CroCo: Self-Supervised Pre-training for 3D Vision Tasks by Cross-View Completion

Appendix

Philippe Weinzaepfel **Vincent Leroy** **Thomas Lucas**

Romain Brégier **Yohann Cabon** **Vaibhav Arora** **Leonid Antsfeld**

Boris Chidlovskii **Gabriela Csurka** **Jérôme Revaud**

NAVER LABS Europe
`https://europe.naverlabs.com/research/computer-vision/croco/`

This appendix is structured as follows. In Section A, we first provide a list of acronyms used throughout the paper, followed by further details about our decoder architectures (Section B). In Section C we detail the parameter setting used in our pre-training and we provide an analysis about how the overlap between the training image pairs affects the pre-training and downstream performance (Section C.2) as well as an ablation on the impact of the decoder depth (Section C.3). Section D then gives further details about monocular tasks, including a set of experiments on a complementary task, namely absolute pose regression (Section D.4). In Section E, we give more details and visual examples concerning the binocular tasks as well as a complementary task, namely stereo matching (Section E.3). Next, we list assets and computing resources in Section F. Finally, we provide additional reconstruction examples and introduce a video of sample scene reconstructions (Section G).

A List of acronyms

We provide the list of acronyms used in the paper, excluding acronyms related to papers or datasets with direct references.

acronym	meaning
Acc@X	Accuracy at a certain error threshold X
AEPE	Average EndPoint Error
CroCo	Cross-view Completion
DINO	DIstillation with NO labels [6]
DPT	Dense Prediction Transformer [24]
FLOPs	Floating-Point Operations
IN1K	ImageNet-1K [26]
mIoU	mean Intersection-over-Union
MAE	Masked Auto-Encoder [14]
MIM	Masked Image Modeling
MLP	Multi-Layer Perceptron
MSE	Mean Squared Error
NLP	Natural Language Processing
RPR	Relative Pose Regression
ViT	Vision Transformers [11]

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

B Details on the decoder architectures

In this section, we provide detailed equations about the blocks used in the CroCo model architecture.

For the encoder, we use a standard transformer block. Let $X \in \mathbb{R}^{N \times D}$ be the N D-dimensional tokens as input to the transformer block. The transformer block computes:

$$\begin{aligned}
\bar{X} &= \text{LayerNorm}(X) \\
X' &= X + \text{Attention}\left(W_Q^S \bar{X}, W_K^S \bar{X}, W_V^S \bar{X}\right) \\
\text{Output} &= X' + \text{MLP}(\text{LayerNorm}(X')),
\end{aligned} \tag{1}$$

where W_Q^S, W_K^S, W_V^S are learnable parameters. In practice, a bias is also learned and applied to these 3 projections but are omitted in the equations for the sake of clarity. The attention itself is computed classically as:

$$\text{Attention}(Q, K, V) = \text{Proj}\Big(\text{softmax}\left(\frac{QK^\top}{\sqrt{D}}\right) V\Big), \tag{2}$$

where Proj denotes a projection layer, *i.e.*, a fully-connected layer.

We now provide equations for the two blocks we have tried in the decoder, namely the *CrossBlock* and the *CatBlock*.

In the *CrossBlock* decoder architecture, self-attention and cross-attention are used one after the other. Let $X, Y \in \mathbb{R}^{N \times D}$ be the N input tokens to the block from the two views respectively. The CrossBlock output is computed by:

$$\begin{aligned}
\bar{X} &= \text{LayerNorm}(X) \\
\bar{Y} &= \text{LayerNorm}(Y) \\
X' &= X + \text{Attention}\left(W_Q^S \bar{X}, W_K^S \bar{X}, W_V^S \bar{X}\right) \\
X'' &= X' + \text{Attention}\left(W_Q^C \text{LayerNorm}(X'), W_K^C \bar{Y}, W_V^C \bar{Y}\right) \\
\text{Output} &= X'' + \text{MLP}(\text{LayerNorm}(X'')),
\end{aligned} \tag{3}$$

where W_Q^S, W_K^S, W_V^S denote learnable parameters for the self-attention and likewise W_Q^C, W_K^C, W_V^C for the cross-attention.

For the *CatBlock* decoder architecture, and following the same notations, we first concatenate X and Y to form $Z = [X + v_1, Y + v_2] \in \mathbb{R}^{2N \times D}$ before the first block, where v_1, $v_2 \in \mathbb{R}^D$ are learnable embeddings specifying the input view. The CatBlock output is computed by:

$$\begin{aligned}
\bar{Z} &= \text{LayerNorm}(Z) \\
Z' &= Z + \text{Attention}\left(W_Q^S \bar{Z}, W_K^S \bar{Z}, W_V^S \bar{Z}\right) \\
\text{Output} &= Z' + \text{MLP}(\text{LayerNorm}(Z')).
\end{aligned} \tag{4}$$

C Further details on cross-view completion pre-training

C.1 Detailed pre-training settings

We report below the detailed parameter setting we used in our pre-training.

Hyperparameters	Value
Optimizer	AdamW [20]
Base learning rate	1.5e-4
Weight decay	0.05
Adam β	(0.9, 0.95)
Batch size	256
Learning rate scheduler	Cosine decay
Training epochs	400
Warmup learning rate	1e-6
Warmup epochs	40
Masked tokens	90%
Input resolution	224×224
Augmentation	Homography, Color jitter

C.2 Ablation on overlaps between pre-training pairs

For pre-training, we use synthetic pairs generated using the Habitat simulator, keeping pairs with a co-visibility ratio over 0.5. Figure 1 presents some statistics about the distribution of viewpoints considered. Intuitively, the choice of this co-visibility threshold was guided by two important observations: (a) if two images composing a pair overlap too little, the task boils down to auto-completion, therefore we set a threshold on the minimal co-visibility ratio of 0.5 in our main experiments, (b) if two images composing a pair overlap too much, the task becomes trivial, therefore we encourage large viewpoint changes between images in our training set. In this section, we present an ablation to better understand what makes good pre-training pairs.

Formally, we define the visibility ratio $v_{i,j}$ of a view i with respect to an other view j as the ratio of pixels of image i that are visible in image j. We ignore pixels where no geometry is rendered by the Habitat simulator for this computation. We similarly define the co-visibility ratio between two views i and j as $\min(v_{i,j}, v_{j,i})$. To study the influence of co-visibility on pre-training, we generate multiple training sets each composed of $700,000$ image pairs with different co-visibility distributions, and pre-train CroCo on these sets for a number of steps equivalent to 200 epochs of the original dataset, while keeping all other parameters fixed. Figure 2 provides examples of such training pairs. We report in Figure 3 the performance achieved when pre-training with such pairs and then finetuning on Taskonomy tasks. Overall, we observe better performance for most 3D-related tasks (in particular for *curvature*, *depth*, *keypoints3d*, *normal*, *reshading*) when pre-training with pairs with a co-visibility ratio close to 0.5, compared to pre-training with pairs of greater or lower co-visibility (blue curve in Figure 3). We also experimented with pairs chosen to have co-visibility ratios uniformly distributed over different value ranges and report results in Figure 3. We find that pre-training with co-visibility ratios uniformly distributed over $[0, 1.0]$ leads to worse results than when exclusively sampling pairs with a co-visibility ratio within $[0.4, 0.5]$. This confirms that pairs having an intermediate co-visibility ratio are the most suitable for pre-training, although more extensive experiments would be required to strongly support these findings.

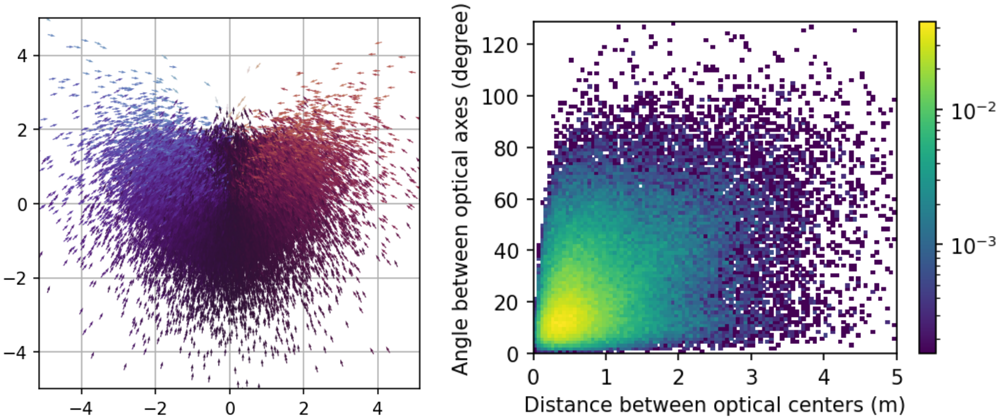
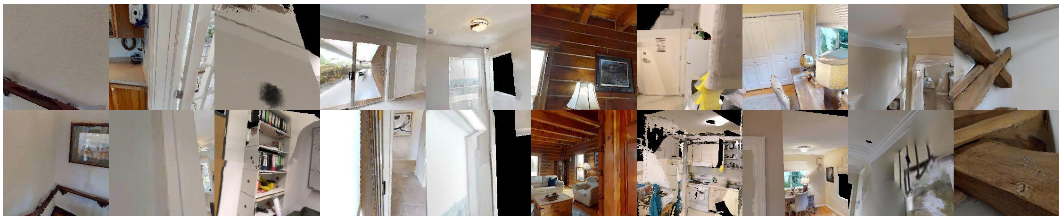
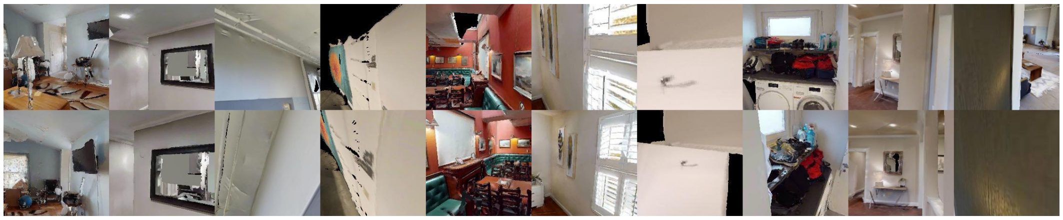


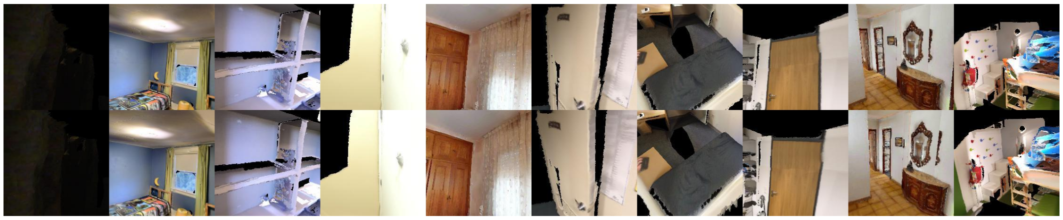
Figure 1: **Distribution of viewpoints pairs used for pre-training.** Left: 2D position (in meters) and orientation (arrow) of one view with respect to the other, once projected on the world horizontal plane. Right: Joint histogram of distances and angles within pairs of views used for training.



Co-visibility ratio between 0 and 0.1



Co-visibility ratio between 0.4 and 0.5



Co-visibility ratio between 0.9 and 1.0

Figure 2: **Some pre-training pair examples for various co-visibility ratios.**

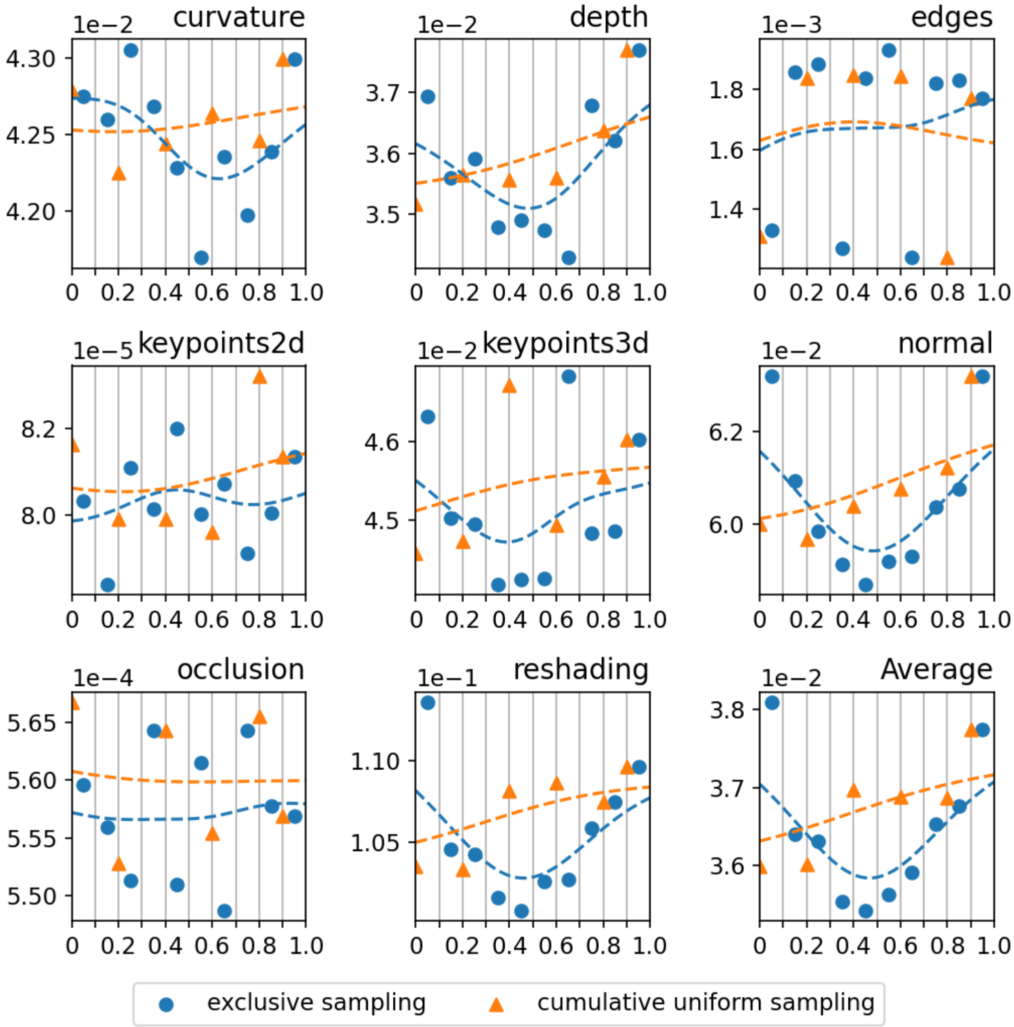


Figure 3: **Ablation on co-visibility between pre-training pairs**. Performance achieved on Taskonomy tasks (vertical axis, lower is better) by a CroCo model pre-trained with image pairs of co-visibility ratio exclusively distributed in the range $[x - 0.05, x + 0.05]$ (blue) or uniformly distributed in the range $[x, 1.0]$ (orange), for various values of x (horizontal axis). Dashed lines represent trend curves obtained by Gaussian smoothing.

C.3 Ablation on decoder depth

In this section, we report an ablation on the decoder depth, *i.e.*, varying the number of decoder blocks for a decoder using *CrossBlock*. The results are shown in Table 1. Note that in the main paper we used a decoder with 8 blocks. We observe that the decoder depth has overall little impact on the performance for monocular tasks (semantic segmentation on ADE, depth prediction on NYUv2 or Taskonomy dense tasks). For the binocular task of optical flow estimation on MPI-Sintel, the error clearly decreases as the depth is increased, showing the importance of a deeper decoder when the decoder is also leveraged. Note however that even with a decoder of depth 2, which means that the network comparing the two images is extremely shallow, a competitive performance can still be reached for optical flow. In particular, the performance with 2 decoder blocks is still largely superior to that of a MAE-pre-trained network having a deep decoder (8 blocks) finetuned in the same conditions.

Table 1: **Impact of decoder depth** with CrossBlock. We used 8 blocks in the decoder in the main paper. Best result per column in bold and second best underlined.

Decoder	ADE ↑	NYUv2 ↑	Taskonomy ↓		MPI-Sintel ↓		FLOPs	Params
Depth	segm.	depth	avg.	rank.	clean	final		
2	38.9	84.6	33.83	2.50	3.59	4.35	39.3G	94M
4	40.2	**86.7**	**32.80**	**1.50**	3.15	3.86	42.9G	103M
6	38.7	83.6	34.14	3.25	3.09	3.79	46.6G	111M
8	**40.6**	85.6	33.00	1.88	**31.00**	**3.60**	50.2G	120M

D Further details and results for the monocular downstream tasks

D.1 Detailed finetuning settings for monocular tasks

We provide in this section details of the finetuning settings used for our monocular tasks experiments. Linear probing settings are summarized below, Note that they correspond exactly to the settings presented by the authors of MAE [14] with the LARS [35] optimizer and a large batch size.

Hyperparameters	Value
Optimizer	LARS [35]
Base learning rate	0.1
Weight decay	0
Optimizer momentum	0.9
Batch size	16,384
Learning rate sched.	Cosine decay
Warmup epochs	10
Training epochs	90
Augmentation	RandomResizeCrop

We further detail the training settings for semantic segmentation on ADE20k, monocular depth estimation on NYUv2 and Taskonomy regression tasks. These settings closely match those presented by the authors of MultiMAE [1].

Hyperparameters	ADE	NYUv2	Taskonomy
Optimizer	AdamW [20]	AdamW [20]	AdamW [20]
Learning rate	1e-4	3e-5	2.5e-4
Layer-wise lr decay 0.75	-	0.75	
Weight decay	0.05	1e-6	2.5e-2
Adam β	(0.9, 0.999)	(0.9, 0.999)	(0.9, 0.999)
Batch size	16	16	32
Learning rate sched.	Cosine decay	Cosine decay	Cosine decay
Training epochs	64	1500	300
Warmup learning rate	1e-6	-	1e-6
Warmup epochs	1	100	5
Input resolution	512×512	256×256	384×384
Augmentation	Large Scale jittering Color jittering	RandomCrop Color jittering	-
Drop path	0.1	0.0	0.1

D.2 Leveraging the decoder for monocular tasks

By default, for dense monocular tasks we append a DPT module [24] to the encoder of our CroCo model; this means that the decoder is discarded when finetuning. We now discuss other possible ways of finetuning our CroCo model on downstream tasks.

- `Using the encoder alone.` The simplest and most lightweight option is to use the encoder alone, without the decoder nor the DPT module, by appending an output linear prediction head, trained from scratch for the downstream task. In Table 2 we see, by comparing rows 1 & 2 and rows 5 & 6, that removing the DPT module results in a significant degradation of performance across all tasks.
- `Decoder instead of the DPT module.` The decoder of our CroCo model can be used instead of the DPT module. While it does not fuse information from several layers at different depths, as the DPT module does, the decoder is trained to output dense predictions, and as such it should be easy to finetune for dense tasks other than RGB predictions. To achieve this we initialize the full CroCo model with pre-trained weights, duplicate the encoded features from the input image to serve as input to the decoder, and simply replace the prediction head of the network by a new one trained from scratch. In Table 2 we observe by comparing rows 2 & 3 as well as 6 & 7 that on most tasks, this leads to a very minor degradation of performance. Furthermore, we found that finetuning the model this way was one order of magnitude faster than training a DPT module: convergence on NYUv2 requires approximately 1500 epochs for models without a decoder, against approximately 200 for models using the decoder.
- `Decoder and DPT module.` The decoder can be used together with the DPT module, by extracting the features to be used as input to the DPT module from the CroCo decoder rather than from the encoder. This increases the cost of running the model. However, in Table 2 we see that this leads to consistent performance gains, by comparing rows 2 & 4, or rows 6 & 8.
- `Frozen backbones.` In all cases, we can choose to freeze the backbone and only train the new prediction layers (DPT module or output prediction head when no DPT module is used). Such an experiment is useful to probe what information the pre-trained model captures. It can also help with over-fitting in cases where very little data is used to train for the downstream task. Empirically, we observe by comparing the lower half of Table 2 to the upper part that freezing the backbone degrades the performance in a majority of cases. On average, however, the performance remains surprisingly high, which demonstrates that frozen output features produced by the CroCo model are already meaningful representations for a wide diversity of 3D vision tasks.

Table 2: **Performance of our CroCo model when finetuned for monocular downstream tasks using different modules**, and possibly freezing the backbone network. Note that in the main paper, performance was reported without the decoder, with DPT and with finetuning of the backbone.

architecture			NYUv2 ↑	Taskonomy ↓								
dec.	DPT	frozen	depth	curv.	depth	edges	kpts2d	kpts3d	normal	occl.	reshad.	avg.
With finetuned backbone												
✗	✗	✗	67.8	45.67	63.56	10.50	0.51	60.20	139.69	0.65	186.6	63.42
✗	✓	✗	86.1	40.91	31.34	1.74	**0.08**	41.69	54.13	**0.55**	93.58	33.00
✓	✗	✗	85.9	45.05	33.93	4.18	0.15	44.90	63.02	**0.55**	103.1	36.86
✓	✓	✗	**88.1**	**39.93**	**30.88**	**1.66**	0.58	**40.87**	**52.26**	0.56	**90.77**	**32.18**
With frozen backbone												
✗	✗	✓	51.3	49.39	69.25	34.76	0.68	60.15	170.10	0.80	198.2	72.92
✗	✓	✓	85.2	42.01	**38.18**	**2.43**	**0.09**	45.50	**64.58**	0.55	117.4	38.85
✓	✗	✓	86.4	44.73	75.97	35.16	0.56	60.54	177.12	0.65	220.0	76.84
✓	✓	✓	**87.1**	**41.57**	41.27	3.64	0.24	**43.49**	**62.77**	**0.54**	**116.9**	**38.81**

D.3 Visualization of monocular depth prediction and Taskonomy results

In Figure 4 we display input images from the NYUv2 dataset (validation set), corresponding depth predictions, alone and overlayed on the input images.

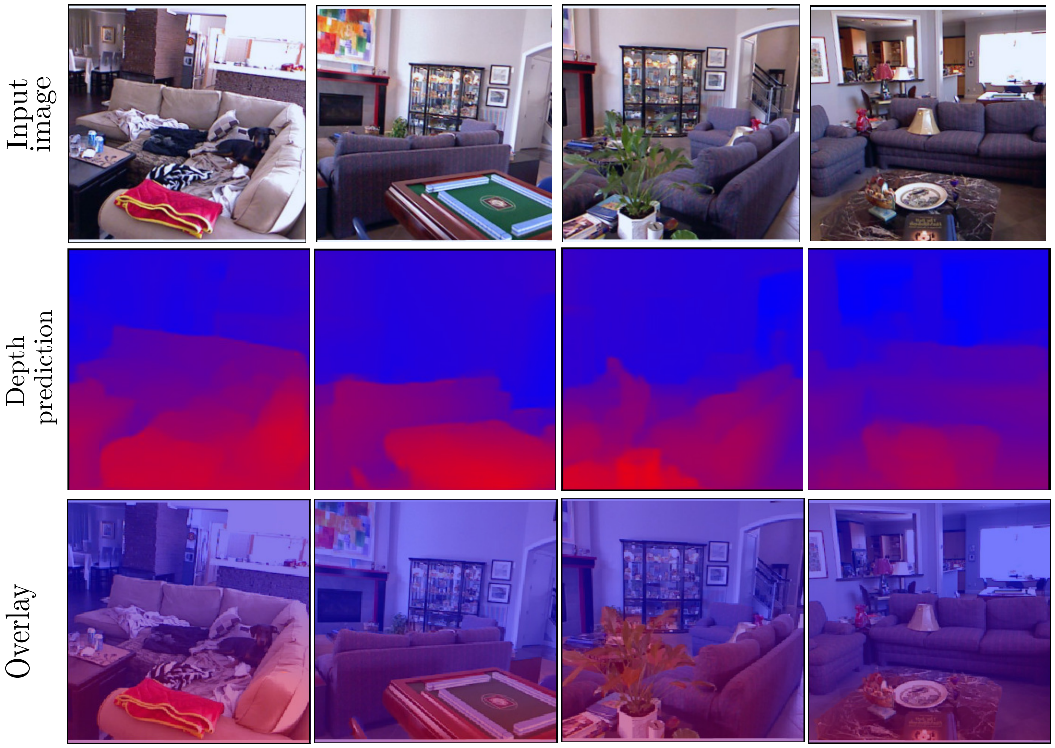


Figure 4: **Qualitative visualization of depth prediction.** First row: the input image; second row: the depth prediction; third row: the depth prediction overlayed over the input image.

Furthermore, in Figure 5 we present results on the Taskonomy dataset for the 8 dense regression tasks.

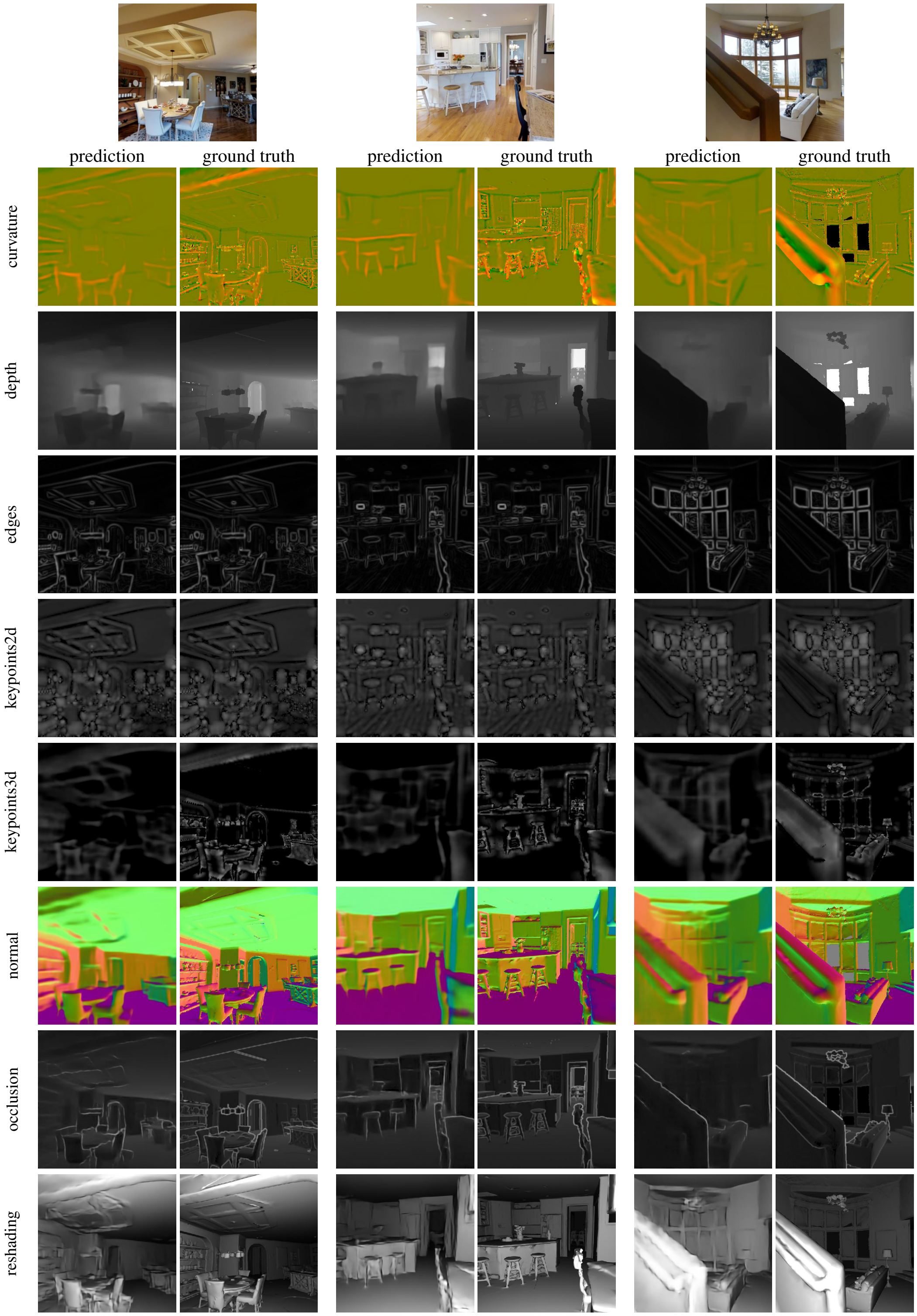


Figure 5: **Example results on Taskonomy.** For the images in the top row, we show the prediction made by our finetuned model and the ground truth, for each of the 8 tasks.

D.4 Application to absolute pose regression from a single image

Task and setting. We apply our pre-trained model to the task of monocular absolute pose regression, where a model directly predicts the absolute camera pose of a given image in a fixed and given environment for which it was specifically trained. In contrast to classical structure-based methods, camera localization methods based on deep neural networks directly regress the absolute pose of a query image [18, 17, 2, 34] and do not require database images or 3D maps at test time. While these methods are in general significantly less accurate than structure-based methods [28, 30, 15, 16], they only require images and their corresponding camera poses as training data.

We use the evaluation code and protocol from AtLoc [34], a recent state-of-the-art improvement of the seminal PoseNet [18] work, which adds an attention module between the encoder output and the pose regressor heads to reweigh the encoded features. The pose regressor is composed of two independent MLP heads: one for predicting the camera pose and one for the camera rotation, represented as a quaternion. In our experiments we append the same regression head (attention module and pose regressor heads) on top of the CroCo encoder, with a global average pooling layer inserted in-between to get a global image representation from the encoder tokens. To finetune the model, we rely on the loss proposed in [2] and also used in [34]:

$$\mathcal{L} = e^{-\beta}\|\boldsymbol{p} - \hat{\boldsymbol{p}}\|_1 + e^{-\gamma}\|\log \boldsymbol{q} - \log \hat{\boldsymbol{q}}\|_1 + \beta + \gamma \tag{5}$$

where β and γ are learned weights that balance the position loss and rotation loss and $\log \boldsymbol{q}$ is the logarithmic form of a unit quaternion (*i.e.*, the corresponding rotation vector). To remove ambiguity between $\boldsymbol{q}$ and $-\boldsymbol{q}$, all quaternions are restricted to the same hemisphere. The finetuning parameters used for the pose regression are the followings:

Hyperparameters	Value
Optimizer	Adam (Base learning rate = 5e-5; weight decay = 5e-4)
Training epochs	500 (with batch size 64)
dropout rate probability	0.5
weights β and γ	initialized with 0 respectively -3
Finetuning dataset	7-Scenes [13]
Input resolution	224×224 (Input field of view 49°)
Augmentation	color jittering and homographies

We benchmark performance on the 7-Scenes dataset [13] on the test split of each scene considered independently. We considered the RGB camera pose annotations from Kapture [16] and the images were rescaled and cropped to a 224×224 resolution with a constant 49° field of view (the largest value fitting in the original images).

Results. We evaluate the impact of the choice of backbone and pre-training method in Table 3 as a function of the size of the training set. The first row corresponds to a ResNet34 backbone, as used in AtLoc, pre-trained with supervision on ImageNet. Other rows correspond to a ViT-Base/16 backbone with different pre-training methods and datasets (MAE, MultiMAE, CroCo). We observe that when using the full training set, all models achieve similar performance, except for the model initialized by MultiMAE which consistently underperforms. Note that CroCo achieves peak validation performance in approx. $50 - 100$ epochs, while ResNet34 pre-trained models need approximately 500 epochs to converge. To better assess the performance of the pre-training model, we significantly reduce the size of the training set and report results in Table 3 using only 5, 10 and 20% of the full training set. We observe that the original AtLoc model, a pre-trained ResNet34 encoder, is unable to learn from such a small amount of data even with data augmentations (random color jittering and homographies simulating camera rotations). Models pre-trained using MAE/MultiMAE perform better, but Croco pre-training leads to the best performance. Finally, while a model trained from scratch (two last rows in Table 3) can achieve decent localization performances using the full training set compared to the other baselines, it generalizes poorly when trained using a more limited amount of data.

Table 3: **Absolute pose regression results** as averaged median errors over the 7 scenes for different backbones, pre-training methods and datasets. Each column corresponds to different ratios of the training set. Best result per column on bold and second best underlined.

Architecture	pre-training	100%	20%	10%	5%
ResNet34	Supervised (ImageNet)	**27.1cm**, 9.0°	34.0cm, 10.9°	43.7cm, 130.°	62.3cm, 17.3°
ViT-Base/16	MAE (ImageNet)	27.9cm, 9.0°	28.0cm, 8.5°	30.6cm, 9.2°	34.4cm, **9.4°**
ViT-Base/16	MAE (Habitat)	28.3cm, 9.1°	28.0cm, 8.3°	30.7cm, 9.1°	35.3cm, 10.1°
ViT-Base/16	MultiMAE (ImageNet)	33.1cm, 9.8°	32.6cm, 9.7°	36.8cm, 10.8°	44.1cm, 12.2°
ViT-Base/16	**CroCo** CrossBlock (Habitat)	27.7cm, **8.6°**	**26.3cm, 7.3°**	**29.0cm, 8.3°**	**32.7cm**, 9.5°
ResNet34	Random	28.1cm, 8.9°	36.6cm, 11.4°	51.3cm, 14.0°	69.3cm, 17.6°
ViT-Base/16	Random	29.1cm, 9.3°	38.3cm, 11.6°	43.6cm, 12.5°	52.7cm, 14.1°

E Further details and results on the binocular downstream tasks

E.1 Optical Flow estimation

Experimental details. We treat optical flow as a straightforward regression task and do not change the pre-training architecture except for modifying the regression head to output two flow channels instead of 3 RGB color channels: the two images are input as such in the Siamese encoders, and the decoder regresses two flow values (u, v) for each pixel using a simple linear head running independently for each output token. We use a *CrossBlock* decoder with 8 layers (similar results are obtained with a *CatBlock* decoder). Training details are provided in Table 4.

Qualitative Results. We show in Figure 6 qualitative visualization of the flow predicted by our method. Overall, we observe that the flow is correctly estimated, even in the case of varied and fast motion. As a limitation, the estimation gets slightly inaccurate when occlusions happen, and blurry on extremely fine-grained motion such as hair tips, as for most methods. Again, we wish to emphasize that the model was finetuned only on 40,000 images without any sort of elaborated data augmentation.

E.2 Relative pose regression

Model Architecture. We replace the original head of the decoder pre-trained with CroCo by a simple head regressing a relative pose $(\boldsymbol{R}, \boldsymbol{t}) \in SO(3) \times \mathbb{R}^3$. This head consists of the following layers. First, a linear projection reduces the dimension of tokens produced by the decoder to 64, in order to limit the computation costs. Second, these tokens are flattened into a unique feature vector, which is

Table 4: **Parameter settings for optical flow, relative pose regression and stereo matching.**

Hyperparameters	Optical Flow	Relative Pose Regression	Stereo Matching
Optimizer	AdamW [20]	AdamW [20]	AdamW [20]
Base learning rate	3e-5	1e-7	1e-4
Weight Decay	0.05	-	-
Batch size	20	64	8
Adam β	(0.9, 0.95)	(0.9, 0.95)	(0.9, 0.95)
Learning rate sched.	Cosine decay	Cosine decay	1 cycle
Finetuning epochs	100	30	400
Linear warmup epochs	1	1	1
Translation weight λ	-	100m	-
Finetuning dataset	AutoFlow [32]	7-scenes [13]	VKITTI [5]
Input resolution	224×224	224×224	1242×375
Augmentation	random crop, color jitter	homography, color jitter	random horizontal flip, color jitter
Input field of view	-	49°	-

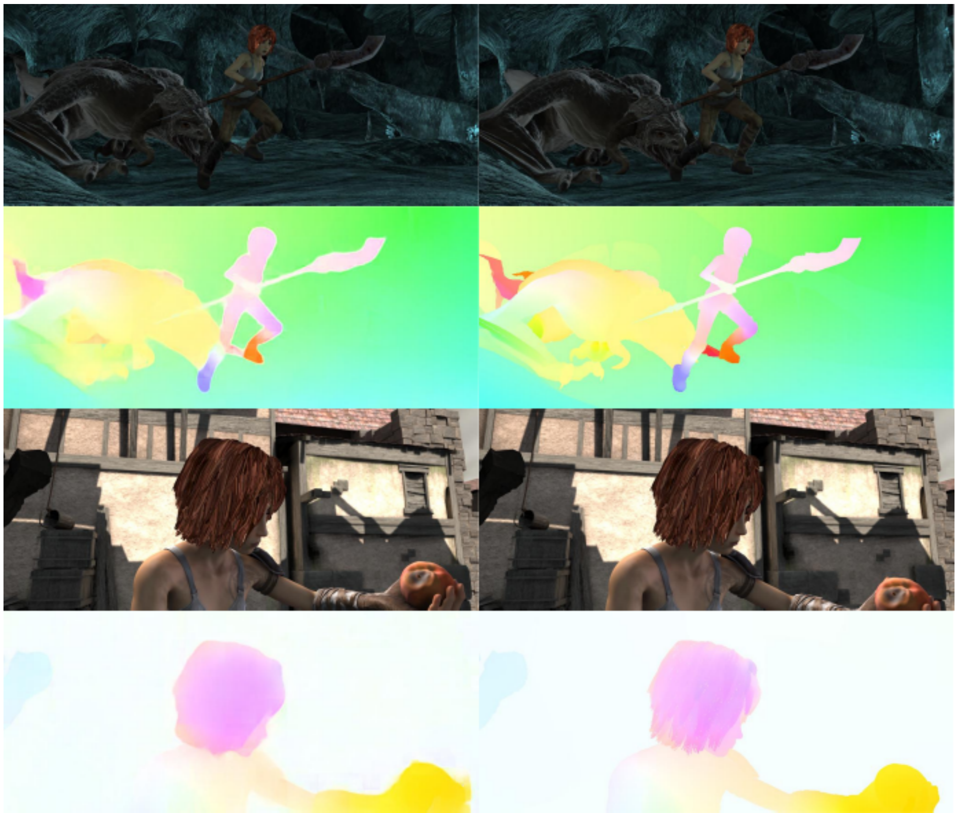


Figure 6: **Qualitative visualization of the estimated flow** on some examples from the MPI-Sintel training set, unseen during training. First and third rows: input image pair, second and fourth row: predicted (*left*) and ground-truth (*right*) flows.

processed by a MLP (with a hidden layer of size 1024 and ReLU activations) to regress a 12D output. It is reshaped into an affine transformation $(\boldsymbol{M}, \boldsymbol{t}) \in \mathbb{R}^{3\times 3} \times \mathbb{R}^3$. Lastly we use a differentiable special Procrustes layer [3] to orthonormalize $\boldsymbol{M}$ into a rotation matrix $\boldsymbol{R}$ and to predict the relative pose $(\boldsymbol{R}, \boldsymbol{t})$.

Experimental details. The RGB camera pose annotations for the 7-scenes dataset [13] are obtained from Kapture [16]. To select training pairs, we extract global AP-GeM-18 descriptors [25] for every image, and match each training image with its 20 closest neighbors according to their descriptor similarity. Our training set is composed of 26k images and 520k training pairs in total. We rescale and crop the images to a square 224 × 224 resolution with a constant 49° field of view (the largest value fitting in the original images), and we augment the training image pairs with random permutations, color jittering and homographies that simulate camera rotation.

In our experiments, we trained models using the AdamW [20] optimizer using settings described in Table 4. We found the training to be quite sensitive to its initialization, and we had to try multiple random seeds to achieve decent performances on the training set when training the decoder from scratch with an encoder pre-trained with MAE on Habitat. Using an encoder pre-trained with MAE on ImageNet furthermore always led to poor performance on both the training and test sets in our experiments. We had none of these issues however when using an encoder and decoder pre-trained using CroCo.

E.3 Stereo image matching

Task and settings. In this section, we experiment on the binocular downstream task of stereo matching [22]. We finetune a pre-trained CroCo model to predict pixel-wise disparity values, by simply replacing the linear prediction head. The model takes two rectified images as input and predicts the disparity of every pixel by matching corresponding pixels in the images. We use a

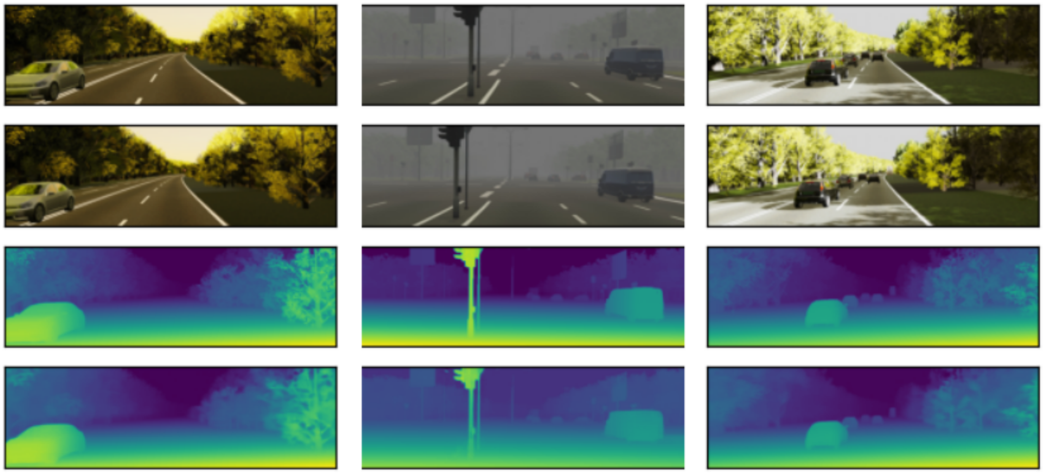


Figure 7: **Qualitative visualization of the estimated disparity** on examples from the VKITTI validation set. First row: input left image; second row: input right image, third row: ground-truth (left) disparity, forth row: predicted (left) disparity. The three examples come from “sunset” (left column), “fog” (center) and “clone” (right column) weather conditions.

MSE-log loss for finetuning and report the error using the 3-pixel 5% discrepancy [7, 8]. We evaluate the stereo matching downstream task on the Virtual KITTI dataset [5] which consists of 5 synthetic sequences cloned from the KITTI tracking benchmark [12]. The dataset additionally provides 9 variants of these 5 sequences under different weather conditions (*e.g.* ‘fog’, ‘rain’, *etc.*) and modified camera configurations (*e.g.* rotation to left or right by 15 or 30 degrees). We downscale VKITTI images to a resolution of 224×742, preserving the aspect ratio and crop to 224×736. Additional details on the finetuning hyper-parameters and settings are described in Table 4.

Results. In Table 5 we report results obtained on all 10 weather condition and camera orientation variants in the VKITTI dataset. We compare results obtained when finetuning the CroCo model and the MAE model with a randomly initialized decoder, pre-trained on Habitat. We observe that CroCo pre-training leads to significantly better results for all scenarios. Next, we compare to the state-of-the-art methods for stereo matching, in particular, PSMNet [7], LaC-GweNet [19] and LEAStereo [8]. Finetuning a model pre-trained with CroCo achieves results competitive with the state of the art, without any task-specific model design, such as spatial pyramid pooling [7], hierarchical neural search in 4D feature volume [8] or local similarity patterns for explicit neighbor relationships [19]. We point out that, in contrast to the other methods, our model is directly finetuned on VKITTI without pre-training on the large SceneFlow dataset [21]. In Figure 7 we visualize the disparity prediction by our method, for three different conditions in the VKITTI dataset.

Table 5: **Stereo matching results** with the average 3-px error for 10 VKITTI variants. We compare CroCo pre-training with MAE pre-training as well as other state-of-the-art methods. Best result per column on bold and second best underlined.

Method/ Pre-training	fog	sun-set	clone	over-cast	rain	mor-ning	15°-left	15°-right	30°-left	30°-right	Ave-rage
MAE (Habitat)	1.80	1.75	1.96	1.89	2.13	2.17	2.93	2.07	4.06	2.25	2.30
CroCo (Habitat)	**1.15**	1.69	1.72	1.49	1.77	1.68	2.49	1.58	3.43	1.78	1.88
PSMNet [7]	2.36	2.30	2.38	2.45	2.31	2.39	2.36	2.41	2.33	2.25	2.36
LaC-GwcN [19]	1.67	**1.41**	1.52	1.39	1.30	1.72	1.65	**1.51**	1.50	1.79	1.54
LEAStereo [8]	1.24	1.42	**1.27**	**1.17**	**1.05**	**1.09**	**1.18**	1.65	**1.06**	**1.22**	**1.23**

F Compute resources, code and dataset assets

F.1 Floating point operations (FLOPs)

We calculate the number of floating point operations (or FLOPs) in the most commonly accepted manner, *i.e.*, counting additions and multiplications as separate floating-point operations. To that aim, we borrow and slightly adapt the `flops_computation.py` code from [9]. Note that most of the FLOPs comes from matrix-matrix multiplication operations in the attention and feed-forward layers. For a standard ViT (encoder only), the number of FLOPs can thus be approximated as

$$\#FLOPs \simeq 2L\left(F_{attn-kqv} + F_{attn-scores} + F_{attn-avg} + F_{attn-out} + F_{ff-in} + F_{ff-out}\right)$$

where

$$\begin{aligned} F_{attn-kqv} &= 3ND^2 \\ F_{attn-scores} &= DN^2 \\ F_{attn-avg} &= DN^2 \\ F_{attn-out} &= ND^2 \\ F_{ff-in} &= 4ND^2 \\ F_{ff-out} &= 4ND^2. \end{aligned}$$

For instance, for ViT-Base/16, setting the number of blocks $L = 12$, the number of tokens $N = 14^2$ and the embedding dimension $D = 768$, we find 34.7 GFLOPs with the formula above, which is close to the 35.3 GFLOPs calculated including all other operations (layer norms, patch embeddings, *etc.*). Note that we report exact theoretical FLOPs (counting all operations) in the paper.

F.2 Compute resources

Pre-training a CroCo model for 400 epochs takes about 64 GPU-days on NVIDIA V100. For instance on a 4-GPU server, this is about two weeks.

F.3 Assets used in this submission

We provide below an overview of assets used in our experiments and their licenses.

Asset	License
Habitat pre-training	
HM3D [23]	academic, non-commercial research [hyperlink]
ScanNet [10]	non-commercial research and education [hyperlink]
Replica [31]	non-commercial research and education [hyperlink]
ReplicaCAD [33]	Creative Commons Attribution 4.0 International (CC BY 4.0) [hyperlink]
Habitat simulator [27]	MIT [hyperlink]
High-level semantic tasks	
ImageNet-1K [26]	non-commercial research and education [hyperlink]
ADE [37]	images: non-commercial research and education – annotations: Creative Commons BSD-3 [hyperlink]
Monocular 3D vision tasks	
NYUv2 [29]	public [hyperlink]
Taskonomy [36]	non-commercial research and education [hyperlink]
Optical flow	
AutoFlow [32]	Create Commons Attribution 4.0 International (CC BY 4.0) [hyperlink]
MPI-Sintel [4]	images: Creative Commons Attribution 3.0 (CC BY 3.0) – copyright Blender Foundation | www.sintel.org [hyperlink]
Absolute / relative pose regression	
7-scenes [13]	non-commercial [hyperlink]
Kapture package [16]	BSD 3-Clause Revised [hyperlink]
Stereo matching	
Virtual KITTI [5]	Creative Commons Attribution-NonCommercial-ShareAlike 3.0 [hyperlink]
Code and pre-trained models	
MAE [14]	Creative Commons Attribution-NonCommercial 4.0 International (CC BY NC 4.0) [hyperlink]
MultiMAE [1]	Creative Commons Attribution-NonCommercial 4.0 International (CC BY NC 4.0) [hyperlink]
DINO [6]	Apache License 2.0 [hyperlink]

G Further visual examples

G.1 Video of sample CroCo reconstructions

We attach a video of reconstruction results obtained using the proposed CroCo model on some validation scenes from the HM3D dataset [23], unseen at training time (see Figure 8). For each scene, we consider a constant reference input image and try to reconstruct a sequence of target images from masked inputs, each frame being processed independently using CroCo. The capability of CroCo to solve this challenging task suggests that it is somehow able to capture the 3D layout of the scene, as well as the relative pose of the masked view with respect to the reference view.

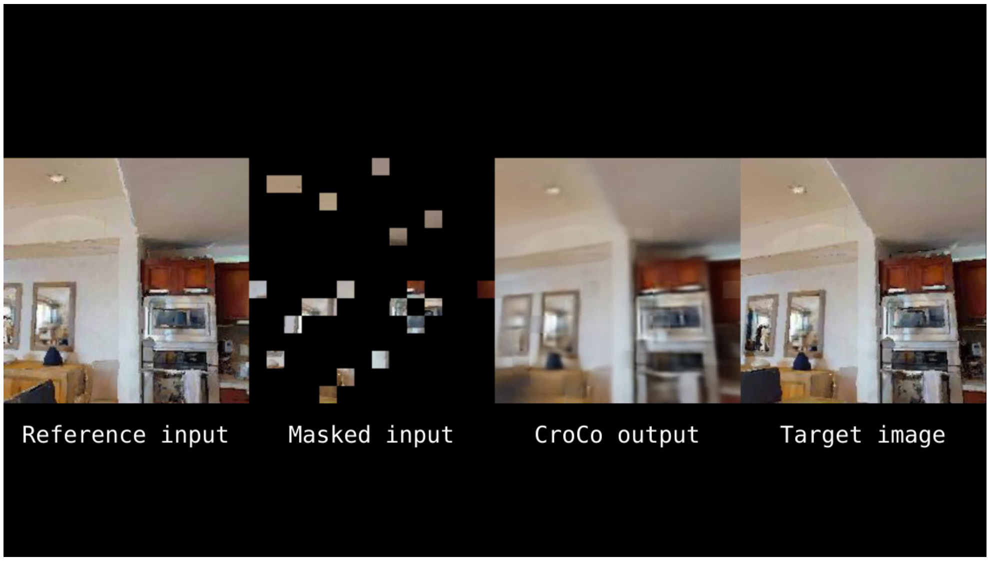


Figure 8: **Excerpt from the supplementary video.** Reconstructions are displayed while varying the viewpoint, with a fixed reference input.

G.2 Additional reconstruction examples

We provide additional reconstruction examples in Figure 9, where we compare reconstructions obtained using our CroCo model pre-trained with an RGB loss and a reference input image (*CroCo RGB*) with reconstructions obtained after replacing this reference image by an image of random uniform noise, independent at each pixel (*CroCo RGB random noise ref.*). Cross-view completion enables a decent reconstruction of the masked image in general, except for areas non-visible in the reference image (*e.g*. *CroCo RGB*, left part of the painting in the third column). When replacing the reference image by some random noise, the reconstruction problem becomes similar to inpainting and some parts of the scene may be wrongly reconstructed due to the lack of available information (*e.g*. wrong size for the window in the first column, missing ice maker in the fridge in the second column).

We also compare reconstructions obtained with CroCo and MAE models pre-trained on Habitat to reconstruct normalized patches (*CroCo norm* and *MAE norm*, last two rows). We use patch statistics from the target image to un-normalize these reconstructions for visualization purposes, which explains that the mean color of each reconstructed patch is relatively well reconstructed in all cases. Yet, we observe that CroCo produces more detailed reconstructions than MAE, the latter often appearing quite tessellated due to an inconsistent reconstruction of normalized patches.

References

[1] Roman Bachmann, David Mizrahi, Andrei Atanov, and Amir Zamir. MultiMAE: Multi-modal Multi-task Masked Autoencoders. In *ECCV*, 2022.

[2] Samarth Brahmbhatt, Jinwei Gu, Kihwan Kim, James Hays, and Jan Kautz. Geometry-Aware Learning of Maps for Camera Localization. In *CVPR*, 2018.

[3] Romain Brégier. Deep regression on manifolds: a 3D rotation case study. In *3DV*, 2021.

[4] Daniel J Butler, Jonas Wulff, Garrett B Stanley, and Michael J Black. A naturalistic open source movie for optical flow evaluation. In *ECCV*, 2012.

[5] Yohann Cabon, Naila Murray, and Martin Humenberger. Virtual kitti 2. *arXiv preprint arXiv:2001.10773*, 2020.

[6] Mathilde Caron, Hugo Touvron, Ishan Misra, Hervé Jégou, Julien Mairal, Piotr Bojanowski, and Armand Joulin. Emerging Properties in Self-Supervised Vision Transformers. In *ICCV*, 2021.

[7] Jia-Ren Chang and Yong-Sheng Chen. Pyramid stereo matching network. In *CVPR*, 2018.

[8] Xuelian Cheng, Yiran Zhong, Mehrtash Harandi, Yuchao Dai, Xiaojun Chang, Hongdong Li, Tom Drummond, and Zongyuan Ge. Hierarchical neural architecture search for deep stereo matching. In *NeurIPS*, 2020.

[9] Kevin Clark, Minh-Thang Luong, Quoc V. Le, and Christopher D. Manning. ELECTRA: Pre-training text encoders as discriminators rather than generators. In *ICLR*, 2020.

[10] Angela Dai, Angel X. Chang, Manolis Savva, Maciej Halber, Thomas Funkhouser, and Matthias Nießner. ScanNet: Richly-annotated 3D Reconstructions of Indoor Scenes. In *CVPR*, 2017.

[11] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. In *ICLR*, 2021.

[12] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *CVPR*, 2012.

[13] Ben Glocker, Shahram Izadi, Jamie Shotton, and Antonio Criminisi. Real-time RGB-D camera relocalization. In *ISMAR*, 2013.

[14] Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, and Ross Girshick. Masked Autoencoders are Scalable Vision Learners. In *CVPR*, 2022.

[15] Jared Heinly, Johannes L. Schönberger, Enrique Dunn, and Jan-Michael Frahm. Reconstructing the World in Six Days as Captured by the Yahoo 100 Million Image Dataset. In *CVPR*, 2015.

[16] Martin Humenberger, Yohann Cabon, Nicolas Guerin, Julien Morat, Jérôme Revaud, Philippe Rerole, Noé Pion, Cesar de Souza, Vincent Leroy, and Gabriela Csurka. Robust Image Retrieval-based Visual Localization using Kapture. *arXiv preprint arXiv:2007.13867*, 2020.

[17] Alex Kendall and Roberto Cipolla. Geometric Loss Functions for Camera Pose Regression with Deep Learning. In *CVPR*, 2017.

[18] Alex Kendall, Matthew Grimes, and Roberto Cipolla. PoseNet: a Convolutional Network for Real-Time 6-DOF Camera Relocalization. In *ICCV*, 2015.

[19] Biyang Liu, Huimin Yu, and Yangqi Long. Local similarity pattern and cost self-reassembling for deep stereo matching networks. In *AAAI*, 2022.

[20] Ilya Loshchilov and Frank Hutter. Decoupled Weight Decay Regularization. In *ICLR*, 2019.

[21] Nikolaus Mayer, Eddy Ilg, Philip Häusser, Philipp Fischer, Daniel Cremers, Alexey Dosovitskiy, and Thomas Brox. A large dataset to train convolutional networks for disparity, optical flow, and scene flow estimation. In *CVPR*, 2016.

[22] Matteo Poggi, Fabio Tosi, Konstantinos Batsos, Philippos Mordohai, and Stefano Mattoccia. On the synergies between machine learning and stereo: a survey. *IEEE Trans. PAMI*, 2021.

[23] Santhosh Kumar Ramakrishnan, Aaron Gokaslan, Erik Wijmans, Oleksandr Maksymets, Alexander Clegg, John M Turner, Eric Undersander, Wojciech Galuba, Andrew Westbury, Angel X Chang, Manolis Savva, Yili Zhao, and Dhruv Batra. Habitat-Matterport 3D Dataset (HM3D): 1000 Large-scale 3D Environments for Embodied AI. In *NeurIPS datasets and benchmarks*, 2021.

[24] René Ranftl, Alexey Bochkovskiy, and Vladlen Koltun. Vision transformers for dense prediction. In *ICCV*, 2021.

[25] Jerome Revaud, Jon Almazan, Rafael Sampaio de Rezende, and Cesar Roberto de Souza. Learning with Average Precision: Training Image Retrieval with a Listwise Loss. In *ICCV*, 2019.

[26] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *IJCV*, 2015.

[27] Manolis Savva, Abhishek Kadian, Oleksandr Maksymets, Yili Zhao, Erik Wijmans, Bhavana Jain, Julian Straub, Jia Liu, Vladlen Koltun, Jitendra Malik, Devi Parikh, and Dhruv Batra. Habitat: A Platform for Embodied AI Research. In *ICCV*, 2019.

[28] Johannes L. Schönberger and Jan-Michael Frahm. Structure-from-motion Revisited. In *CVPR*, 2016.

[29] Nathan Silberman, Derek Hoiem, Pushmeet Kohli, and Rob Fergus. Indoor segmentation and support inference from RGBD images. In *ECCV*, 2012.

[30] Noah Snavely, Steven M. Seitz, and Richard Szeliski. Modeling the World from Internet Photo Collections. *IJCV*, 2008.

[31] Julian Straub, Thomas Whelan, Lingni Ma, Yufan Chen, Erik Wijmans, Simon Green, Jakob J. Engel, Raul Mur-Artal, Carl Ren, Shobhit Verma, Anton Clarkson, Mingfei Yan, Brian Budge, Yajie Yan, Xiaqing Pan, June Yon, Yuyang Zou, Kimberly Leon, Nigel Carter, Jesus Briales, Tyler Gillingham, Elias Mueggler, Luis Pesqueira, Manolis Savva, Dhruv Batra, Hauke M. Strasdat, Renzo De Nardi, Michael Goesele, Steven Lovegrove, and Richard Newcombe. The Replica dataset: A digital replica of indoor spaces. *arXiv preprint arXiv:1906.05797*, 2019.

[32] Deqing Sun, Daniel Vlasic, Charles Herrmann, Varun Jampani, Michael Krainin, Huiwen Chang, Ramin Zabih, William T Freeman, and Ce Liu. AutoFlow: Learning a better training set for optical flow. In *CVPR*, 2021.

[33] Andrew Szot, Alex Clegg, Eric Undersander, Erik Wijmans, Yili Zhao, John Turner, Noah Maestre, Mustafa Mukadam, Devendra Chaplot, Oleksandr Maksymets, Aaron Gokaslan, Vladimir Vondrus, Sameer Dharur, Franziska Meier, Wojciech Galuba, Angel Chang, Zsolt Kira, Vladlen Koltun, Jitendra Malik, Manolis Savva, and Dhruv Batra. Habitat 2.0: Training Home Assistants to Rearrange their Habitat. In *NeurIPS*, 2021.

[34] Bing Wang, Changhao Chen, Chris Xiaoxuan Lu, Peijun Zhao, Niki Trigoni, and Andrew Markham. AtLoc: Attention Guided Camera Localization. In *AAAI*, 2020.

[35] Yang You, Igor Gitman, and Boris Ginsburg. Large batch training of convolutional networks. *arXiv preprint arXiv:1708.03888*, 2017.

[36] Amir Zamir, Alexander Sax, William Shen, Leonidas Guibas, Jitendra Malik, and Silvio Savarese. Taskonomy: Disentangling task transfer learning. In *CVPR*, 2018.

[37] Bolei Zhou, Hang Zhao, Xavier Puig, Sanja Fidler, Adela Barriuso, and Antonio Torralba. Scene parsing through ADE20K Dataset. In *CVPR*, 2017.

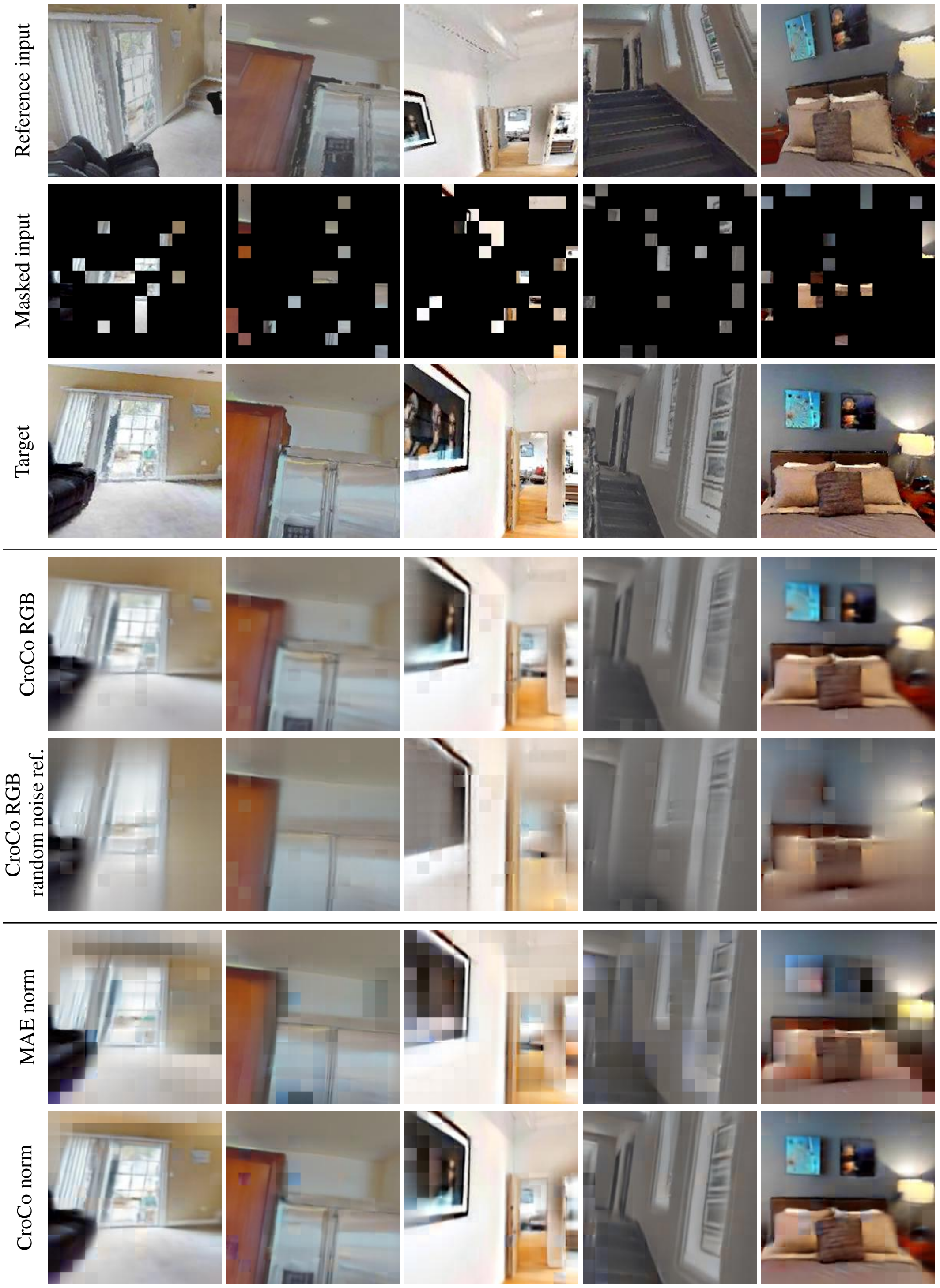


Figure 9: **Additional reconstruction examples** for scenes unseen during training. We compare image reconstructions obtained using our CroCo model pre-trained with an RGB loss using the reference input image (*CroCo RGB*) or replacing it by random noise (*CroCo RGB random noise ref.*). We also compare reconstructions of CroCo and MAE after a pre-training on Habitat to reconstruct normalized patches (*CroCo norm* and *MAE norm*). Patch un-normalization is performed using patch statistics from the target image, for visualization purposes.