# Supplementary material
# BrightFlow: Brightness-Changes-Aware Unsupervised Learning of Optical Flow

## 1. Soft census loss

Let $A$, $B$ be two same-size greyscale images, $p$ a pixel coordinate, $\mathcal{N}(p)$ its neighbors and $\rho$ the soft census loss.

$$\rho(A,B)(p) = \Big(\sum_{q\in\mathcal{N}(p)} H\{C(A(p)-A(q)) - C(B(p)-B(q))\} + \varepsilon\Big)^{\alpha} \quad (1)$$

being $H(z) = z^2/(z^2+\varepsilon)$ the soft Hamming distance and $C(z) = z/\sqrt{\epsilon+z^2}$ the soft census transform. The neighborhood we consider is a 7-pixel-wide patch, $\alpha = 0.4$, $\varepsilon = 0.1$ and $\epsilon = 0.81$. This implementation is the same as DDFlow [3], UFlow [2] and SMURF [4].

## 2. Pseudo-Code of BrightFlow

```
# epochs: number of epochs
# dataloader: loader of batches
# I1_aug and I2_aug: augmented versions of I1 and I2
# OFN: optical flow network
# BCN: brightness correction network
# warp: warping function
# get_occ: compute the occlusions maps
# Census: Census loss
# crop: crop the center of a tensor
# smoothness_loss: apply the smoothness loss formula

for i in range(epochs):
  for (I1, I2, I1_aug, I2_aug) in dataloader:
    F1_aug, F2_aug = OFN(I1_aug, I2_aug)

    # Photometric losses
    O1, O2 = get_occ(F1_aug, F2_aug)

    I1_aug_hat = warp(I2_aug, F1_aug)
    I2_aug_hat = warp(I1_aug, F2_aug)

    C1_aug = BCN(I1_aug, I1_aug_hat.detach(), O1)
    C2_aug = BCN(I2_aug, I2_aug_hat.detach(), O2)

    I1_aug_hat_c = warp(I2_aug + C2_aug * O2, F1_aug.
     detach())
    I2_aug_hat_c = warp(I1_aug + C1_aug * O1, F2_aug.
     detach())

    L_ph_c1 = (abs(I1_aug - I1_aug_hat_c) * O1).sum() /
     O1.sum()
    L_ph_c2 = (abs(I2_aug - I2_aug_hat_c) * O2).sum() /
     O2.sum()

    I1_hat = warp(I2, F1_aug)
    I2_hat = warp(I1, F2_aug)

    with no_grad():
        C1 = BCN(I1, I1_hat, O1)
        C2 = BCN(I2, I2_hat, O2)

    I1_hat_c = warp(I2 + C2 * O2, F1_aug)
    I2_hat_c = warp(I1 + C1 * O1, F2_aug)

    M1 = abs(I1 - I1_hat_c) <= abs(I1 - I1_hat)
    M2 = abs(I2 - I2_hat_c) <= abs(I2 - I2_hat)

    L_ph_f1 = (Census(I1, M1 * I1_hat_c + ~M1 * I1_hat)
     * O1).sum() / O1.sum()
    L_ph_f2 = (Census(I2, M2 * I2_hat_c + ~M2 * I2_hat)
     * O2).sum() / O2.sum()

    #Smoothness loss
    L_sm1 = smoothness_loss(I1, F1_aug)
    L_sm2 = smoothness_loss(I2, F2_aug)

    # Self-supervised loss
    F1_teacher, F2_teacher = OFN(I1, I2)
    F1_student, F2_student = OFN(crop(I1_aug), crop(
     I2_aug))
    L_self1 = Charbonnier(crop(F1_teacher), F1_student)
    L_self2 = Charbonnier(crop(F2_teacher), F2_student)

    loss = gamma_ph * (L_ph_f1 + L_ph_f1 + gamma_ph_c *
     (L_ph_c1 + L_ph_c2)) + gamma_sm * (L_sm1 + L_sm2) +
      gamma_self * (L_self1 + L_self2)
    loss.backward()

    update(OFN.params)
    update(BCN.params)
```

Algorithm 1: Pseudo-Code of BrightFlow in a pytorch style

## 3. Supplementary experiments

### 3.1. Checking what the brightness correction learns

These experiments aim to check that what the brightness correction network learn is:

1. not the trivial solution mentioned in section 3.2.1 of the paper,

2. nor equivalent to a mechanism that filters out outputs of the census loss beyond a certain threshold.

To test these hypotheses, we implemented the following experiments:

1. training RAFT with BrightFlow but using the trivial solution of the photometric loss instead of the outputs of the brightness correction network,

| | $\tau$ | Sintel clean | | Sintel final | |
|---|---|---|---|---|---|
| | | EPE | ER | EPE | ER |
| RAFT | | 3.93 | 8.24 | 3.97 | 11.22 |
| Exp. 1 | | 3.40 | 8.22 | 3.95 | 10.49 |
| Exp. 2 | 2 | 3.98 | 10.64 | 4.23 | 13.78 |
| | 3 | 3.79 | 8.38 | 3.95 | 11.34 |
| | 4 | 3.56 | 7.84 | 4.05 | 11.19 |
| Exp. 3 | 2 | 4.16 | 10.78 | 4.27 | 13.91 |
| | 3 | 3.85 | 8.08 | 3.75 | 10.75 |
| | 4 | 4.35 | 8.49 | 3.82 | 10.91 |
| RAFT+BrightFlow | | **3.25** | **7.49** | **3.33** | **10.26** |

Table 1: Comparison of BrightFlow against our RAFT [5] baseline with three different heuristics: Exp. 1: replacing the brightness correction network predictions with the trivial solution mentioned in section 3.2.1 of the article, Exp. 2: using a thresholded soft census loss and Exp. 3: combining both previous experiments

2. training RAFT without BrightFlow but using a thresholded soft census loss that filters out errors beyond a certain threshold,

3. both the previous experiments at the same time.

In the thresholded soft census loss, the soft Hamming distance is replaced by a thresholded soft Hamming distance $H_\tau(z) = H(z) \times \mathbb{1}_{\{H(z)\leq\tau\}}$, being $\tau$ the threshold. The codomain of the thresholded soft census loss with our hyperparameters and 7 pixels wide patches being in the interval $[0.40, 4.66]$, we have tried 2, 3, 4 as values for $\tau$. All the experiments have been conducted on Sintel [1] with RAFT architecture [5]. Results (see table 1) show that Exp. 1 overtakes the baseline on Sintel clean and final. About Exp. 2 and 3, whatever the threshold, none of the experiments exceeds all the baselines at once. BrightFlow performance remains better than Exp. 1, Exp. 2 and Exp. 3. We deduce that what BrightFlow learns cannot be restricted to a threshold to reduce outliers nor to the trivial solution. Interestingly, if one does not have enough computing resources to train the entire BrightFlow method, one can still use the trivial solution instead of the brightness correction network to get a performance gain but not as much as with the whole BrightFlow.

### 3.2. Impact of data augmentation on performance

Results of experiments with and without data augmentation are available table 2. Whether BrightFlow is used or not, it demonstrates the importance of using photometric data augmentation.

| | Data aug. | Sintel clean | | Sintel final | |
|---|---|---|---|---|---|
| | | EPE | ER | EPE | ER |
| RAFT | | 3.99 | 8.76 | 4.40 | 12.03 |
| | ✓ | 3.93 | 8.24 | 3.97 | 11.22 |
| RAFT+BrightFlow | | 3.59 | 8.18 | 3.77 | 11.29 |
| | ✓ | **3.25** | **7.49** | **3.33** | **10.26** |

Table 2: Impact of photometric data augmentation on performance

## 4. Qualitative results on Sintel

Figure 1 shows qualitative results on Sintel [1].

## References

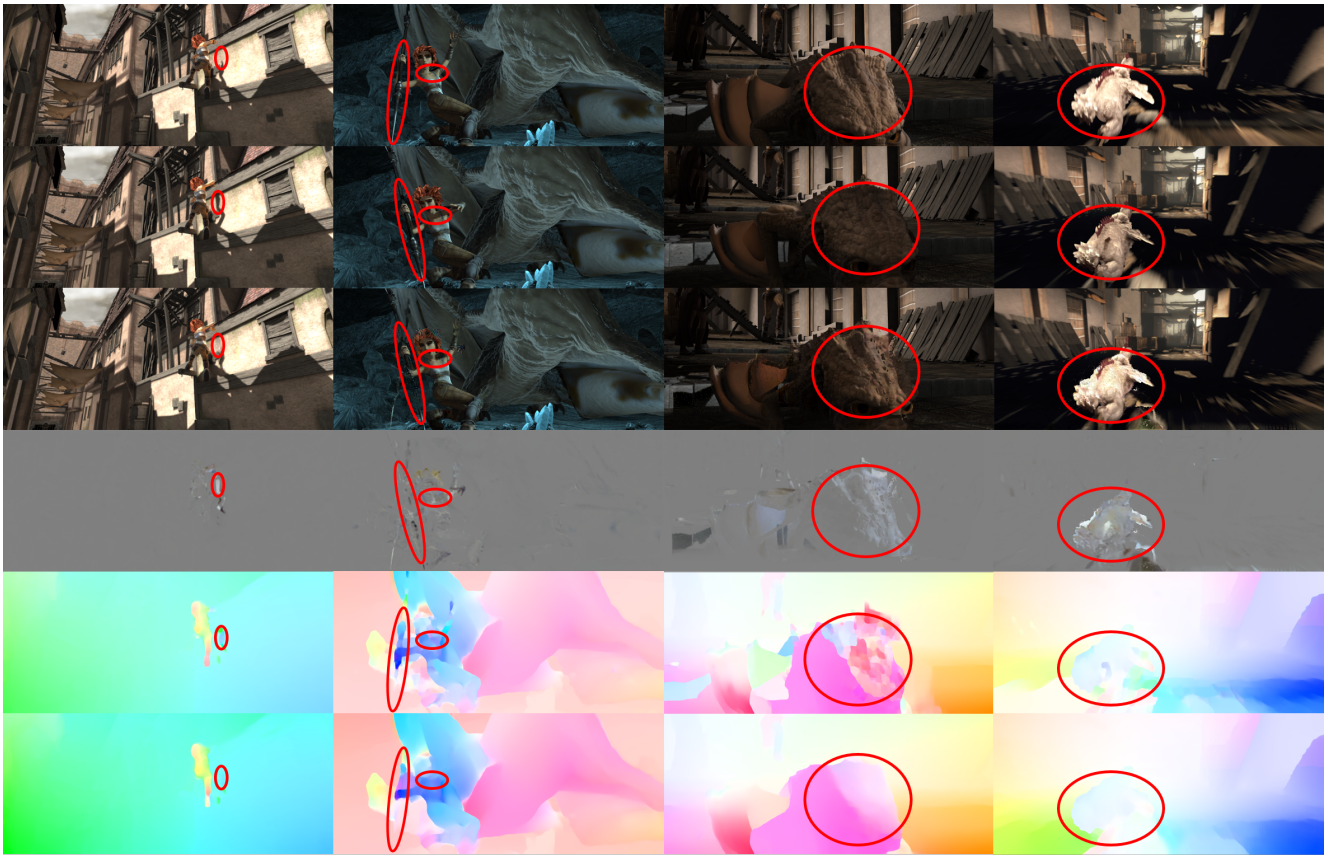

[1] D. J. Butler, J. Wulff, G. B. Stanley, and M. J. Black. A naturalistic open source movie for optical flow evaluation. In *ECCV*, 2012.

[2] Rico Jonschkowski, Austin Stone, Jonathan T Barron, Ariel Gordon, Kurt Konolige, and Anelia Angelova. What matters in unsupervised optical flow. In *ECCV*, 2020.

[3] Pengpeng Liu, Irwin King, Michael R Lyu, and Jia Xu. Ddflow: Learning optical flow with unlabeled data distillation. In *AAAI*, 2019.

[4] Austin Stone, Daniel Maurer, Alper Ayvaci, Anelia Angelova, and Rico Jonschkowski. Smurf: Self-teaching multi-frame unsupervised raft with full-image warping. In *CVPR*, 2021.

[5] Zachary Teed and Jia Deng. Raft: Recurrent all-pairs field transforms for optical flow. In *ECCV*, 2020.



(1) (2) (3) (4)

Figure 1: Qualitative results for RAFT trained with or without BrightFlow on Sintel (best viewed in color). For top to bottom, images corresponds to $I_1$, $I_2$, $I_2 + C_2$, $C_2$, $F_1$ predicted by RAFT trained without BrightFlow and then with BrightFlow. We notice that BrightFlow enables a more consistent object-wise optical flow, especially in case of strong brightness changes ((2), (3), (4)). It also prevents artifacts in the optical flow that come from shadows (1). However, this induces the loss of some small details that the optical flow network learned to ignore due to some incorrect corrections that have not been filtered out (2).