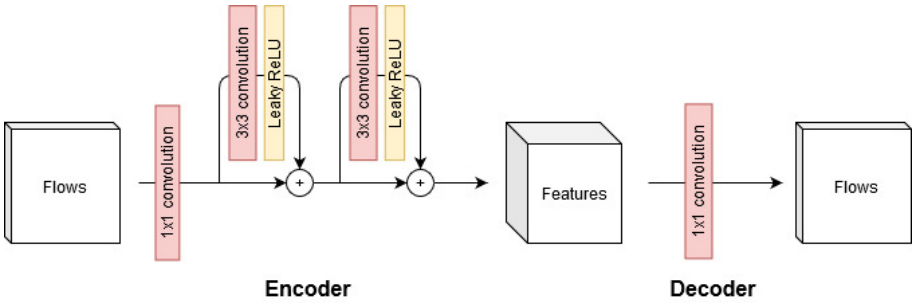
Partial convolutions allow to do a spatio-temporal propagation, instead of having to alternate spatial and temporal confidence spreading.

3.3 Pseudo algorithm

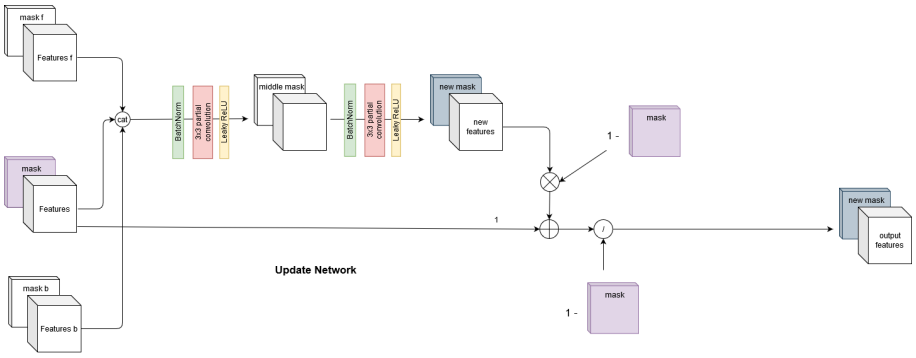
Algorithm 1: Iterative inpainting

Initialize Flow2Feature, Feature2Flow and Update networks
while *not converged* **do**
  get data batch
  $F_f = Flow2Feature((forward, backward)_f)$
  `// encode the forward and backward into a single representation flow for each frame f`
  $loss_1 = \mathcal{L}^1${flows,Feature2Flow(Flow2Feature(flows))}
  `// reconstruction error for confident pixels in the input`
  for *step in* $[1, n_{steps}]$ **do**
    for *f in* $[1, n_{frames}]$ `// frames in the video` **do**
      compute flows from features
      use this flow to warp the features and confidence masks of neighboring to current frame
      $new\_F, new\_mask = Update(warped\_F, warped\_mask)$
    end
    Compute and back-propagate the loss
  end
end

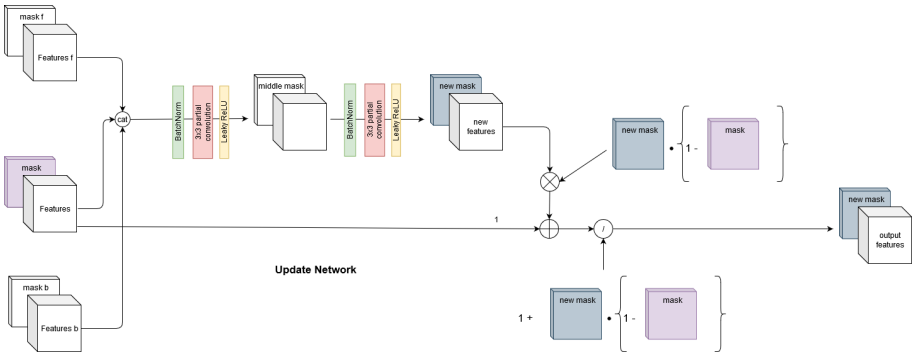
3.3.1 Architectures



(a) Encoder Decoder architectures



(b) Update network architecture 1



(c) Update network architecture 2

Figure 16: Architectures

3.3.2 Encoding and decoding Networks

We want to embed the forward and backward flow information at each pixel of each frame into a single representation to be processed for inpainting purposes. We need to be able to reconstruct said flow information: this is why we choose a simple $1x1$ convolution layer to pass from one representation to the other. To allow encoding of information from the spacial surroundings while maintaining an structure that can easily reconstruct the flows, we add residual layers (see Figure 16 (a)).

3.3.3 Updating Network

Since we want to iterate the updating network, it makes sense to see it as a residual network, that enhances the representation at each iteration step. It takes as input the current features and masks, plus the features and masks from the previous and next frames, that have been warped onto the current ones following the flows obtained by decoding the current features.
A set of two partial convolutions [11] provides us with new features, and an updated confidence mask 'new mask' in 16 (b).
We normalize the features to avoid gradient vanishing when iterating the same network over and over again.
The term residual comes from the fact that the output feature map is a weighted sum of the input features of the current frames, and the new features:

$$F_{out} = \frac{F_{in} + F_{new} \cdot mask_{new} \cdot (1 - mask_{in})}{1 + mask_{new} \cdot (1 - mask_{in})}$$

We don't want pixels with input confidence 1 to be updated, hence the $(1 - mask_{in})$ term, furthermore, we don't want to update the pixels that are not yet confident, hence the $mask_{new}$ term.
In our experiments, this is the best we found to sum two features maps with different confidence masks.
This formula implies that we will use features from inside the initial inpainting hole, although they will be updated a lot. This means that we need a relevant initialization of the features inside the inpainting zone. To do so, we first smoothly interpolate the flow inside the inpainting region, then pass it through the encoder to get our initial feature volume.
The mask is initialized as 1 outside the inpainting zone, and 0 inside, although some experiments on what the initial mask should be have been conducted.

3.3.4 Partial convolution mask update

Citing the paper [11], partial convolution is described as:
"
We refer to our partial convolution operation and mask update function jointly as the Partial Convolutional Layer. Let W be the convolution filter weights for the convolution filter and b its the corresponding bias. X are the feature

values (pixels values) for the current convolution (sliding) window and $\mathbf{M}$ is the corresponding binary mask. The partial convolution at every location, similarly defined in [7] , is expressed as:

$$x' = \begin{cases} \mathbf{W}^T(\mathbf{X} \odot \mathbf{M})\frac{\text{sum}(\mathbf{1})}{\text{sum}(\mathbf{M})} + b, & \text{if sum}(\mathbf{M}) > 0 \\ 0, & \\ \text{otherwise} & \end{cases}$$

where $\odot$ denotes element-wise multiplication, and 1 has same shape as M but with all elements being 1.
As can be seen, output values depend only on the unmasked inputs. The scaling factor sum(1)/sum(M) applies appropriate scaling to adjust for the varying amount of valid (unmasked) inputs.
After each partial convolution operation, we then update our mask as follows: if the convolution was able to condition its output on at least one valid input value, then we mark that location to be valid. This is expressed as:

$$m' = \begin{cases} 1, & \text{if sum}(\mathbf{M}) > 0 \\ 0, & \text{otherwise} \end{cases}$$

”
The paper update rule is

$$m' = min(sum(\mathbf{M}), 1).$$

The coded update rule on their Github however was found to be

$$m' = mean(\mathbf{M})).$$

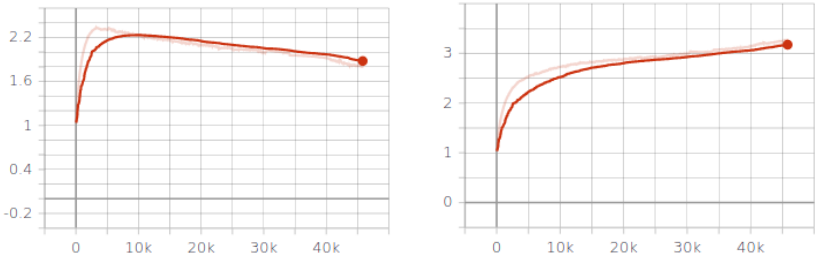
We propose new learnable update rules to obtain the new confidence mask, since it is central in our iterative scheme and our case differs a bit from theirs since the first partial convolution layer of the update network takes 3 masks as input instead of 1 (16 (b)).

Generalized mean update

The first idea is using the weighted generalized power mean with trainable parameters w and p:

$$m' = \left(\frac{sum(w \cdot \mathbf{M}^p)}{sum(w)}\right)^{\frac{1}{p}} \tag{1}$$

Here, The values of $\mathbf{M}$ differ along the channel dimension.
This lets the network choose if the update rule should be closer to the mean ($p = 1$) or to the maximum ($p \to \infty$) (see Figure 17).



(a) evolution of the parameter p in the first partial convolution layer of the update network

(b) evolution of the parameter p in the second partial convolution layer of the update network

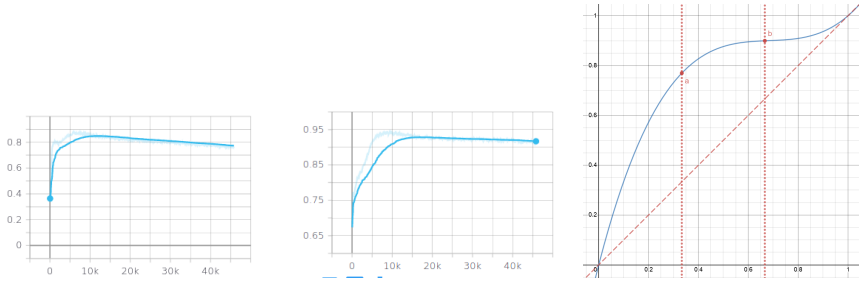
Figure 17: Evolution of parameters when partial convolution layers use the generalized mean update

Polynomial of mean update

Taking $m' = mean(\mathbf{M})$) might lead to a slow spreading of the confidence, we might want to increase the mean when its values are low, and maybe decrease it when its value gets too high. To address this issue, we propose to update the mean this way:

$$m' = \mathcal{P}_{a,b}(\frac{sum(w \cdot \mathbf{M})}{sum(w)}),$$
$$\text{where } \mathcal{P}_{a,b} = \left(\frac{27a}{2} - \frac{27b}{2} + \frac{9}{2}\right) \cdot x^3 + \left(-\frac{45a}{2} + 18b - \frac{9}{2}\right) \cdot x^2 + \left(9a - \frac{9b}{2} + 1\right) \cdot x \tag{2}$$

$\mathcal{P}_{a,b}$ is the Lagrange interpolator of the 4 points $(0,0), (\frac{1}{3}, a), (\frac{2}{3}, b), (1,1)$.
a, b and w are set as trainable parameters. cf Figure 18.



(a) evolution of the parameter a in the first partial convolution layer of the update network

(b) evolution of the parameter b in the first partial convolution layer of the update network

(c) Plot of $\mathcal{P}_{a,b}$ for the final values of $(a, b) = (0.77, 0.9)$

Figure 18: Evolution of parameters when partial convolution layers use the polynomial of mean update

3.4 Losses

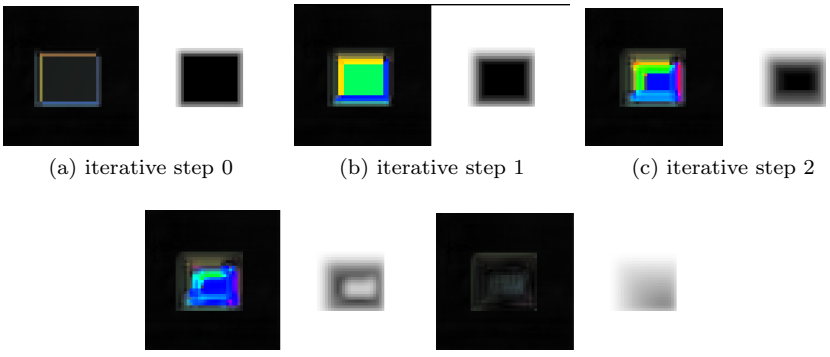
The two losses used in every experiment are $\mathcal{L}^1$ flow reconstruction inside and outside the original inpainting hole:
Let $\mathcal{H}$ be the inpainting mask such that $\mathcal{H} = 0$ inside the inpatinting zone; t be the current step in the iterative process; and $flow_t = Dec(F_t)$ and $bflow$ the flows and backward flows extracted from the features at step t:

- $\mathcal{L}_{decoder} = \|(1 - \mathcal{H}) \cdot (flow_{gt} - flow_0))\|_1$
- $\mathcal{L}_1^t = \|\mathcal{H} \cdot (flow_{gt} - flow_t))\|_1$
- $\mathcal{L}_{TV}^t = \|(flow_{gt} - flow_t)\|_{TV}$
- $\mathcal{L}_{minfb/bf}^t = mean\Big(\min(\Big\|flow_t^f - bflow_t^{f+1}(flow_t^f)\Big\|_2(x), \Big\|bflow_t^f - flow_t^{f+1}(bflow_t^f)\Big\|_2)(x)\Big)$
  ,where f is a video frame

3.4.1 $\mathcal{L}^1$ reconstruction

A big choice was whether or not to compute the inpainting loss weighted by the current confidence mask or not:

- if so, the network learns to give 0 confidence to the inpainting zone, unless given an additional objective on confidence itself (see Fig 20).
- if not, the network is trained and modified according to unconfident pixels, yielding unexpected behaviors discussed in the next section (Figure 19).



(a) iterative step 0 (b) iterative step 1 (c) iterative step 2

(d) iterative step 3 (e) iterative step 20

Figure 19: Failure of asking $\mathcal{L}^1$ reconstruction inside the mask at the final step.

3.4.2 Total variation loss

Flow images tend to be piece-wise smooth, so a total variation loss term allows to recover this property.

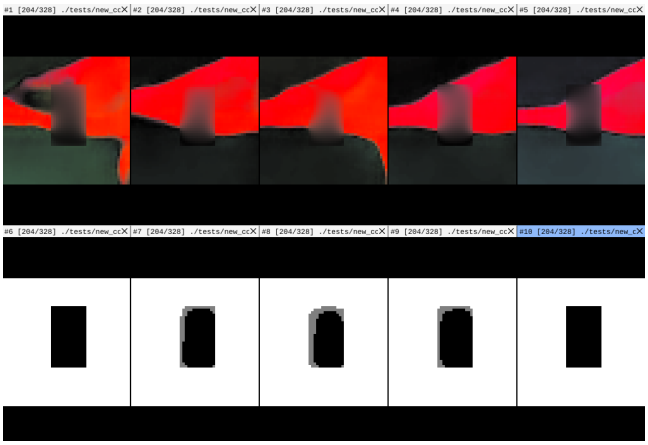


Figure 20: final step of iterative refinement using $\mathcal{L}^1$ weighted by confidence values, flows and masks.

3.4.3 Flow consistency loss

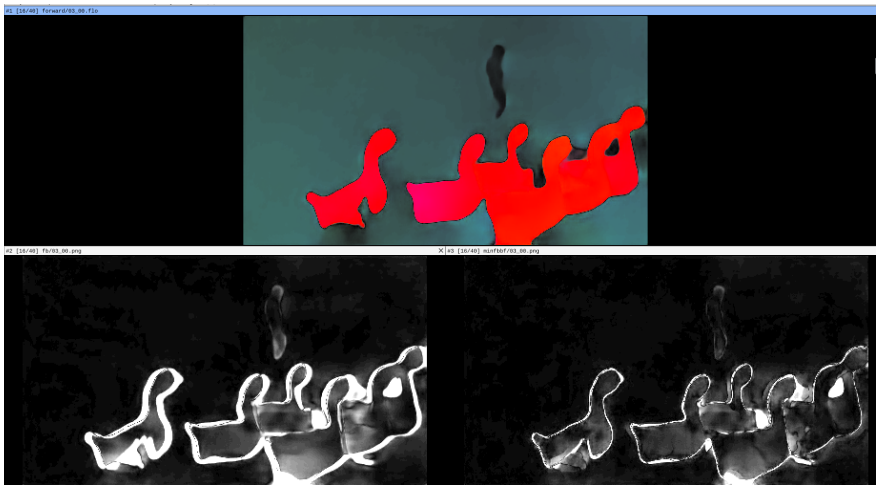


Figure 21: top: flow current frame, bottom left: point-wise forward/backward error, bottom right: point-wise minimum of forward/backward and backward/forward errors

We saw in Figure 3 that the forward-backward error is a bad supervision for flow reconstruction. To further improve results, we propose another flow-related supervision: $\mathcal{L}^t_{minfb/bf}$
It is obtained computing for each pixel the forward/backward and backward/forward errors with the neighboring frames, and taking the minimum.

Figure 21 shows that for a consistent flow sequence (taken from the dataset), this error is only high on very difficult border pixels, unlike the forward/backward error which is high on all the occlusion area.
Adding this error to the loss allows sharper and more consistent reconstruction.

In the end the loss reads:

$$\mathcal{L} = \mu_{decoder}.\mathcal{L}_{decoder} + \sum_{t=0}^{t_{max}} \mu_{inpaint}^{t}.\mathcal{L}_{inpaint}^{t}$$

with μ described in the following section, and $\mathcal{L}_{inpaint}^{t}$ being a combination of $\mathcal{L}_{1}^{t}$, $\mathcal{L}_{TV}^{t}$, and $\mathcal{L}_{minfb/bf}^{t}$

3.5 Training Scheme

Since we need faithful flow reconstruction to achieve meaningful forward backward error estimation, it is sound to train the encoder/decoder structure first, to then proceed to train the iterative inpainting network (cf Eq. 3 and 4, the beginning of the training is focused on the encoder/decoder).
Furthermore, computing the L^1 fidelity loss term (without confidence weighting) on the final output of many iterations of the same refinement network can lead to unexpected and unwanted results (Figure 19).

Since the network relies on confidence spreading through neighbors and along optical flow, it learns to hallucinate big optical flow values (b,c in Figure 19) to leverage confidence through optical flow propagation. Although this does not seem sane, in the end, the refinement takes place and provides acceptable results.
To tackle this issue, we first tried to manually weight the different losses by μ according to 3.

$$\begin{aligned} &\forall s \text{ training step}, \begin{cases} \mu_{decoder} = 1 \\ \mu^t_{inpaint} = c.(1 - e^{-\frac{s}{s_0}}).e^{-\frac{1}{2\sigma^2}(t+1-\frac{s}{s_0})^2} \end{cases} \\ &\text{where } c \text{ is a normalization constant: } \sum_{t=0}^{t_{max}} e^{-\frac{1}{2\sigma^2}(t+1-\frac{s}{s_0})^2} \end{aligned} \tag{3}$$

This scheme means that for the first training steps (small s), the loss objective is mainly focused on the reconstruction of the flow from the feature representation ($1-e^{-\frac{s}{s_0}}$ being small), then we ask a good flow reconstruction after one iterative step, and progressively ask a good reconstruction for an increasing number of iterative step (at training step $s = (t+1)s_0$, the loss is focused on achieving a good flow prediction for t iterations of the inpainting network).
In practice we used $\begin{cases} s_0 = 1000 \\ \sigma = 500 \end{cases}$ This scheme, however, did not prevent the network for converging to a state producing the same kind of unwanted behavior (Figure 19): once the scheme has reached the maximum t (set for memory issues purposes), the network is only trained to achieve good reconstruction after this maximal number of steps, and 'forgets' the progressive training.

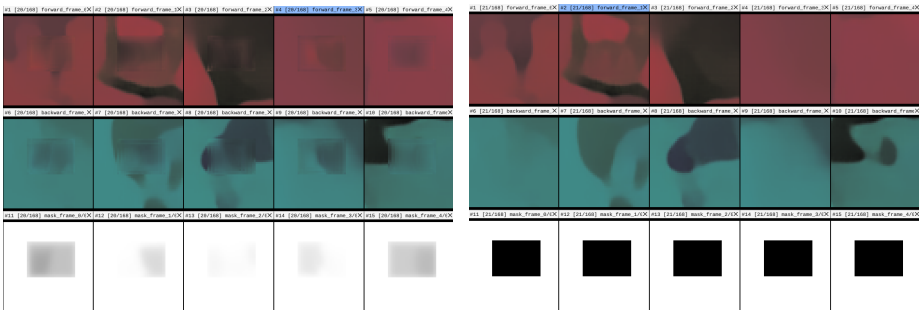
The scheme we ended up using was the following:

$$\begin{aligned} &\forall s \text{ training step}, \begin{cases} \mu_{decoder} = 1 \\ \mu^t_{inpaint} = c.(1 - e^{-\frac{s}{s_0}}).(t+1)^2 \end{cases} \\ &\text{where } c \text{ is a normalization constant: } \sum_{t=0}^{t_{max}} (t+1)^2 \end{aligned} \tag{4}$$

This scheme imposes good reconstruction at every step of the iterative refinement while giving more importance to the final steps. This solves the aforementioned problem.

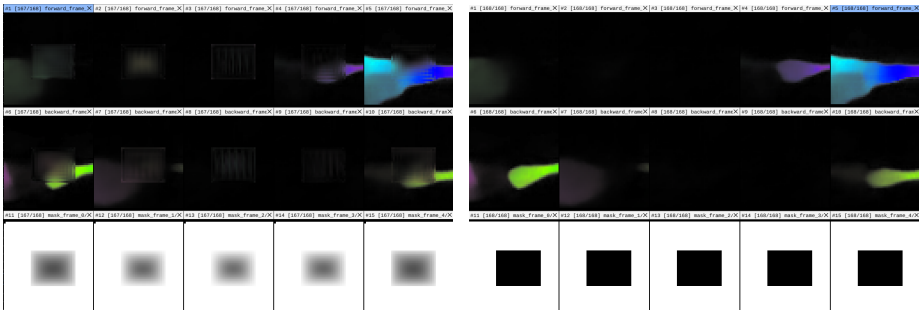
3.6 Some results

For all the following figures, the layout is the following:
top row, forward flows of all frames, middle row: backward flows of all frames, bottom row: confidence masks



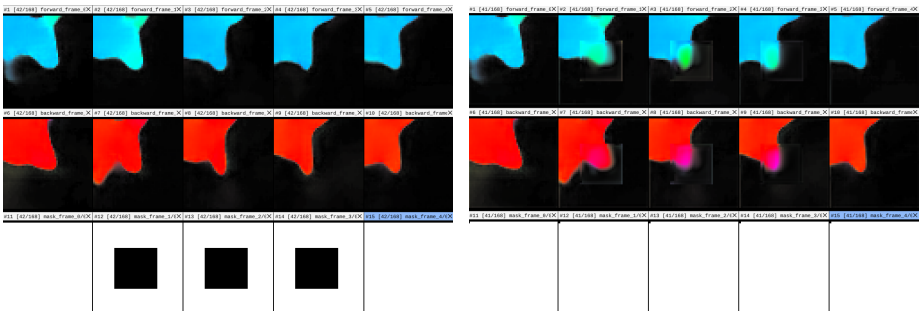
(a) last step of the iterative refinement (b) ground truth

Figure 22: Architecture 1, with power update and scheduled (cf Section 3.6) $\mathcal{L}^1$ supervision



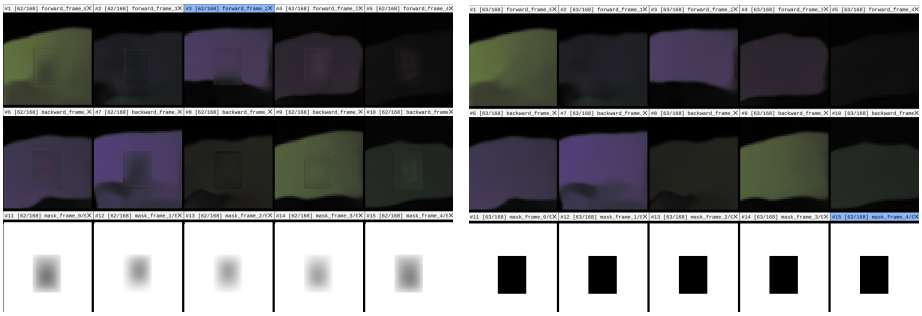
(a) last step of the iterative refinement (b) ground truth

Figure 23: Architecture 1, with power update and scheduled (cf Section 3.6) $\mathcal{L}^1$ supervision, combined with $\mathcal{L}_{minfb/bf}$



(a) last step of the iterative refinement (b) ground truth

Figure 24: Architecture 1, with power update and scheduled (cf Section 3.6) $\mathcal{L}^1$ supervision, combined with $\mathcal{L}_{TV}$, in the 'sandwich' setup, where the first and last frames are not masked.



(a) last step of the iterative refinement (b) ground truth

Figure 25: Architecture 2, with power update and scheduled (cf Section 3.6) $\mathcal{L}^1$ supervision, combined with $\mathcal{L}_{minfb/bf}$.

4 Conclusion

After failing to reproduce an article, we took our take on the problem, exploring and investigating on many topics on the way, such as flow consistency, occlusion issues, warping, edge information incorporation.

We then left this framework of direct inpainting to explore the path of iterative flow refinement, with confidence propagation.

In contrast with my masters 2 internship, which lead to a published article, the year has taught me some aspects of struggle in research: working with other researchers codes, abandoning unsuccessful leads... It's also taught me the power of discussion in research and how a coffee can lead to many ideas in other fields of research.

I would like to thank the hole team for their great supervision, and allowing me to work on our article (together with Coloma) in the beginning.

References

[1] Sergi Caelles, Kevis-Kokitsi Maninis, Jordi Pont-Tuset, Laura Leal-Taixé, Daniel Cremers, and Luc Van Gool. One-shot video object segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 221–230, 2017.

[2] Sergi Caelles, Alberto Montes, Kevis-Kokitsi Maninis, Yuhua Chen, Luc Van Gool, Federico Perazzi, and Jordi Pont-Tuset. The 2018 DAVIS challenge on video object segmentation. *CoRR*, abs/1803.00557, 2018.

[3] Ishaan Gulrajani, Faruk Ahmed, Martín Arjovsky, Vincent Dumoulin, and Aaron C. Courville. Improved training of wasserstein gans. *CoRR*, abs/1704.00028, 2017.

[4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[5] Jia-Bin Huang, Sing Bing Kang, Narendra Ahuja, and Johannes Kopf. Temporally coherent completion of dynamic video. *ACM Transactions on Graphics (TOG)*, 35(6):196, 2016.

[6] Eddy Ilg, Nikolaus Mayer, Tonmoy Saikia, Margret Keuper, Alexey Dosovitskiy, and Thomas Brox. Flownet 2.0: Evolution of optical flow estimation with deep networks. *CoRR*, abs/1612.01925, 2016.

[7] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A. Efros. Image-to-image translation with conditional adversarial networks. *CoRR*, abs/1611.07004, 2016.

[8] Max Jaderberg, Karen Simonyan, Andrew Zisserman, and Koray Kavukcuoglu. Spatial transformer networks. *CoRR*, abs/1506.02025, 2015.

[9] Thuc Trinh Le, Andrés Almansa, Yann Gousseau, and Simon Masnou. Motion-consistent video inpainting. In *2017 IEEE International Conference on Image Processing (ICIP)*, pages 2094–2098. IEEE, 2017.

[10] Thuc Trinh Le, Andrés Almansa, Yann Gousseau, and Simon Masnou. Removing objects from videos with a few strokes. In *SIGGRAPH Asia 2018 Technical Briefs*, page 22. ACM, 2018.

[11] Guilin Liu, Fitsum A. Reda, Kevin J. Shih, Ting-Chun Wang, Andrew Tao, and Bryan Catanzaro. Image inpainting for irregular holes using partial convolutions. *CoRR*, abs/1804.07723, 2018.

[12] Rosanne Liu, Joel Lehman, Piero Molino, Felipe Petroski Such, Eric Frank, Alex Sergeev, and Jason Yosinski. An intriguing failing of convolutional neural networks and the coordconv solution. *CoRR*, abs/1807.03247, 2018.

[13] Kamyar Nazeri, Eric Ng, Tony Joseph, Faisal Z. Qureshi, and Mehran Ebrahimi. Edgeconnect: Generative image inpainting with adversarial edge learning. *CoRR*, abs/1901.00212, 2019.

[14] Maria Oliver, Roberto P Palomares, Coloma Ballester, and Gloria Haro. Spatio-temporal binary video inpainting via threshold dynamics. In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1822–1826. IEEE, 2017.

[15] Maria Oliver, Lara Raad, Coloma Ballester, and G Haro. Motion inpainting by an image-based geodesic amle method. In *2018 25th IEEE International Conference on Image Processing (ICIP)*, pages 2267–2271. IEEE, 2018.

[16] Anurag Ranjan, Varun Jampani, Lukas Balles, Kihwan Kim, Deqing Sun, Jonas Wulff, and Michael J Black. Competitive collaboration: Joint unsupervised learning of depth, camera motion, optical flow and motion segmentation. *arXiv preprint arXiv:1805.09806*, 2018.

[17] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. *CoRR*, abs/1505.04597, 2015.

[18] Rida Sadek, Gabriele Facciolo, Pablo Arias, and Vicent Caselles. A variational model for gradient-based video editing. *International Journal of Computer Vision*, 103(1):127–162, May 2013.

[19] Carles Ventura, Miriam Bellver, Andreu Girbau, Amaia Salvador, Ferran Marques, and Xavier Giro-i Nieto. Rvos: End-to-end recurrent network for video object segmentation. *arXiv preprint arXiv:1903.05612*, 2019.

[20] Xiaolong Wang, Allan Jabri, and Alexei A Efros. Learning correspondence from the cycle-consistency of time. *arXiv preprint arXiv:1903.07593*, 2019.

[21] Saining Xie and Zhuowen Tu. Holistically-nested edge detection. *CoRR*, abs/1504.06375, 2015.

[22] Ning Xu, Linjie Yang, Yuchen Fan, Jianchao Yang, Dingcheng Yue, Yuchen Liang, Brian L. Price, Scott Cohen, and Thomas S. Huang. Youtube-vos: Sequence-to-sequence video object segmentation. *CoRR*, abs/1809.00461, 2018.

[23] Rui Xu, Xiaoxiao Li, Bolei Zhou, and Chen Change Loy. Deep flow-guided video inpainting. *CoRR*, abs/1905.02884, 2019.

[24] Jiahui Yu, Zhe Lin, Jimei Yang, Xiaohui Shen, Xin Lu, and Thomas S Huang. Free-form image inpainting with gated convolution. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 4471–4480, 2019.

[25] Shay Zweig and Lior Wolf. Interponet, A brain inspired neural network for optical flow dense interpolation. *CoRR*, abs/1611.09803, 2016.