Guided Image-to-Image Translation with Bi-Directional Feature Transformation

Badour AlBahar Jia-Bin Huang
Virginia Tech
{badour,jbhuang}@vt.edu

1. Overview

In this supplementary material, we provide additional results to complement the main paper. First, we provide examples of our model by visualizing the feature activations of each layer. Second, we describe the implementation details of our work. Third, we include figures for explaining the architectural differences in our ablation study. Fourth, we show the qualitative improvements of our proposed bi-directional feature transformation compared to uni-directional feature transformation.

2. Visualizing Feature Activations

We visualize the intermediate feature map representations in Figure 1 for both the pose transfer and depth upsampling tasks. We show the features immediately *before* and *after* each feature transformation layer for both the input and guidance branches.

3. Implementation Details

Network architecture We use the same bFT architecture to integrate the guidance signal into the image-to-image translation model for all tasks. We include the code of the bFT architecture for a layer l in Figure 2. For the image-to-image translation model, we use GAN framework with either UNet or ResNet based generators and the basic discriminator of PatchGAN [1] as the base architecture. We use the publicly available Pytorch implementations for these models [1]. In addition to the encoder for processing an input image, we add another encoder branch for the additional guidance signal. We use a feature transformation layer in place of each normalization layer in both encoders. This enables us to have a bi-directional flow of information between the input and the conditional guidance. For both pose transfer and depth upsampling we use a Resnet base architecture with 3 downsampling layers, while for texture transfer we use 7 layers of Unet.

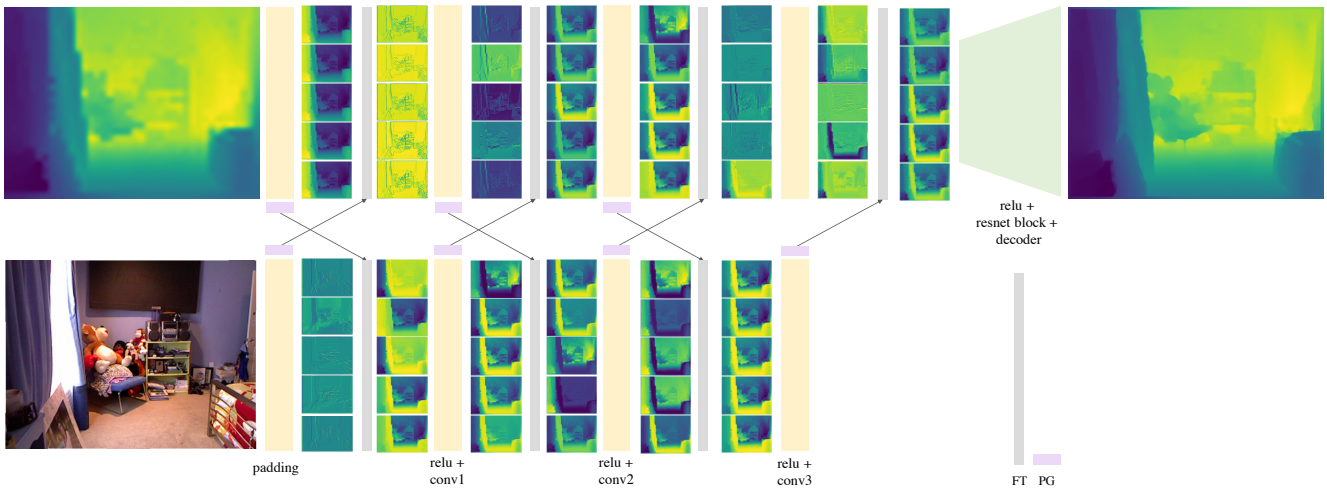
Hyper-parameters We have two hyper-parameters in our proposed model. We have the parameter generator bottleneck layer dimension of 100. We also have the loss function parameter λ, which we set to 100 for all experiments.

Model parameters and time We show the comparison of the number of parameters, training and inference time, and performance on depth upsampling with a scale factor of 16 for the different possible conditioning schemes. We show the training time of 1 epoch on NYU v2 [2] train set and the inference time on the whole test set.

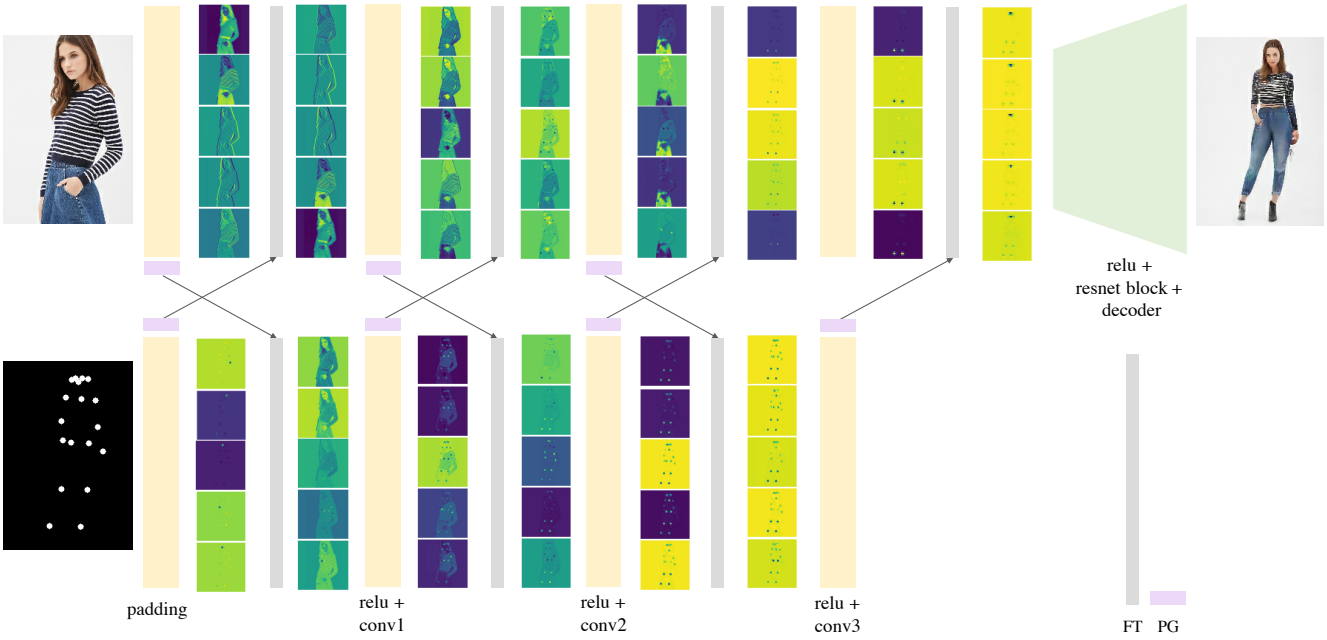
Table 1. Number of parameters and time.

	Number of parameters (millions)	Training time per epoch (seconds)	Inference time test set (seconds)	Depth upsampling RMSE
Input concatenation	45.590	215	33	11.86
Feature concatenation	47.664	250	37	11.59
uFT	47.526	320	50	11.41
bFT	47.913	393	63	9.01

[1] https://github.com/junyanz/pytorch-CycleGAN-and-pix2pix/blob/master/models/networks.py



(a) Depth upsampling feature map visualization



(b) Pose transfer feature map visualization

Figure 1. **Feature map visualization.** From each feature map, we show randomly sampled five response channels to visualize the effect before and after the proposed bi-directional feature transformation layer. Through the transformation, the guidance signal can be effectively propagated from the guidance branch to the input branch (and vice versa).

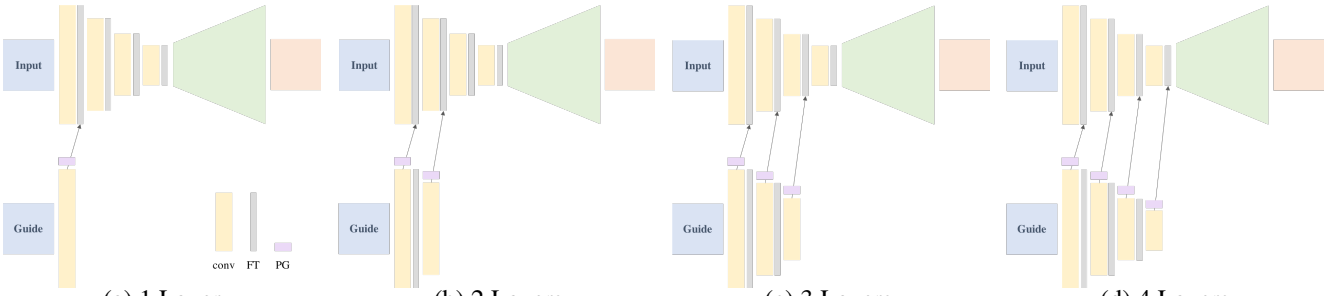
4. Ablation Study

In the ablation study experiments in the main paper, we evaluate the performance of variants of our method on pose transfer, depth upsampling, and texture transfer tasks.

Implementation details For pose transfer, we train all architectural variations with a learning rate of 0.0002 for 100 epochs. For depth upsampling, we train all architectural variations for 500 epochs with a learning rate of 0.0002, except for uFT, input concatenation, and feature concatenation we use a learning rate of 0.002. For texture transfer, we train all architectural variations for 100 epochs with batch size of 256 using a learning rate of 0.0002 with 7 layers of Unet architecture.

```
# feature transformation
def FT(x, gamma, beta):
    # we perform our FT pixel-wise affine transformation on the feature x
    mean, std = calculate_mean_std(x)
    mean = mean.expand_as(x)
    std = std.expand_as(x)
    return gamma * ((x-mean)/std) + beta

# parameter generator
def bottleneck_layer(nc, bottleneck_depth=100):
    return nn.Sequential(*[nn.Conv2d(nc, bottleneck_depth, kernel_size=1),
                nn.ReLU(True), nn.Conv2d(bottleneck_depth, nc, kernel_size=1)])

# Example of bi-directional communication
# assuming we are at layer = l
nc = number_of_channels_of_layer_l
input = input_features_of_layer_l
guidance = guidance_features_of_layer_l

# parameter generator for the input
input_gamma_generator = bottleneck_layer(nc)
input_beta_generator = bottleneck_layer(nc)

# parameter generator for the guide
guidance_gamma_generator = bottleneck_layer(nc)
guidance_beta_generator = bottleneck_layer(nc)

# compute parameters
input_gamma = input_gamma_generator(input)
input_beta = input_beta_generator(input)
guidance_gamma = guide_gamma_generator(guidance)
guidance_beta = guide_beta_generator(guidance)

# perform bFT
input = FT(input, guidance_gamma, guidance_beta)
guidance = FT(guidance, input_gamma, input_beta)
```

Figure 2. **Example of our bFT code for a layer l.** We use Pytorch to implement our work and show the code for how we apply bFT regardless of the guidance signal we have.

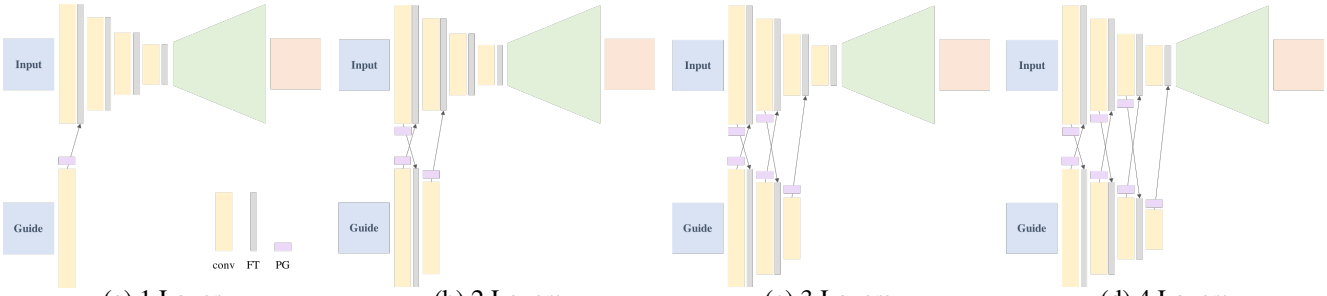


(a) 1 Layer (b) 2 Layers (c) 3 Layers (d) 4 Layers

Figure 3. **Number of feature transformation layers with uni-directional feature transformation.** We show the uni-directional feature transformation model with different number of feature transformation layers; 1, 2, 3 and 4. For our proposed uFT model we use 4 FT layers. The quantitative performance of these variations is shown in Table 5 in the main paper.

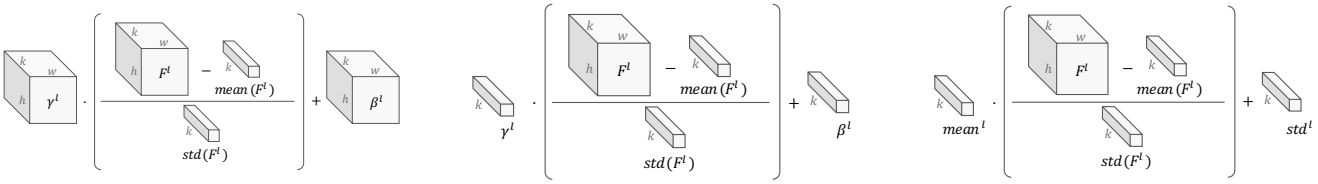
Architectural variations Here we visualize the differences of the variant architectures described in the main paper. We show the architectures used for testing the different number of FT layers for the *uni-directional* feature transformation in Figure 3 and the *bi-directional* variants in Figure 4.

To better illustrate the differences between our approach with affine transformation approaches CIN and AdaIN, we visualize the feature transformation layer in Figure 5. Moreover, we included in the paper the performance of different affine



(a) 1 Layer (b) 2 Layers (c) 3 Layers (d) 4 Layers

Figure 4. **Number of feature transformation layers with bi-directional feature transformation.** We show the bi-directional feature transformation model with different number of feature transformation layers; 1, 2, 3 and 4. For our proposed bFT model we use 4 FT layers. The quantitative performance of these variations is shown in Table 5 in the main paper.



(a) FT (b) CIN (c) AdaIN

Figure 5. **Approaches to affine transformation.** In (a) we show our proposed feature transformation layer which performs pixel-wise affine transformation. In contrast, in both (b) CIN and (c) AdaIN, we show the channel-wise affine transformations. Moreover, we show that CIN learns the scaling and shifting parameters vectors, while AdaIN, uses the mean and standard deviation as the scaling and shifting parameters, respectively. We show the quantitative comparison of these approaches in Table 6 in the main paper.

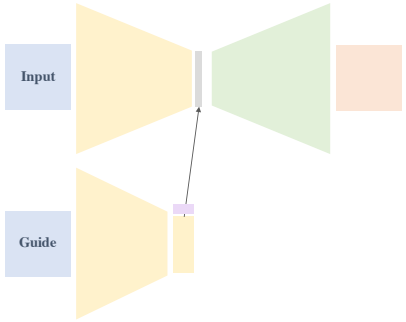


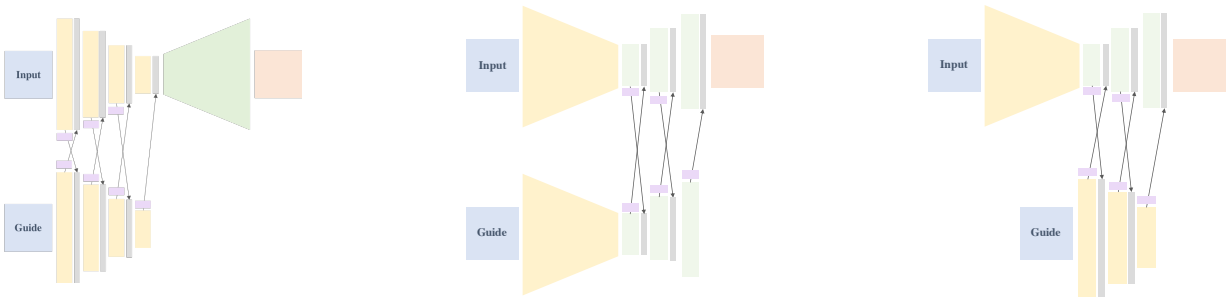
Figure 6. **Final layer affine transformation.** In this approach we only apply the affine transformation approach (CIN, AdaIN, or FT) at the final layer of the encoder. We show the quantitative comparison of these variations in Table 6 in the main paper.

transformation approaches when applied only at the last layer of the encoder. We show this architecture in Figure 6.

We also test different variations of where to apply our proposed bFT. In our work, we apply bFT between the encoder of the input and the encoder of guide. We compare our approach with bFT between the decoder of the input and the decoder of the guide, as well as between the decoder of the input and the encoder of the guide. The quantitative results are shown in Table 2. We illustrate the variations of where we apply our bi-directional feature transformation in Figure 7.

Table 2. bFT location.

Method guide-input	Depth Upsampling x16	Pose Transfer SSIM	 IS
Ours	9.01	0.767	3.17
decoder-decoder	10.97	0.760	3.03
encoder-decoder	11.96	0.773	3.04



(a) (b) (c)

Figure 7. **Location of the bi-directional feature transformation.** In our proposed work, we apply bFT between the encoder of the input and the encoder of guide as shown in (a). In (b) we apply bFT between the decoder of the input and the decoder of the guide. In (c) we apply bFT between the decoder of the input and the encoder of the guide. We show the quantitative comparison of these variations in Table 7 in the main paper. Such that (a) corresponds to the first row in Table 7, while (b) corresponds to the second row, and (c) corresponds to the third row.

Figure 8. **Uni-directional transformation vs. bi-directional transformation on pose transfer.** The qualitative results of our proposed bi-directional transformation compared to uni-directional transformation on pose transfer shows sharper results with better facial/hair and clothes synthesis.

5. Improvements of our bi-directional approach

We show the qualitative improvements of our proposed bi-directional approach compared to the uni-directional approach on pose transfer (Figure 8), depth upsampling (Figure 9), and texture transfer (Figure 10).

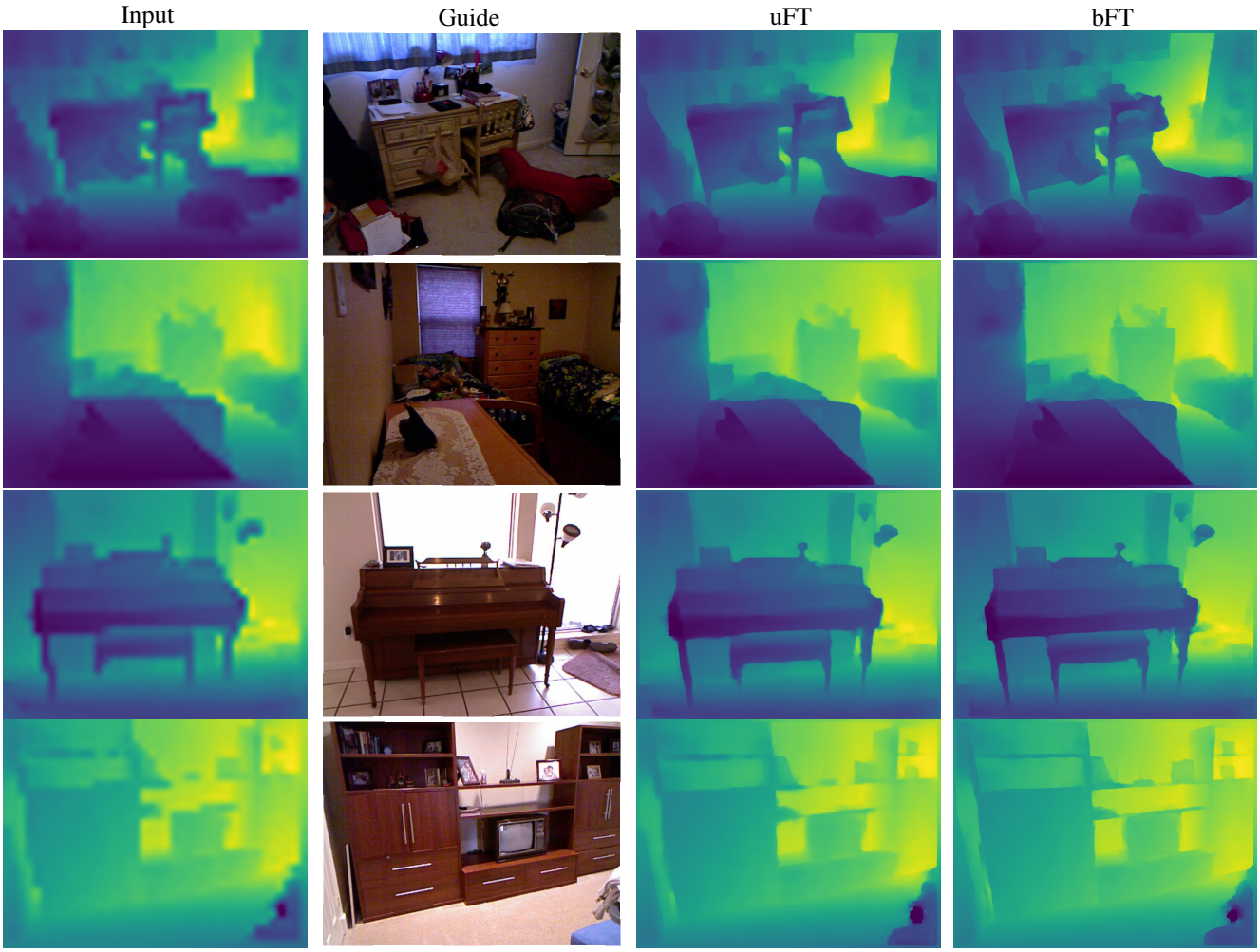


Figure 9. **Uni-directional transformation vs. bi-directional transformation on depth upsampling.** The qualitative results of our proposed bi-directional transformation compared to uni-directional transformation on depth upsampling shows sharper results.

Figure 10. **Uni-directional transformation vs. bi-directional transformation on texture transfer.** The qualitative results of our proposed bi-directional transformation compared to uni-directional transformation on texture transfer shows less artifacts. Moreover, the clothes dataset shows that bFT respects the texture patch more than uFT.

References

[1] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. *CVPR*, 2017. 1

[2] Nathan Silberman, Derek Hoiem, Pushmeet Kohli, and Rob Fergus. Indoor segmentation and support inference from rgbd images. In *European Conference on Computer Vision*, pages 746–760. Springer, 2012. 1