InstaFormer: Instance-Aware Image-to-Image Translation with Transformer -Supplementary Material-

In this document, we describe detailed network architecture, PyTorch-like pseudo-code, and additional results for "InstaFormer: Instance-Aware Image-to-Image Translation with Transformer".

A. Experimental Details

A.1. Network Architecture of InstaFormer

We first summarize the detailed network architecture of our InstaFormer in Table 1. We basically follow the content encoder and generator architecture from CUT [5] with Transformer blocks and style encoder. The double-line inside Encoder table indicates the end of content encoder, while style encoder has same structure with content encoder but has additional Adaptive Avg. Pool. and Conv-4 as shown below. $(.)$ in the convolution indicates the padding technique, and the structures of Upsample , Downsample and discriminator are the same as those of CUT [5].

Encoder

Layer	Parameters $(\texttt{in}, \texttt{out}, \texttt{k}, \texttt{s}, \texttt{p})$	Output shape $(C \times H \times W)$
Conv-1 (Reflection)	$(3, 64, 7, 1, 3)$	$(64, 352, 352)$
InstanceNorm	-	$(64, 352, 352)$
ReLU	-	$(64, 352, 352)$
Conv-2 (Zeros)	$(64, 128, 3, 1, 1)$	$(128, 352, 352)$
InstanceNorm	-	$(128, 352, 352)$
ReLU	-	$(128, 352, 352)$
Downsample	-	$(128, 176, 176)$
Conv-3 (Zeros)	$(128, 256, 3, 1, 1)$	$(256, 176, 176)$
InstanceNorm	-	$(256, 176, 176)$
ReLU	-	$(256, 176, 176)$
DownSample	-	$(256, 88, 88)$
AdaptiveAvgPool	-	$(256, 1, 1)$
Conv-4	$(256, 8, 1, 1, 0)$	$(8, 1, 1)$

Transformer Aggregator

Layer	Parameters $(\texttt{in}, \texttt{out})$	Output shape (C)
AdaptiveInstanceNorm	-	(1024)
Linear-1	$(1024, 3072)$	(3072)
Attention	-	(1024)
Linear-2	$(1024, 1024)$	(1024)
AdaptiveInstanceNorm	-	(1024)
Linear-3	$(1024, 4096)$	(4096)
GELU	-	(4096)
Linear-4	$(4096, 1024)$	(1024)

Generator

Layer	Parameters $(\texttt{in}, \texttt{out}, \texttt{k}, \texttt{s}, \texttt{p})$	Output shape $(C \times H \times W)$
UpSample	-	$(256, 176, 176)$
Conv-1 (Zeros)	$(256, 128, 3, 1, 1)$	$(128, 176, 176)$
InstanceNorm	-	$(128, 176, 176)$
ReLU	-	$(128, 176, 176)$
UpSample	-	$(128, 352, 352)$
Conv-2 (Zeros)	$(128, 64, 3, 1, 1)$	$(64, 352, 352)$
InstanceNorm	-	$(64, 352, 352)$
ReLU	-	$(64, 352, 352)$
Conv-3 (ReflectionPad)	$(64, 3, 7, 1, 3)$	$(3, 352, 352)$
Tanh	-	$(3, 352, 352)$

Table 1. **Network architecture of our InstaFormer.**

A.2. PyTorch-like Pseudo-code

Pseudo-code for Instance-level Content Loss. Here we provide the (PyTorch-like) pseudo-code for $\mathcal{L}_{\text{NCE}}^{\text{ins}}$ in InstaFormer. To re-emphasize, our simple instance-level content loss learns the representations for a translation task, effectively focusing on local object-region; this is unlike classical PatchNCE loss that utilizes the regular-grid patches from features in unconditional way.

```
cross_entropy_loss = CrossEntropyLoss()

# Input: f_q (BxCxS) and sampled features from MLP(c^ins_i),
# where c^ins_i is from E(x)
# Input: f_k (BxCxS) are sampled features from MLP(hat(c^ins_i)),
# where hat(c^ins_i) is from E(hat(y^ins_i))
# Input: tau is the temperature used in NCE loss.
# Output: InstNCELoss loss
def InstNCELoss(f_q, f_k, tau=0.07):
    # batch size, channel size, and number of sample locations
    B, C, S = f_q.shape

    # calculate v * v+: BxSx1
    l_pos = (f_k * f_q).sum(dim=1)[:, :, None]

    # calculate v * v-: BxSxS
    l_neg = bmm(f_q.transpose(1, 2), f_k)

    # The diagonal entries are not negatives. Remove them.
    identity_matrix = eye(S)[None, :, :]
    l_neg.masked_fill_(identity_matrix, -float('inf'))

    # calculate logits: (B)x(S)x(S+1)
    logits = cat((l_pos, l_neg), dim=2) / tau

    # return NCE loss
    predictions = logits.flatten(0, 1)
    targets = zeros(B * S)
    return cross_entropy_loss(predictions, targets)
```

Pseudo-code for Transformer Aggregator. We also provide pseudo-code for the input of $\mathcal{T}$, in order to show how our novel technique aggregates *instance*-level content features and *global*-level content features simultaneously. This enables the model to pay more attention to the relationships between global scenes and object instances.

```
# Content_out: output feature from content encoder
# box_info: box information dimension: batch_size, the number of box, box_information[class, left, top, right, bottom])
# num_box: the number of box
# patch_embed: patch embedding function
# box_patch_embed: box patch embedding function (same as patch embedding)
# roi align follow Mask RCNN

def Box_feature_extrcat(feature, box_info):
    B, C, H, W = feature.shape
    batch_index = arange(0.0, B).repeat(num_box).view(num_box, -1).transpose(0,1).flatten(0,1) # (#Box  x B)

    # (B x #Box), box_info
    box_info = box_info.view(-1,5)
    roi_info = stack((batch_index, box_info[:, 1] * W, box_info[:, 2] * H, \
                                   box_info[:, 3] * W,  box_info[:, 4] * H), dim = 1)
    roi_feature = roi_align(feature, roi_info, patch_size)  # (B x #Box), C, patch_size, patch_size

    return roi_feature

def add_box_feature(self, embed_feature, roi_featrue):
    B = embed_feature.shape[0]
    embed_box_feature = box_patch_embed(roi_featrue).squeeze().view(B, num_box, -1) # B, #Box, embed_C
    aggregator_input = cat((embed_feature, embed_box_feature), dim=1)               # B, (#Patch + #Box), embed_C

    return aggregator_input

# B,C,H,W & B,#Box,box_info > (B x #Box), C, patch_size, patch_size
roi_feature = Box_feature_extrcat(content_out, box_info)
# B,C,H,W > B, #Patch, embed_C
embed_feature = patch_embed(content_out)
# B, (#Patch + #Box), embed_C
aggregator_input = add_box_feature(embed_feature, roi_feature)
```

B. Additional Results

B.1. Visualization of Multi-modal Image Translation

We visualize the multimodal translated results in Fig. 1. Our InstaFormer generates not only high-quality visual results, but also produces results with large diversity.

(a) Input image (Sunny) (b) Translated image 1 (Night) (c) Translated image 2 (Night) (d) Translated image 3 (Night)

Figure 1. **Results of multi-modal image translation.** We use randomly sampled style codes to generate night images from a sunny image.

B.2. Qualitative Results of Domain Adaptive Object Detection

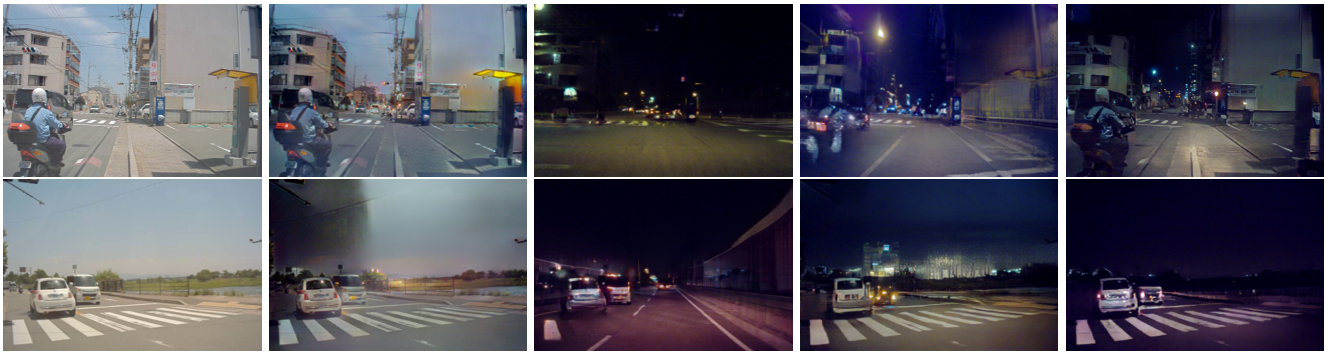
In the main paper, we have evaluated our method on the task of unsupervised domain adaptation for object detection providing quantitative results. In this section, we also show the qualitative results of the task in Fig. 2. Our model successfully works on complex domain adaptation tasks.

(a) Input image (KITTI) (b) Translated image (CityScape)

Figure 2. **Qualitative results for domain adaptive detection for KITTI → CityScape.**

B.3. Comparison with the State-of-the-Art I2I methods.

As shown in Fig. 3 and Table 2, we additionally compare InstaFormer with SoTA I2I methods, such as TSIT [3], StarGANv2 [1] and Smoothing [4] on INIT dataset [6] for sunny→night.



(a) Input (b) TSIT (c) StarGAN2 (d) Smoothing (e) Ours

Figure 3. **Comparison with TSIT [3], StarGANv2 [1] and Smoothing [4].**

Methods	sunny→night	
	FID↓	SSIM↑
TSIT	90.28	0.822
StarGANv2	88.49	0.545
Smoothing	85.28	0.667
InstaFormer	**84.72**	**0.872**

Table 2. **More quantitative evaluation.**

B.4. Additional Examples on Ablation Study

In the main paper, we have examined the impacts of instance-level loss ($\mathcal{L}^{\text{ins}}_{\text{NCE}}$), Transformer encoder ($\mathcal{T}$), normalization, and another backbone (MLP-Mixer). CUT equals to the setting w/o $\mathcal{L}^{\text{ins}}_{\text{NCE}}$, $\mathcal{T}$, and AdaIN. We provide more examples on INIT dataset [6], depicted in Fig. 4. Our InstaFormer produces better visual results. In particular, as shown in Fig. 4 (d), tiny objects tend to disappear or be blurred without $\mathcal{L}^{\text{ins}}_{\text{NCE}}$.

(a) Content image (b) InstaFormer (c) MLP-Mixer [7] (d) w/o $\mathcal{L}^{\text{ins}}_{\text{NCE}}$ (e) w/o $\mathcal{L}^{\text{ins}}_{\text{NCE}}$, $\mathcal{T}$ (f) CUT [5] (g) w/o AdaIN

Figure 4. **Ablation study on different settings:** instance-level loss ($\mathcal{L}^{\text{ins}}_{\text{NCE}}$), Transformer encoder ($\mathcal{T}$), normalization, and another backbone (MLP-Mixer). Note that CUT equals to the setting w/o $\mathcal{L}^{\text{ins}}_{\text{NCE}}$, $\mathcal{T}$, and AdaIN.

B.5. Additional Translation Results

In the main paper, we have shown some of our results of instance-aware image-to-image translation. Here, we show additional results in Fig. 5 to demonstrate the robustness of InstaFormer.

(a) Input image (Sunny) (b) Translated image (Night) (c) Translated image (Cloudy) (d) Translated image (Rainy)

Figure 5. **Qualitative comparison on INIT dataset [6]:** (left to right) sunny, sunny→night, sunny→cloudy, and sunny→rainy results. Our method achieves high-quality of realistic results while preserving object details as well.

B.6. Variants of InstaFormer Architecture

Effects of the Number of Transformer Blocks. We show qualitative and quantitative comparisons of the number of Transformers blocks. As described in Table 3 and Fig. 6, the results using 6 and 9 blocks show almost close FID metric [2] and SSIM metric [8] scores and visual results, while the score of 3 blocks shows insufficient result. As smaller models have a lower parameter count, and a faster throughput, we choose 6 blocks for our architecture.

Effects of the Number of Heads. We analyze the effects of the number of heads in our model. We show the quantitative results in Table 3 and visualization of self-attention maps in Fig. 7 according to the number of heads. More heads tend to bring lower FID metric score and higher SSIM metric score with better self-attention learning, but the scores using 8 heads shows slight better results compared the scores using 4 heads. Thus, we decide to use 4 heads considering memory-efficiency.

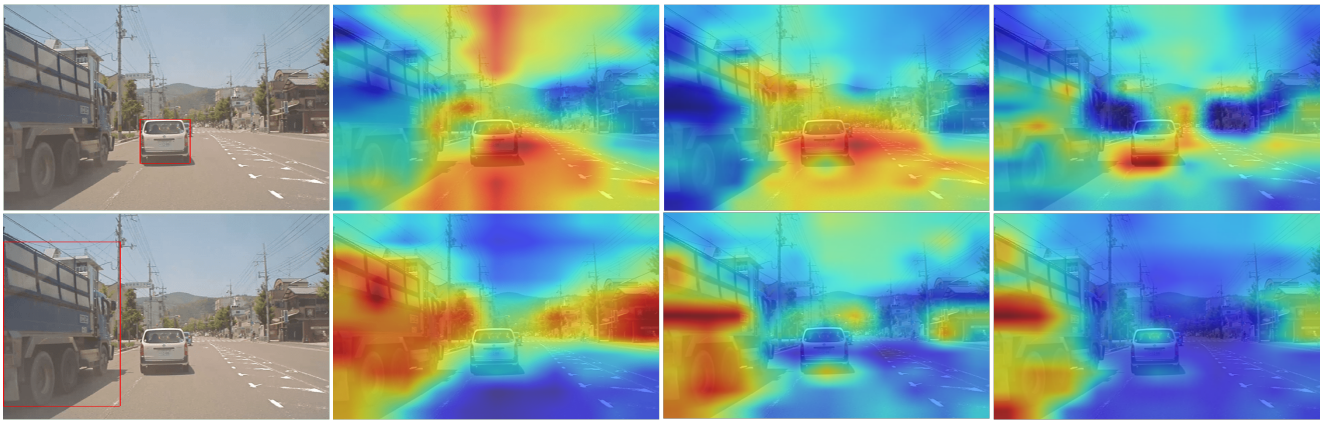
Effects of Transformers. We additionally compare InstaFormer with CNN-based model in Table 3. While single head version of InstaFormer has a lower parameter count, it shows better performance in terms of FID metric compared to CNN-based model. This demonstrates that our outstanding performance is not due to its complexity.

Variants	#blocks	#heads	#params	FID↓	SSIM↑
Less blocks	3	4	37.776M	89.96	0.711
Ours	6	4	75.552M	84.72	0.872
More blocks	9	4	113.329M	85.28	**0.879**
Less heads	6	1	4.732M	89.17	0.738
Ours	6	4	75.552M	84.72	0.872
More heads	6	8	302.100M	**81.92**	0.873
CNN-based	6	-	7.081M	89.73	0.708

Table 3. **Effects of the number of blocks and heads in our InstaFormer architecture, providing quantitative evaluations with FID [2] and SSIM [8].** The only setting that vary across model is the number of Transformer blocks or heads, and we keep the others constant for sunny→night on INIT dataset [6]. Larger models tend to have a higher parameter count, and better FID [2] and SSIM [8] metric scores.

(a) Input image (b) #blocks = 3 (c) #blocks = 6 (d) #blocks = 9

Figure 6. **Visual results on variants of the number of Transformer blocks.** (a) input image and translated images (b,c,d) with different number of Transformer blocks for sunny→night.



(a) Input image with instances (b) #heads = 1 (c) #heads = 4 (d) #heads = 8

Figure 7. **Visualization of learned self-attention.** (from top to bottom) attention map for a car and a truc, respectively. (a) input image and following self-attention maps (b,c,d) for different number of heads.

References

[1] Yunjey Choi, Youngjung Uh, Jaejun Yoo, and Jung-Woo Ha. Stargan v2: Diverse image synthesis for multiple domains. In *CVPR*, pages 8188–8197, 2020.

[2] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. In *NeurIPS*, pages 6626–6637, 2017.

[3] Liming Jiang, Changxu Zhang, Mingyang Huang, Chunxiao Liu, Jianping Shi, and Chen Change Loy. Tsit: A simple and versatile framework for image-to-image translation. In *European Conference on Computer Vision*, pages 206–222. Springer, 2020.

[4] Yahui Liu, Enver Sangineto, Yajing Chen, Linchao Bao, Haoxian Zhang, Nicu Sebe, Bruno Lepri, Wei Wang, and Marco De Nadai. Smoothing the disentangled latent style space for unsupervised image-to-image translation. In *CVPR*, pages 10785–10794, 2021.

[5] Taesung Park, Alexei A Efros, Richard Zhang, and Jun-Yan Zhu. Contrastive learning for unpaired image-to-image translation. *arXiv preprint arXiv:2007.15651*, 2020.

[6] Zhiqiang Shen, Mingyang Huang, Jianping Shi, Xiangyang Xue, and Thomas S Huang. Towards instance-level image-to-image translation. In *CVPR*, pages 3683–3692, 2019.

[7] Ilya Tolstikhin, Neil Houlsby, Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Thomas Unterthiner, Jessica Yung, Andreas Steiner, Daniel Keysers, Jakob Uszkoreit, et al. Mlp-mixer: An all-mlp architecture for vision. *arXiv preprint arXiv:2105.01601*, 2021.

[8] Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. Image quality assessment: from error visibility to structural similarity. *TIP*, 13(4):600–612, 2004.