Supplementary Material

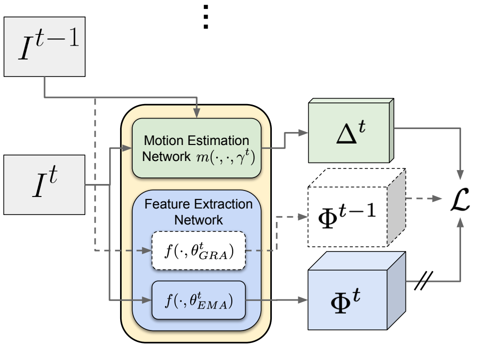
A Appendix

This appendix provides further details about the proposed model. This includes in-depth descriptions of the neural architectures and the selected competitors, the grids of values of the hyper-parameters that we validated in our experimental activity, together with the best selected values. We also provide additional results and qualitative investigations, as well as further information on how to reproduce the results of this paper (including implementation-oriented descriptions and references to the code we share with this paper). Notice that the bibliographic references contained in this appendix are about the bibliography of the main paper.

A.1 Sampling

In Section 2.1 we described our motion-driven sampling procedure, motivated by the fact that the number of positive and negative pairs might easily become large, since it is quadratic in the number of pixels, making Eq. 9 computationally costly. This choice is loosely inspired to what was done by [40], where the authors proposed to randomly sub-sample the pixel coordinates in function of an externally computed focus of attention trajectory. Differently, we propose a selection criterion based on motion and representations, following the spirit on which our whole proposal is rooted. In order to provide more formal descriptions, we recall that given any pair of frames of $\mathcal{S}$ from which we compute Δ^t, we model an ad-hoc distribution over the pixel coordinates to inject priors into the sampling procedure. Each pixel x at time t has probability P_x^t of being sampled, with $\sum_{x\in\mathcal{X}} P_x^t = 1$. We compute $\{P_x^t, x \in \mathcal{X}\}$ ensuring that ($i.$) the probability of sampling a point in moving areas is the same as sampling in static areas, and that ($ii.$) the probability of sampling in areas where the j-th feature has the strongest activation (absolute value) is the same for all j's. The rationale behind this idea is that ($i.$) we want to ensure that in scenes with limited motion the sampling is still biased toward moving areas, since the contrastive loss is built on motion information, and that ($ii.$) we want to foster the development of all the d features. Formally, $\mathcal{M}$ is the set of coordinates of moving pixels and $\neg\mathcal{M}$ are the coordinates of the static points.[7] We compute $\mathcal{F}_j$, $j = 1, \ldots, d$, where set $\mathcal{F}_j$ collects the coordinates of the pixels for which the j-th component of the learned representation is the largest one, i.e., $\mathcal{F}_j = \{z\colon \arg\max |\Phi_z^{t-1}| = j\}$.[8] Each pixel coordinate x belongs to one of the following $2d$ possible sets, $\{\mathcal{A}_j = \mathcal{F}_j \cap \mathcal{M},\ \mathcal{A}_{2j} = \mathcal{F}_j \cap \neg\mathcal{M},\ j = 1, \ldots, d\}$, and P_x^t is set to $1/(2d{\cdot}\mathrm{card}(\mathcal{A}_z))$, being $\mathcal{A}_z$ the set to which x belongs and $\mathrm{card}(\mathcal{A}_z)$ its cardinality.[9] This yields $\sum_{x\in\mathcal{A}_1} P_x^t = \ldots = \sum_{x\in\mathcal{A}_{2d}} P_x^t = \frac{1}{2d}$, and we sample $\ell > 1$ pixel coordinates for each t, collecting their coordinates into $\mathcal{X}^t$. The outer sum of Eq. 9 is then restricted to the coordinates belonging to $\mathcal{X}^t$. Fig. 1 (b) highlights with white circles the elements of $\mathcal{X}^t$ (toy example). An additional speed-up of the training process can be achieved by further limiting the summations of Eq. 9 (both of them) by keeping only a fraction $\aleph \in (0, 1]$ of the positive and negative pairs (balanced). In particular, we experimented the case in which we kept those positive (resp. negative) pairs for which representations are the least similar (resp. most similar), i.e., focusing on the ones that mostly contribute to larger values of Ω_f. We remark that identifying the least/most similar pairs of representations can be efficiently done *after* having computed $\mathcal{X}^t$, where the number of pairs to evaluate can be easily controlled acting on ℓ. Differently, the sampling procedure that builds $\mathcal{X}^t$ is purposely based on properties of single pixels (motion and strongest feature), instead of pairs, making it scalable.

A.2 Learning over Time


$\vdots$
I^{t-1}
I^t
Motion Estimation Network $m(\cdot,\cdot,\gamma^t)$
Feature Extraction Network
$f(\cdot,\theta^t_{GRA})$
$f(\cdot,\theta^t_{EMA})$
Δ^t
Φ^{t-1}
Φ^t
$\mathcal{L}$

```
Require: Stream $\mathcal{S}$ of length $T$, potentially infinite; neural nets $m$ and
    $f$ with initial parameters $\gamma^1$ and $\theta^1$. Learning rates $\alpha_m, \alpha_f > 0$ and
    $\xi \in [0, 1)$.
    $t \leftarrow 1,\ \theta^1_{GRA} \leftarrow \theta^1,\ \theta^1_{EMA} \leftarrow \theta^1,\ I^0 \leftarrow I^1,$
    while $t \leq T$ do
        $I^t \leftarrow$ next_frame($\mathcal{S}$)
        $\Delta^t \leftarrow m(I^{t-1}, I^t, \gamma^t)$
        $\Phi^{t-1} \leftarrow f(I^{t-1}, \theta^t_{GRA})$
        $\Phi^t \leftarrow f(I^t, \theta^t_{EMA})$
        $\gamma^{t+1} \leftarrow \gamma^t - \alpha_m \nabla_\gamma \mathcal{L}(\Phi^{t-1}, \Phi^t, \Delta^t)$
        $\theta^{t+1}_{GRA} \leftarrow \theta^t_{GRA} - \alpha_f \nabla_\theta \mathcal{L}(\Phi^{t-1}, \hat{\Phi}^t, \Delta^t)$
        $\theta^{t+1}_{EMA} \leftarrow \xi\theta^t_{EMA} + (1-\xi)\theta^{t+1}_{GRA}$
        $t \leftarrow t + 1$
    end while
```

Figure 5. Learning over time with CMOSS. (Top) An Exponential Moving Average (EMA) network and a gradient-updated network (GRA) extract features from I^t and I^{t-1}, respectively. White-dotted parts of the model are in addition to the ones of Fig. 1 (a), and the slanted bars indicate that no gradients are propagated on that path. The loss function $\mathcal{L}$ of Eq. 3 drives the learning process of the motion predictor and of the feature extractor, enforcing GRA to be consistent with EMA and instantiating the motion-induced contrastive criterion. (Bottom) The learning procedure.

We propose to extract features from I^{t-1} with network f using weights θ^t_{GRA} (GRAdient-updated), and to extract features from I^t with an EMA-updated network with the same architecture and weights θ^t_{EMA},

$$\Phi^{t-1} = f(I^{t-1}, \theta^t_{GRA}), \qquad \Phi^t = f(I^t, \theta^t_{EMA}),$$

as shown in Fig. 5, with $\theta^1_{GRA} = \theta^1_{EMA}$. Then learning proceeds following the algorithm in the figure.

A.3 Neural Architectures

We provide additional details on the selected neural architectures and on the competitors.

Feature Extractors. The neural branch of CMOSS yielding features, i.e., function f, is a ResUnet architecture, inspired to the one presented in [40] but having a smaller amount of parameters.

[7] Moving and static points are determined as already introduced when describing function p and n.

[8] We use Φ^{t-1} to be coherent with the coordinate frame of Δ^t, and $|a|$ is the element-wise absolute value of vector a.

[9] The overall number of sets ($2d$) can be smaller in the case of one or more empty intersections.

It is a UNet-like [34] architecture[10], based on a ResNet18 backbone. Among the pretrained competitors, DPT-Hybrid [32] is a Transformer-based ViT model, trained on the ADE20K dataset [48]. All the other considered models adopt ResNet backbones, in particular the ResNet-50 architecture, except for DeepLab v3 (ResNet-101). DeepLab models apply Atrous Spatial Pyramid Pooling (ASPP) on top of the backbone features and they are trained on COCO dataset [21][11]. MoCo models[12] are trained with a contrastive unsupervised objective on ImageNet-1M. MoCo v2 differs from the original version because of an improved training procedure (better augmentation, cosine learning rate schedule, extra MLP in training phase). MoCo v3[13] introduces some tricks to improve the learning stability both for ViT and Resnets backbones. We focus our analysis on the latter backbone. In the case of PixPro[14] models, we use backbone weights obtained from pretraining on ImageNet-1M with their pixel-level contrastive task. Concerning all the pretrained ResNets, we upsampled the features picked at the last three residual stages, and reported the configuration achieving the best performance.

Motion Estimation. CMOSS neural branch for motion estimation, i.e., function m, inherits its structure and composition from the feature extraction one (ResUnet), except that we cut the skip connections connecting the input to the output layer, since we empyrically noticed in preliminary experiments that we could obtain a more sensible estimation. Such a network outputs the vertical and horizontal displacement maps for all the frame pixels, collected in Δ.

A.4 *Additional implementation details and hyper-parameters*

In order to complement the already described experimental setup of Section 3, here we introduce some further details to make our experience reproducible. For completeness we report that, in our implementation, the maximum distance between pairs of pixels in Eq. 8 (i.e., $\sqrt{h^2+w^2}$ in the normalization factors) is replaced with the maximum distance between pairs of sampled coordinates. Concerning real-world streams HORSE and RAT, we provide[15] preprocessed files in order to easily reproduce the experiments.

Data augmentation. For each processed frame pair, we create a mini-batch composed of the two original frames (I^{t-1}, I^t), as well as multiple augmented views/transformations, in order to improve the feature robustness, as common in deep net training procedures. Augmentations are of 3 different types: (A) large random crops (with crop ratio greater than 0.9) followed by upscaling to $w \times h$; (B) random horizontal/vertical flips; (C) random color distortion and Gaussian blur. For the augmentations of type (A) and (C), only one of the frames composing the pair (I^{t-1}, I^t) is transformed, randomly selected at each t, whilst the other is kept as it is.

Evaluating representations. The quality of the developed pixel-wise representations is evaluated strictly following the benchmark proposed in [40]. In particular, such a benchmark consists in an initial unsupervised representation learning stage in which data is progressively streamed over time (30 laps per object), as described in the main paper, followed by a supervision stage in which the model keeps processing the stream and receives a total of 3 pixel-wise supervisions per object, spaced out by at least 100 frames (5 laps). Supervisions are not used for representation learning purposes, and they are provided on those pixels that belong to the trajectory of a human-like focus of attention model [45]. Thus, the feature vectors associated to the labeled pixels are stored as templates in M. The evaluation stage consists in still processing the streamed data (1 lap), not only extracting features but also providing them to an open-set classifier $c(\Phi_x, M)$ that predicts the class-membership scores of pixel coordinates x given the representation Φ_x produced by network f. Scores are about a certain number of classes that depends on the considered stream. The quality of the predictions of c is measured by the F1 index. The classifier c is implemented as a distance-based model (cosine similarity in our case). In such a way, the model is capable not only to predict the target classes, but also to not make any predictions when the minimum distance from all the templates is greater than a certain threshold ξ (open-set, see [40]).

Parameters and validation. In order to make the comparisons fair, we followed the exact same hyper-parameter validation procedure of [40]. In particular, for each video stream we selected the values of the hyper-parameters which maximize the F1 score measured during the 30-th lap, only considering one pixel per frame, along the trajectory of a human-like focus of attention mechanism attending the scene. Given that the stream HORSE is much longer than the others, in that case learning proceeds for 15 laps. We sampled the best hyper-parameters from the following grids: $\lambda_m \in \{1\}$, $\beta_m \in \{10^{-4}, 10^{-3}, 10^{-2}, \}$, $\beta_f \in \{1\}$, $\tau_p \in \{0.7, 0.8, 0.9\}$, $\tau_n \in \{-0.5, -0.3, 0, 0.3, 0.5, 0.7\}$, $\ell \in \{50, 100, 200\}$, $\aleph \in \{0.5, 0.7, 1.0\}$, $\tau \in \{0.1, 0.3, 0.5\}$, $\tau_m \in \{0.5, 1.0, 1.5, 2.0, 2.5\}$, $\alpha_m \in \{10^{-4}, 5 \cdot 10^{-4}, 10^{-3}, 10^{-2}\}$, $\alpha_f \in \{10^{-4}, 5 \cdot 10^{-4}, 10^{-3}, 10^{-2}\}$, $\xi \in \{0.5, 0.9, 0.99\}$. The motion estimator m is optimized using Adam [18], while we empirically found better results by optimizing the features extractor f with SGD (with the exception of the EMPTYSPACE stream, where Adam usually yields better performances). In addition to the regularizer introduced in Eq. 5, we also exploited a further regularization modulated by a customizable parameter β_{mr}, in order to keep the estimated motion magnitude small in the case of uniform backgrounds,

$$\Omega_r(\Delta^t) = (wh)^{-1} \sum_{x \in \mathcal{X}^t} \|\Delta^t_{x,1}\|^2 + \|\Delta^t_{x,2}\|^2, \qquad (13)$$

with $\beta_{mr} \in \{5 \times 10^{-3}, 10^{-3}, 10^{-2}\}$. We also tuned the relative importance of the leftmost term in Eq. 6, using a scalar $\beta_{\bowtie} \in \{10^{-4}, 10^{-3}, 10^{-2}\}$. In all our experiments, we reported the average results obtained over 10 runs, initializing random generators with seeds (`torch.manual_seed` in PyTorch) from the following list: $\{1234, 123456, 12345678, 1234567890, 9876, 98765, 987654, 9876543, 98765432, 987654321\}$.

We report in Tab. 2 the best-found values for the hyperparameters. We provide further details in descriptions attached to the code we shared with this paper (Appendix A.9).

A.5 *Additional Results*

We report in Tab. 3 the F1 score computed during the validation procedure. We recall that, in such a procedure, the F1 computation is limited to predictions on one pixel per frame, along the focus-of-attention trajectory, as described in Appendix A.4. Lower performances of CMOSS are expected in this metric with respect to those approaches that are specifically designed to learn around the focus of attention coordinates.

[10] Inspired to https://github.com/usuyama/pytorch-unet (MIT License), but with one fourth of filters in each layer. Also, the last layer has an output dimensionality of 32 features

[11] Available at https://pytorch.org/vision/stable/models.html

[12] Available at https://github.com/facebookresearch/moco

[13] Available at https://github.com/facebookresearch/moco-v3

[14] Available at https://github.com/zdaxie/PixPro

[15] Available at https://drive.google.com/file/d/1sB3idnhbaBdOggWccVqyIDBbr3lJMGuI/view?usp=sharing

Table 2. Optimal parameters. The best selected hyperparameters drawn from the grids described in the text, for all the datasets. See the code for further details.

	EMPTYSPACE		SOLID	LIVINGROOM		RAT	HORSE
param.	BW	RGB	BW	BW	RGB	RGB	RGB
$\aleph$	1.0	0.7	1	1	1	1	1
α_f	10^{-3}	10^{-3}	10^{-2}	10^{-3}	5.0×10^{-4}	10^{-3}	10^{-2}
α_m	10^{-4}	10^{-4}	10^{-4}	10^{-4}	10^{-4}	5×10^{-4}	10^{-4}
β_f	1	1	1	1	1	1	1
β_m	10^{-4}	10^{-4}	10^{-4}	10^{-4}	10^{-4}	10^{-3}	10^{-3}
ℓ	100	100	100	100	100	100	200
λ_m	1	1	1	1	1	1	1
τ	0.5	0.5	0.5	0.5	0.5	0.3	0.1
τ_m	1	1	2	2	1.5	2.5	2.5
τ_n	-0.3	-0.3	-0.3	-0.3	0	0.7	-0.5
τ_p	0.9	0.9	0.7	0.8	0.8	0.7	0.7
ξ	0.99	0.99	0.99	0.99	0.99	0.99	0.99
$\beta_{\bowtie}$	10^{-2}	10^{-4}	10^{-2}	10^{-4}	10^{-2}	10^{-2}	10^{-3}
β_{mr}	5×10^{-3}	5×10^{-3}	10^{-3}	10^{-2}	5×10^{-3}	10^{-2}	5×10^{-3}

Furthermore, the validation procedure dictated by the considered benchmark favours the discovery of parameter configurations that are well-suited for each considered stream, but that do not necessarily transfer well to the other streams. Noticeably, during our experimental activity with CMOSS we found a set of hyperparameter values that turned out to attain good performances in all the considered streams. We denote the model using such values with CMOSS†, and we report both its F1 when measured on the attention trajectory (Tab. 3) and when computed over the whole frames (Tab- 4). We remark that CMOSS† achieves performance very similar/better (EMPTYSPACE-RGB, LIVINGROOM) to the ones of the main competitor [40], which uses specialized sets of parameter values for each stream.

A.6 Transferability of visual features

To better understand the CMOSS generalization to unseen data, we set up additional experiments. In particular, we consider the best performing models from Tab. 1 in the main paper and test their capability of generalizing to unseen environments of different streams (without further learning). We neglect LIVINGROOM since its visual appearance is extremely different and the comparison would not be informative. We report results in Tab. 5, where the F1 obtained on some evaluation stream should be compared with the result reported in the diagonal cell (gray) of the same column. For instance, the agent learned in the environment SOLID-BW is evaluated on EMPTYSPACE-BW achieving 90% of the F1 obtained by the specific agent (0.54 vs 0.60). Similar considerations holds for the other streams, meaning that learnt features are not strictly related to the objects experienced during training. We remark that transferability is marginal in the learning protocol we consider. In fact, the learning agent is supposed to autonomously develop representations useful for the environment where it learns (lifelong learning).

A.7 Full ablations

In Fig. 6 we show results anticipated in Fig. 3, now considering all the datasets. In any case, the comments in the main paper still align with results on the other datasets. Moreover, we additionally propose two other ablations. An additional speed-up of the training process can be achieved by further limiting the summations of Eq. 9 (both of them) by keeping only a fraction $\aleph \in (0, 1]$ of the positive and negative pairs (balanced). In particular, we experimented the case in which we kept those positive (resp. negative) pairs for which representations are the least similar (resp. most similar), i.e., focusing on the ones that mostly contribute to larger values of Ω_f. Selecting only the most violated negative/positive pairs, $\aleph < 1$, (Fig. 6 (a-5)) improved the performance on EMPTYSPACE, but not on the other streams (where a finer-grained tuning would help). Concerning our contrastive learning criterion, we wondered whether the variability of the amount of negative pairs over time hampers the stability of the learning process (Fig. 6 (a-6)). However, it turned out that averaging, instead of summing, the terms in the rightmost summation in the denominator of Eq. 9 results in decreased performance.

A.8 Additional Qualitative Analysis

Fig. 7 (right) reports examples of motion predictions obtained for the three different streams. Noticeably, the estimated motion fields (third row) are very similar to the ground truth motion yielded by the 3D environment (second row), sometimes a bit thicker, coherently with our previous comment. It should be noted that optical flow methods inherently find very challenging to extract motion on nearly texture-free surfaces, as the ones of the cylinder in the SOLID dataset. Conversely, the 3D engine possesses a complete understanding of the actual 3D motion field. In Fig. 7 (left) we display again Fig. 3 for better readability. In that figure we presented a further analysis of the performance of the selected model for the EMPTYSPACE-RGB stream along the temporal dimension, letting objects perform an additional lap, and measuring the evolution of the F1 at several fractions of the lap.

We report in Fig. 8 some additional qualitative results of the predictions obtained leveraging the features by CMOSS, on six frames from the EMPTYSPACE-RGB stream (first row). The model by [40] yields features which are not capable to correctly discriminate pixels from the chair legs/thinner segments, as long as some textures on the pillow: see for instance the borders of the pillow which are misled with the ewer (given the similar brightness/RGB values). Similarly, some parts of the ewer (the spout and the handle) are mistaken for pixels belonging to the pillow. Conversely, CMOSS allows the classification procedure (which is not involved in the feature learning process) to almost completely disentangle all the different objects and their parts, even when they appear in peculiar poses and orientations (see the chair legs in the second and fifth image). As a general consideration, CMOSS features (and thus predictions) tend to slightly overflow with respect to the ground-truth object borders.

A.9 Code

We provide a Python implementation of CMOSS, as well as instructions to run experiments, attached as additional material in the submission files. The README file contained in the archive describes how to execute the code, and it gives further details on the mapping between the notation used in the main paper for naming hyperparameters and the one used in the code (see "RUNNING EXPERIMENTS - Argument description/mapping with respect to the paper notation" therein).

A.10 Efficiency Analysis

CMOSS is agnostic to the specific structure of the model exploited for feature extraction and the one about motion estimation, thus the user can determine what is the most appropriate trade-off between

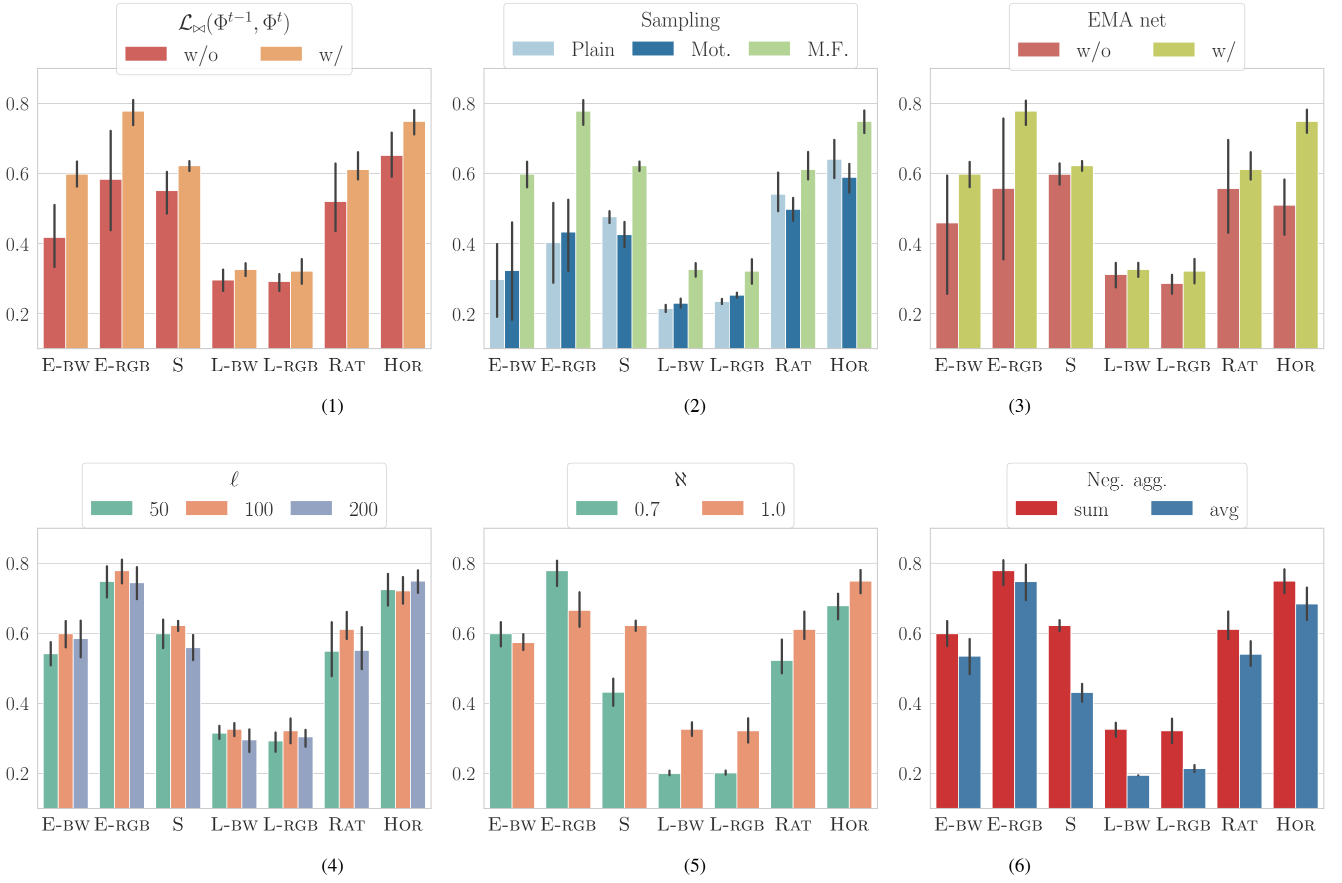


Figure 6. Ablation of the model components on all the streams (E, S, L stands for EMPTYSPACE, SOLID, LIVINGROOM). (a-1) features-motion consistency term $\mathcal{L}_{\bowtie}$; (a-2) sampling strategy; (a-3) EMA network; (a-4) number of sampled points ℓ; (a-5) fraction ℵ of kept exemplars; (a-6) aggregation of negative exemplars in the contrastive loss. F1 scores on the y-axis. See the main text for details.

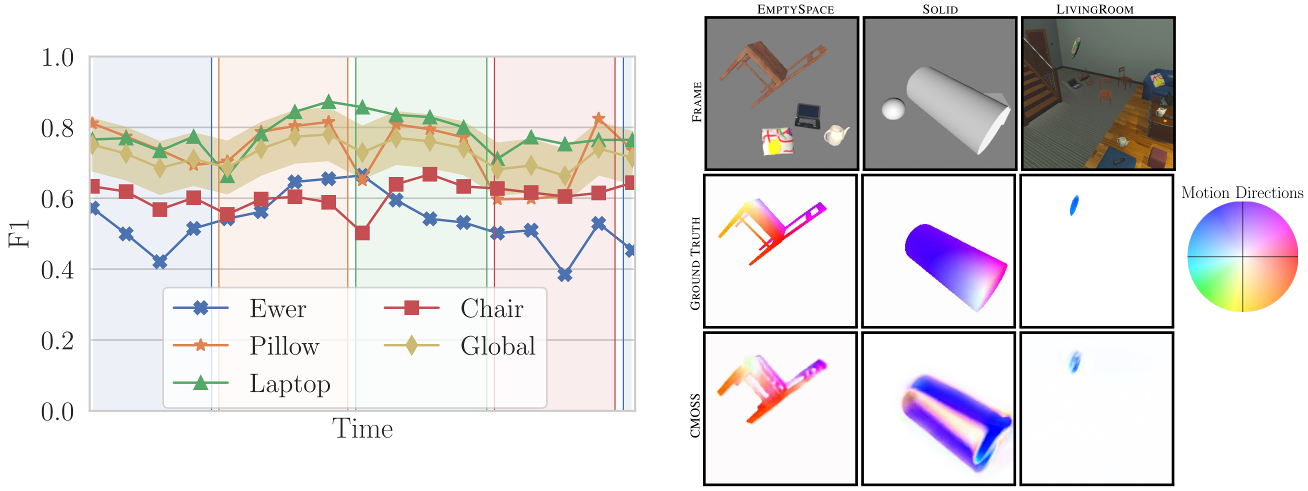


Figure 7. (Left): Evolution of the learning process (EMPTYSPACE) right after the time in which results of Tab. 1 were computed and until each object has performed an additional lap (avg of 10 runs). Each curve is the F1 of an object-specific predictor (ewer, pillow, laptop, chair) and the overall F1 (Global, with std reported as a partially-transparent overlay). Vertical colored bands indicate when each object is moving. (Right): frames from three streams (first-row, EMPTYSPACE-RGB, SOLID, LIVINGROOM-RGB), the ground truth dense motion fields from the 3D environment (second row) and motion estimated by CMOSS (third row).

Table 3. F1 score (mean ± std), 10 runs measured along the focus of attention trajectory. Focus of attention is computed by an external algorithm, provided by the selected benchmark. The scores are used for validating the optimal values of the hyper-parameters, as indicated in the benchmark rules.

		# Params	EmptySpace		Solid	LivingRoom		Rat	Horse
			BW	RGB	BW	BW	RGB	RGB	RGB
Offline Pre-Trained	DPT-C [32]	121M	0.70	0.76	0.54	0.38	0.39	0.76	0.99
	DPT-B [32]	120M	0.73	0.77	0.53	0.29	0.32	0.76	0.96
	DeepLab-C [7]	58.6M	0.46	0.57	0.56	0.19	0.26	0.53	0.99
	DeepLab-B [7]	42.5M	0.61	0.72	0.66	0.28	0.35	0.85	0.90
	MoCo v1 [15]	8.5M	0.79	0.82	0.79	0.58	0.56	0.85	0.56
	MoCo v2 [10]	8.5M	0.76	0.79	0.73	0.59	0.59	0.76	0.95
	MoCo v3 [11]	8.5M	0.78	0.80	0.77	0.64	0.48	0.76	0.71
	PixPro [43]	8.5M	0.63	0.74	0.63	0.11	0.38	0.76	0.76
Continual	Raw Image	-	0.39	0.61	0.32	0.52	0.65	0.76	0.76
	HourGlass [40]	17.8M	**0.73**±0.03	0.77±0.05	**0.68**±0.02	**0.59**±0.02	**0.45**±0.12	0.59±0.10	**0.91**±0.05
	FCN-ND [40]	0.1M	0.69±0.03	0.57±0.08	0.58±0.07	0.39±0.02	0.38±0.02	0.49±0.04	0.67±0.20
	CMOSS	1.1M	0.67±0.08	**0.82**±0.09	0.60±0.04	0.41±0.09	0.40±0.07	0.74±0.08	0.88±0.12
	CMOSS†	1.1M	0.61±0.08	0.72±0.08	0.50±0.05	0.37±0.09	0.34±0.15	0.64±0.13	0.85±0.14

Table 4. F1 scores over 10 runs (mean ± std), over seven different video streams (EmptySpace, Solid, LivingRoom, Rat, Horse) with some -BW variants. The bottom part of the table is about the main competitors of the proposed model, including the raw-image (degenerate) baseline. The top part of the table collects reference results obtained with large offline-pretrained models, publicly available, and an offline-trained version of CMOSS.

		# Params	EmptySpace		Solid	LivingRoom		Rat	Horse
			BW	RGB	BW	BW	RGB	RGB	RGB
Offline Pre-Trained	DPT-C [32]	121M	0.66	0.67	0.64	0.35	0.39	0.59	0.83
	DPT-B [32]	120M	0.71	0.69	0.68	0.39	0.39	0.58	0.87
	DeepLab-C [7]	58.6M	0.49	0.61	0.57	0.31	0.34	0.56	0.86
	DeepLab-B [7]	42.5M	0.70	0.65	0.66	0.34	0.44	0.57	0.81
	MoCo v1 [15]	8.5M	0.73	0.73	0.74	0.33	0.35	0.70	0.66
	MoCo v2 [10]	8.5M	0.75	0.76	0.74	0.41	0.43	0.59	0.79
	MoCo v3 [11]	8.5M	0.76	0.76	0.76	0.41	0.44	0.58	0.64
	PixPro [43]	8.5M	0.31	0.46	0.41	0.30	0.22	0.59	0.70
	CMOSS (Offline)	1.1M	0.64	0.82	0.65	0.43	0.40	0.69	0.81
Continual	Raw Image	-	0.50	0.45	0.18	0.10	0.23	0.52	0.66
	HourGlass [40]	17.8M	0.55±0.03	0.71±0.03	0.50±0.01	0.31±0.04	0.25±0.07	0.59±0.08	0.71±0.07
	FCN-ND[40]	0.1M	**0.60**±0.05	0.51±0.07	0.48±0.03	0.24±0.01	0.28±0.03	0.42±0.05	0.50±0.08
	CMOSS	1.1M	0.60±0.06	**0.78**±0.06	**0.62**±0.02	**0.33**±0.03	**0.34**±0.05	**0.61**±0.05	**0.75**±0.06
	CMOSS†	1.1M	0.56±0.09	0.72±0.04	0.49±0.02	0.32±0.04	0.33±0.07	0.60±0.04	0.73±0.03

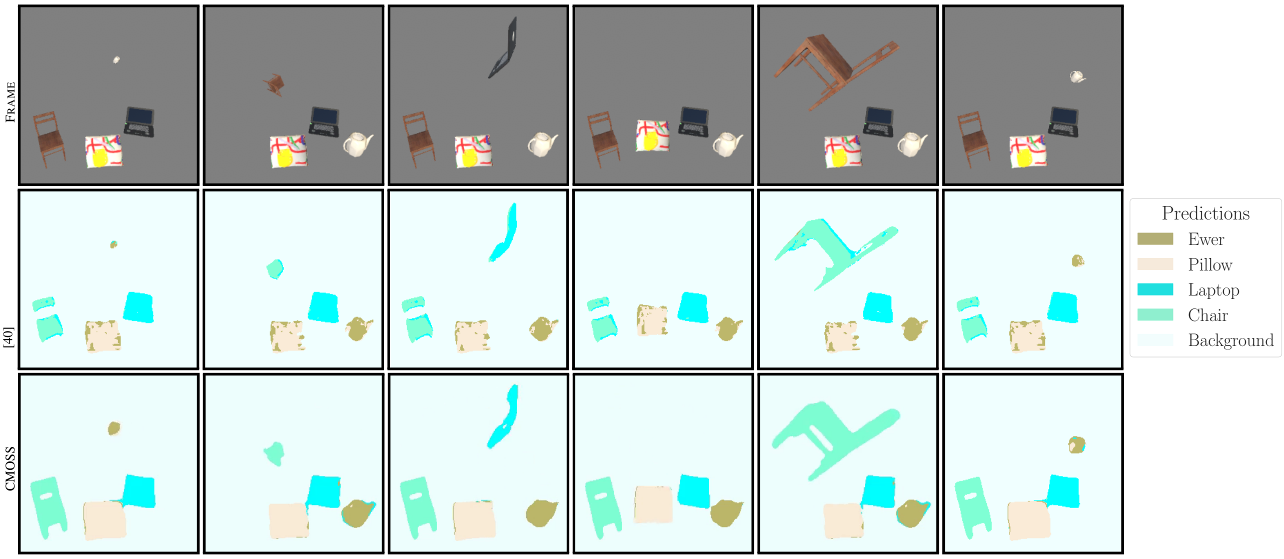


Figure 8. Six frames sampled from the EmptySpace-RGB stream (first row). We compare the predictions yielded by the main competitor (second row) and the outputs by CMOSS (third row). Different colors are used to denote different predictions.

Table 5. Transferability experiment. F1 scores, measured over the whole frame area. Cells along the diagonal are taken from the main results (training and evaluation happens on the same stream).

	Evaluation stream		
Training stream	E-BW	E-RGB	S-BW
E-BW	0.60	0.56	0.51
E-RGB	0.43	0.78	0.38
S-BW	0.54	0.58	0.62

computational time and quality of the predictions, ensuring the fulfilment of real-time processing constraints, if needed. The model reported in the main paper is actually able to process the video stream at approx. 20 fps on the machine we describe in the experimental section. When logging data (storing the metrics, outputs and data needed to collect experimental evidences), overall processing is of course slower than that. Roughly, each single run took 30-50 minutes depending on the processed stream.