Generative Flows with Invertible Attentions (Supplementary Material)

Rhea Sanjay Sukthanker[1], Zhiwu Huang[1,2], Suryansh Kumar[1], Radu Timofte[1], Luc Van Gool[1,3]
[1]CVL, ETH Zürich, Switzerland [2]SAVG, SMU, Singapore [3]PSI, KU Leuven, Belgium
srhea@alumni.ethz.ch {zhiwu.huang,sukumar,radu.timofte,vangool}@vision.ee.ethz.ch

Abstract

The supplementary document includes: (i) further illustration of the suggested Jacobian computation, (ii) neural architecture design details of the proposed attention flows (AttnFlows), (iii) detailed experimental settings, (iv) training details and curves, (v) more visual results on MNIST, CIFAR10, CelebA and Cityscapes, (vi) better/more visualization plots of the ablation study presented in the main paper on the used datasets, as well as some more results for different iTrans-based attention head numbers on CelebA, (vii) pseudo code of the proposed key components (i.e., iMap and iTrans), and (viii) further remarks on the possible future directions.

1. Jacobian Computation

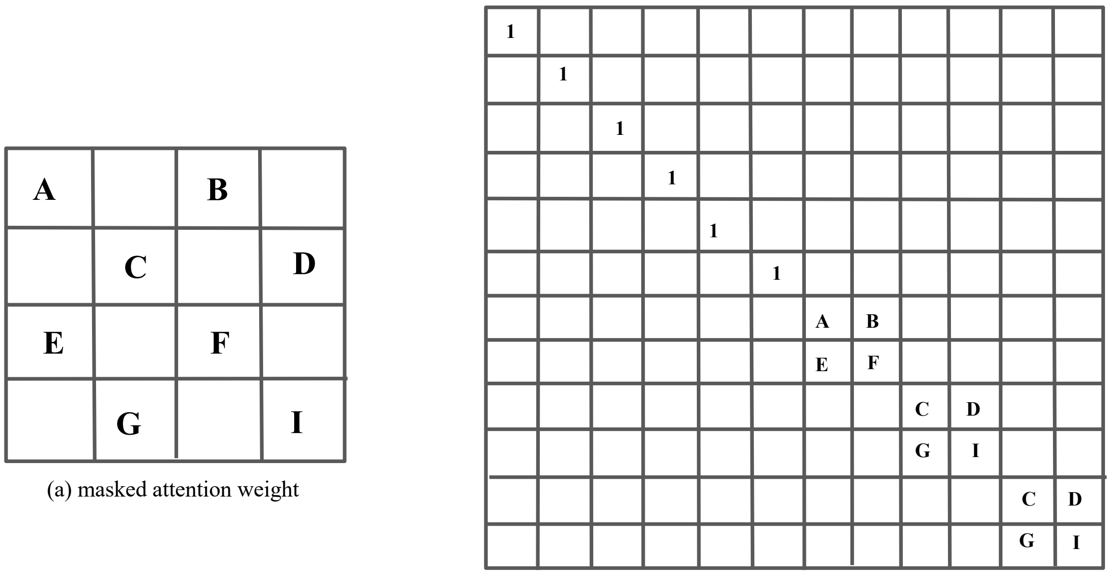
The suggested masking over the learned attention weights (Eqn.6 and Eqn.8 of the main paper) additionally leads to tractable Jacobian computation, i.e., Eqn.7 and Eqn.9 of the main paper. One example of the masked attention weight is illustrated in Fig.(1) (a). For this example, the resulting Jacobian is a block-lower triangular matrix, as shown in Fig.(1) (b). This is because one part of the input is made to depend on the other portion of the input. In this case, the determinant of a block triangular matrix can be easily computed by the product of the determinants of its block diagonal matrices. The resulting Jocobian determinant enables us to compute the log-likelihood of data efficiently by Eqn.2 (or Eqn.3) of the main paper.

2. AttnFlow Network Architecture

The proposed attention flow model (AttnFlow) aims to insert invertible map-based (iMap) and transformer-based (iTrans) attentions to regular flow-based generative models. Fig.(2) (a) (b) show the neural architecture design of regular flow-based generative models and the proposed AttnFlow respectively. As shown in Fig.(2) (b), the invertible attention modules (i.e., iMap and iTrans) can be stacked on the affine coupling layers. It is also possible to add the attention modules at any other positions, such as before invertible 1×1 convolution, or actnorm. The detailed architectures of the proposed iMap and iTrans are illustrated in Fig. (2) (c) (d) respectively. Their designs follow the conceptual graphs of forward and inverse propagation of the proposed Map-based and Trans-based attention mechanisms that are shown in Fig.(3) of the major paper. In particular, both of the iMap and iTrans modules apply the 3D checkboard mask in order to make the proposed attention invertible. After the 3D masking, the iMap attention further applies map-based transformations (i.e., 1D convolution and average pooling) for the first-order attention learning. By comparison, the iTrans attention aims at learning the second-order correlations among the flow feature maps with a scaled dot-product over two different transformations of inputs, which are obtained by two 2D convolutions, respectively. More architecture details of the proposed AttnFlow-iMap and AttnFlow-iTrans are shown in Fig.(2) (c) (d).

3. Detailed Experimental Setup

We used MNIST [7], CIFAR10 [6], CelebA [8] and Cityscapes [1] datasets to evaluate the proposed AttnFlows in the main paper. MNIST is a dataset of 70,000 small square 28×28 pixel grayscale images of handwritten single digits between 0 and 9. Following most of the generative modeling works such as [5, 10], we use the whole dataset from the real set to train our AttnFlows and the competing methods. CIFAR10 dataset is comprised of 60,000 32×32 pixel color images of objects from 10 classes, such as frogs, birds, cats, ships, airplanes, etc. To train the proposed AttnFlows and its competitors, we also utilize the whole dataset for the real data. We additionally evaluate the proposed cAttnFlows for face super-resolution ($8\times$) using 5000 160×160 images from the test split of the CelebA dataset. On CelebA, we use the full train split of CelebA for the training high-resolution image set. Following [9], we apply a bicubic kernel to down-scale those selected images into 20×20 low-resolution images. We use 162770 train-



(a) masked attention weight

(b) Jacobian of attention weight

Figure 1. (a) Example of masked attention weight. (b) Example of Jacobian matrix structure.

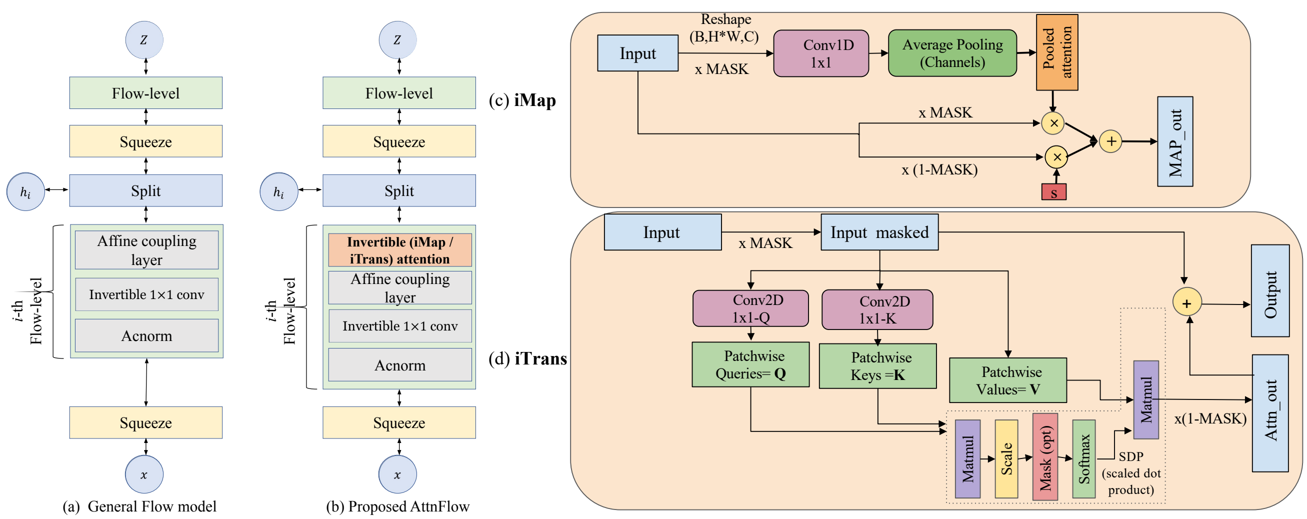
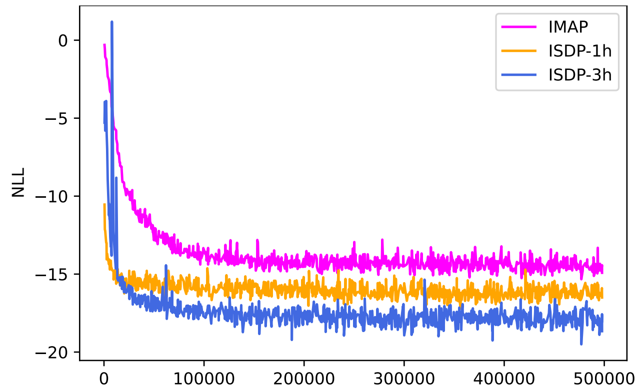
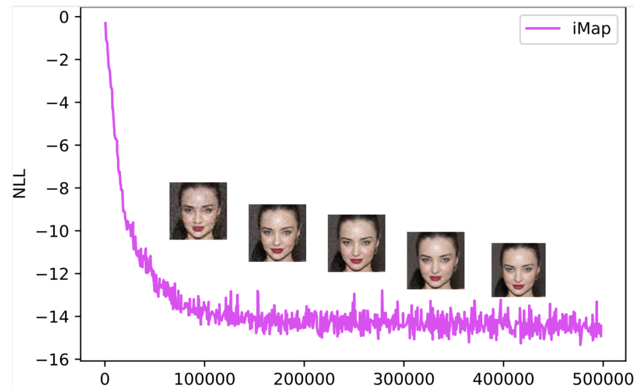


Figure 2. (a) Neural architecture of regular flow-based generative models. (b) Neural architecture design of the proposed attention flow model (AttnFlow), which aims at inserting invertible map-based (iMap) attention and transformer-based (iTrans) attention to regular flow-based generative models. $\mathbf{x}, \mathbf{h}_i, \mathbf{z}$ indicates the data, latent variable and intermediate coding respectively. (c) Detailed design of iMap. B, H, W, C indicate batch size, image hight, width, and channel number respectively. $\times$ MASK represents the masking operation that applies 3D checkboard mask to the input. Conv2D 1×1 is an invertible 2D convolution. s is a learnable scale. Finally averaged pooled features are fed with learnable parameters into MAP_{out} that is sigmoid function. (d) Detailed design of iTrans. MASK indicates the 3D checkboard masking, and Mask(opt) is optional.

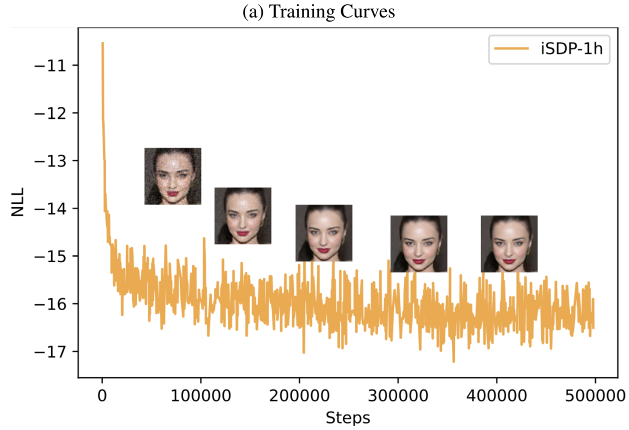
ing images following the same setup as [9] for the train-test split. We also use Cityscapes [1] to evaluate the proposed cAttnFlows. Each instance of this dataset is a 256×256 picture of a street scene that is segmented into objects of 30 different classes, e.g., road, sky, buildings, cars, and pedestrians. 5000 of these images come with fine per-pixel class annotations of the image, and this is commonly called as segmentation masks. We employ the data splits provided by the original dataset (2975 training and 500 validation images), and train different models to generate street-scene images conditioned on their segmentation masks.



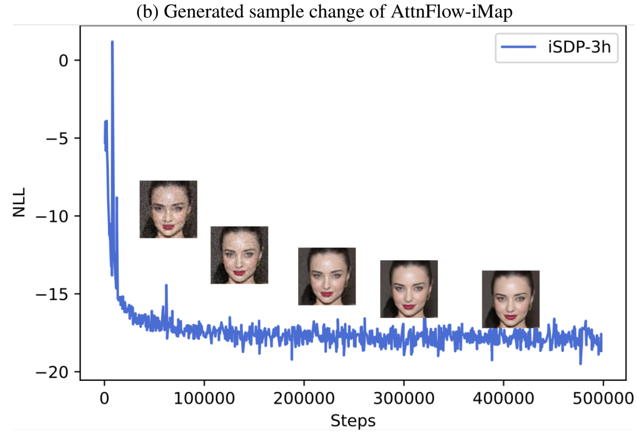
(a) Training Curves



(b) Generated sample change of AttnFlow-iMap



(c) Generated sample change of AttnFlow-iTrans (1 head)



(d) Generated sample change of AttnFlow-iTrans (3 heads)

Figure 3. (a) Training curves of the proposed AttnFlow-iMap and AttnFlow-iTrans (with 1, 3 head(s)) on CelebA. The x axis corresponds to the training iterations, and the y axis indicates the negative log-likelihood (NLL) values. (b-d) Generated sample change along the training process of AttnFlow-iMap and AttnFlow-iTrans (with 1, 3 head(s)), showing that the generated sample keeps improving the quality until the model converges.

4. Training Details and Curves

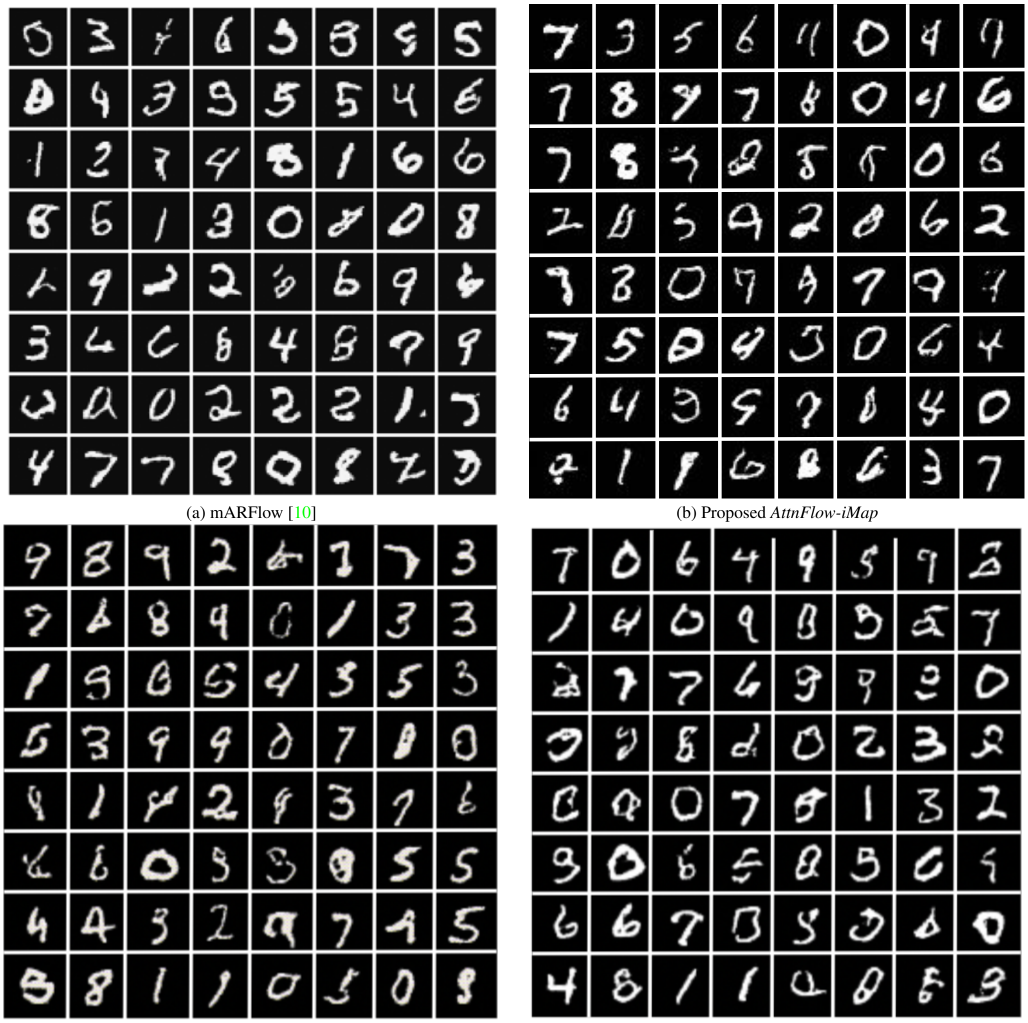
A single TITAN-RTX GPU (24GB) is used to train each of the proposed AttnFlows/cAttnFlows. Specially, the batch size[1] is set to 32 for the training on them on both of MNIST and CIFAR10, as done in [5, 10]. The proposed AttnFlow-iMap and AttnFlow-iTrans models (L=3, K=2) are trained for around 2 days and 3 days on MNIST, and they (L=3,K=4) are trained for 3 days and 5 days on CIFAR10. The number of iterations for convergence are 100k iterations and 90k iterations for MNIST and CIFAR10 respectively. The proposed cAttnFlow-iMap (L=1,K=1) is trained for 1.5 days, and cAttnFlow-iTrans is trained for 2 days on the CelebA datasets. A batch size is fixed as 16 for all the models on CelebA. The number of steps for convergence are 500k iterations for all the models. On the Cityscapes dataset, cAttnFlow-iMap (L=2,K=8) is trained for 2 days, and cAttnFlow-iTrans is trained for 3 days. A batch size is set to 1 for all the models due to the memory limit. The number of steps for convergence are 200k iterations for all the models on Cityscapes. Besides, we further compare our AttnFlows-iMap/iTrans against mARFlow [10] and SRFlow [9] (our two main backbones with the same levels and steps as those of ours) in terms of training time, training epochs and test time (per image) in Table 1. The results show that the proposed AttnFlows and the competing methods are trained at the same epochs. Their training and inference time are relatively comparable.

Method	Train time	Train epoch	Test time
mARFlow (CIFAR)	2.5 GPU days	100	0.48s
AttnFlow-iMap (CIFAR)	3 GPU days	100	0.50s
AttnFlow-iTrans (CIFAR)	5 GPU days	100	0.61s
SRFlow (CelebA)	1 GPU day	20	0.077s
AttnFlow-iMap (CelebA)	1.5 GPU days	20	0.104s
AttnFlow-iTrans (CelebA)	2 GPU days	20	0.317s

Table 1. Comparison of the proposed AttnFlow-iMap and AttnFlow-iTrans against their corresponding backbones mARFlow/SRFlow.

Fig.(3) (a) shows the training curves of the proposed AttnFlow-iMap and AttnFlow-iTrans on CelebA. It reflects that the proposed AttnFlows can be trained smoothly for a good convergence in terms of the negative log-likelihood (NLL) loss. In addition, Fig.(3) (b)(c)(d) demonstrate the

[1] Regarding the hyperparmeter setup, we use Adam with a learning rate of 8×10^{-4}, as done in [5, 10].



(a) mARFlow [10]
(b) Proposed *AttnFlow-iMap*
(c) Proposed *AttnFlow-iTrans* (1 head)
(d) Proposed *AttnFlow-iTrans* (3 heads)

Figure 4. Random samples generated by the proposed AttnFlows and the state-of-the-art flows on the MNIST dataset.

sample change along the training processes of the proposed AttnFlows. They show that the generated samples can keep improving the quality until the convergence.

5. More Results for MNIST, CIFAR10, CelebA and Cityscapes

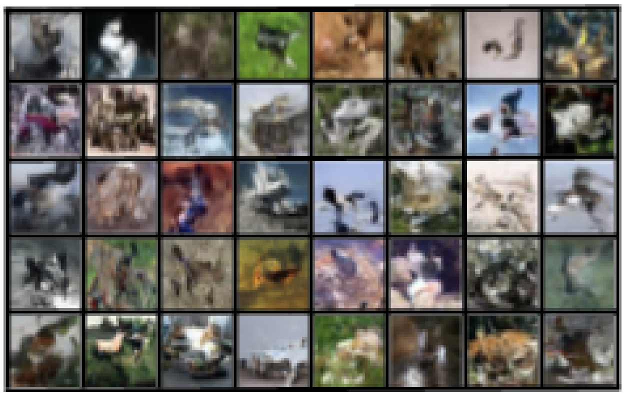
Fig.(4), Fig.(5), Fig.(6) and Fig.(7) show more visual results of the proposed AttnFlows and the competing methods on MNIST, CIFAR10, CelebA and Cityscapes respectively. From the results, we can observe that the generated samples of our proposed AttnFlows are highly competitive, and some are more visually pleasing compared to the competing methods.

6. More Ablation Study

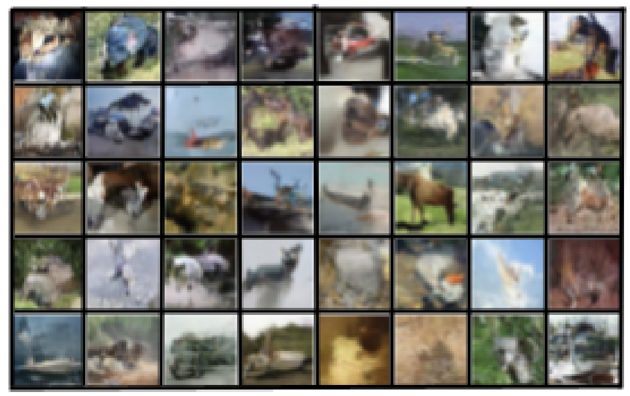
For the ablation study on the proposed AttnFlows, better/more visualizations of the major paper's Fig.(5) are shown in Fig.(8) and Fig.(9). Additionally, Fig.(9) includes the ablation study on different head numbers of the proposed AttnFlow-iTrans for CelebA. As shown in Fig.(8), inserting the attention modules after each coupling layer generally works the best in the most of cases. Besides, Fig.(9) shows that using 5 heads performs the best on the MNIST and CIFAR10 datasets, while employing 3 heads works the best on CelebA. This implies that using 3 or 5 heads is sufficient for the proposed AttnFlow-iTrans on the three employed datasets.

7. Pseudo Code of Proposed AttnFlows

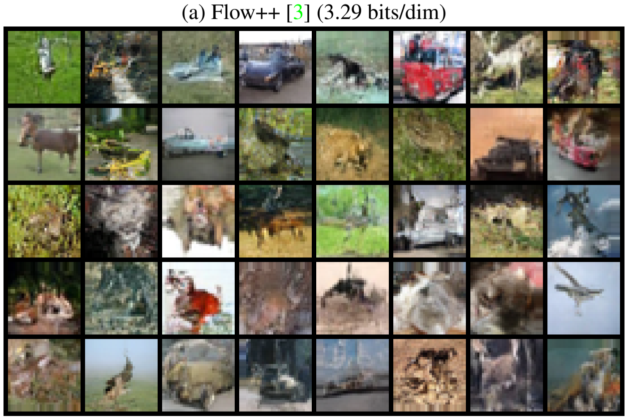
The proposed AttnFlow-iMap and AttnFlow-iTrans are built upon the off-the-shelf flow models. Therefore, the major new implementation is on the proposed iMap and iTrans modules. The pseudo codes for their PyTorch implementation are illustrated in Fig.(10).



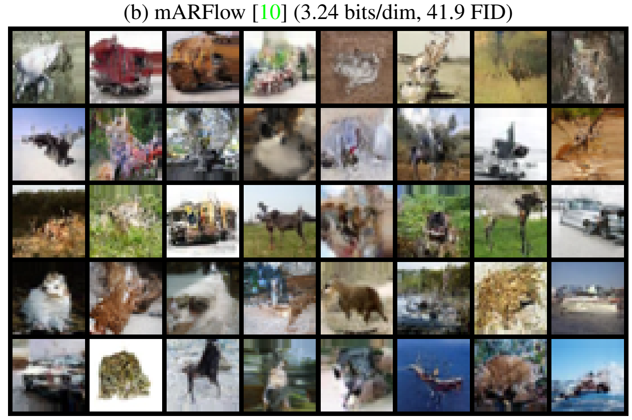
(a) Flow++ [3] (3.29 bits/dim)



(b) mARFlow [10] (3.24 bits/dim, 41.9 FID)



(c) *AttnFlow-iMap* (3.216 bits/dim, 33.6 FID)



(d) *AttnFlow-iTrans* (3.217 bits/dim, 33.8 FID)

Figure 5. Random samples generated by the proposed AttnFlows and the state-of-the-art flows on the CIFAR10 dataset.

8. Further Remarks for the Future Work

For the proposed AttnFlow-iTrans, we introduce a masked version of scaled dot-product, where the introduced masking can serve as a transformation (i.e., binary pattern generation). To improve the generalization capability, we could further apply a 1×1 2D convolution to the value $\mathbf{V}$, as done on the query $\mathbf{Q}$ and the key $\mathbf{K}$. Also, we exploit the multi-head attention mechanism for the AttnFlow-iTrans. To ease the computation on Jacobian determinant of iTrans, we choose to perform a summation instead of the commonly-used concatenation in conventional transformers over the resulting attended features from multi-heads. Following this work, we will be making more comprehensive study on the full exploitation of the multi-head attention scheme. Besides, as discussed in the main paper, it is non-trivial to apply the proposed AttnFlows to deeper flows, such as the full SRFlow model that contains more flow levels. We study that it is mainly because the proposed attentions' inverse and Jacobian determinant computations are often numerically unstable when meeting deeper flows. This is roughly matched with the discovery in [2], which finds pure attentions typically lose rank doubly exponentially with the network depth. Inspired by [2], a natural solution is to apply residual learning that is capable of addressing the deep attention problem. In addition, the comprehensive evaluations over the four used datasets show that the proposed AttnFlow-iMap sometimes outperforms AttnFlow-iTrans, while the former also performs worse in some cases. Hence, it is valuable to optimize the aggregation of the two complementary types of attention (i.e., first- and second-order attentions) for real-world scenarios. To this end, one of the most promising directions is to exploit neural architecture search algorithms over them.

References

[1] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *CVPR*, 2016. 1, 2

[2] Yihe Dong, Jean-Baptiste Cordonnier, and Andreas Loukas. Attention is not all you need: Pure attention loses rank doubly exponentially with depth. *arXiv preprint arXiv:2103.03404*, 2021. 5

[3] Jonathan Ho, Xi Chen, Aravind Srinivas, Yan Duan, and Pieter Abbeel. Flow++: Improving flow-based generative models with variational dequantization and architecture design. In *ICML*, 2019. 5

Figure 6. Super-resolved samples of the proposed cAttnFlows and the state-of-the-art models for 8× face SR on the CelebA dataset.

[4] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. In *CVPR*, 2017. 7

[5] Diederik P Kingma and Prafulla Dhariwal. Glow: Generative flow with invertible 1x1 convolutions. In *NeurIPS*, 2018. 1, 3

[6] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. *Technical report*, 2009. 1

[7] Yann LeCun, L eon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *PROC. OF THE IEEE*, 1998. 1

[8] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. In *ICCV*, 2015. 1

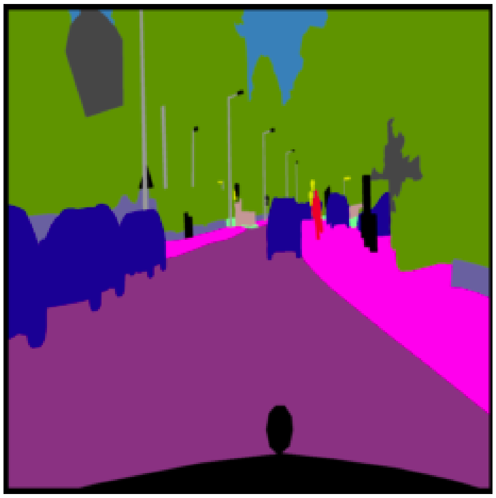
[9] Andreas Lugmayr, Martin Danelljan, Luc Van Gool, and Radu Timofte. SRFlow: Learning the super-resolution space with normalizing flow. In *ECCV*, 2020. 1, 2, 3, 6

[10] Shweta Mahajan, Apratim Bhattacharyya, Mario Fritz, Bernt Schiele, and Stefan Roth. Normalizing flows with multiscale autoregressive priors. In *CVPR*, 2020. 1, 3, 4, 5

[11] Moein Sorkhei, Gustav Eje Henter, and Hedvig Kjellström. Full-Glow: Fully conditional glow for more realistic image generation. In *GCPR*, 2021. 7

[12] Haoliang Sun, Ronak Mehta, Hao H Zhou, Zhichun Huang, Sterling C Johnson, Vivek Prabhakaran, and Vikas Singh. Dual-Glow: Conditional flow-based generative model for modality transfer. In *ICCV*, 2019. 7

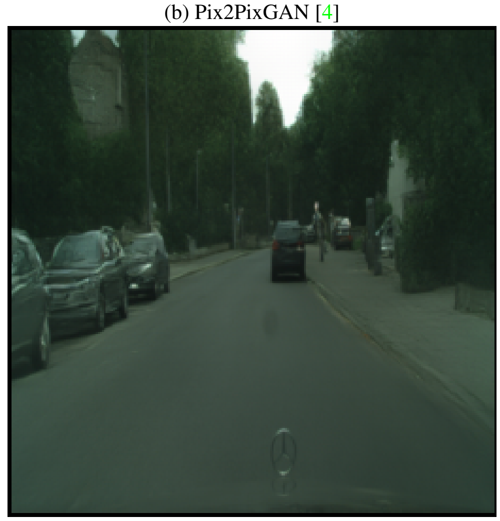
[13] Xintao Wang, Ke Yu, Shixiang Wu, Jinjin Gu, Yihao Liu, Chao Dong, Chen Change Loy, Yu Qiao, and Xiaoou Tang. ESRGAN: Enhanced super-resolution generative adversarial networks. In *ECCV*, 2018. 6



(a) Input

(b) Pix2PixGAN [4]

(c) Dual-Glow [12]



(d) Full-Glow [11]

(e) Proposed *cAttnFlow-iMap*

(f) Proposed *cAttnFlow-iTrans*

Figure 7. Generated samples of the proposed cAttnFlows and the state-of-the-art models for image translation on the Cityscapes dataset. The competing methods and ours are conditioned on the semantic segmentation labels (a) to synthesize the RGB images with the resolution being of 256×256.

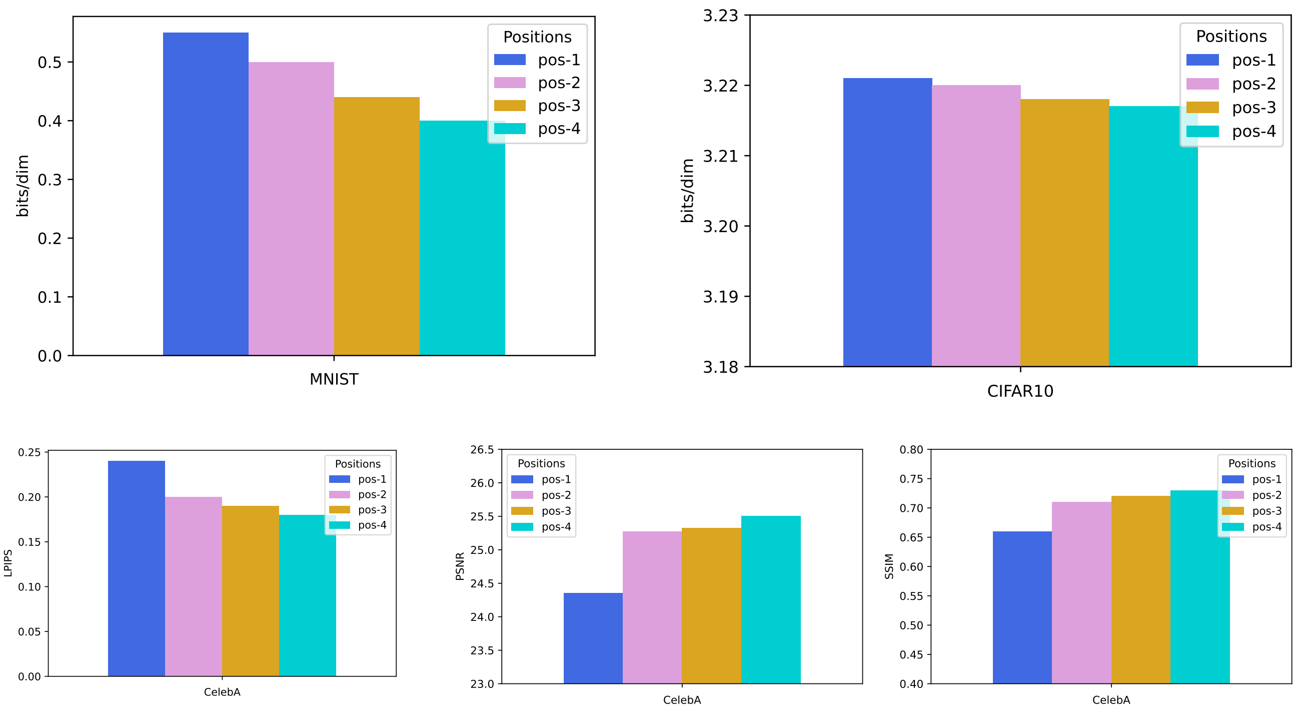


Figure 8. (Better/more visualisation for Fig.(5) in the major paper) Ablation studies of the proposed attentions on different positions in the flow layers (pos-1: before actnorm, pos-2: after actnorm, pos-3: after permutation, pos-4: after coupling layer) on the MNIST, CIFAR10 and CelebA datasets. The Bits/dims metric is employed for MNIST and CIFAR10 (top), and LPIPS, PSNR and SSIM scores are reported on CelebA (bottom).

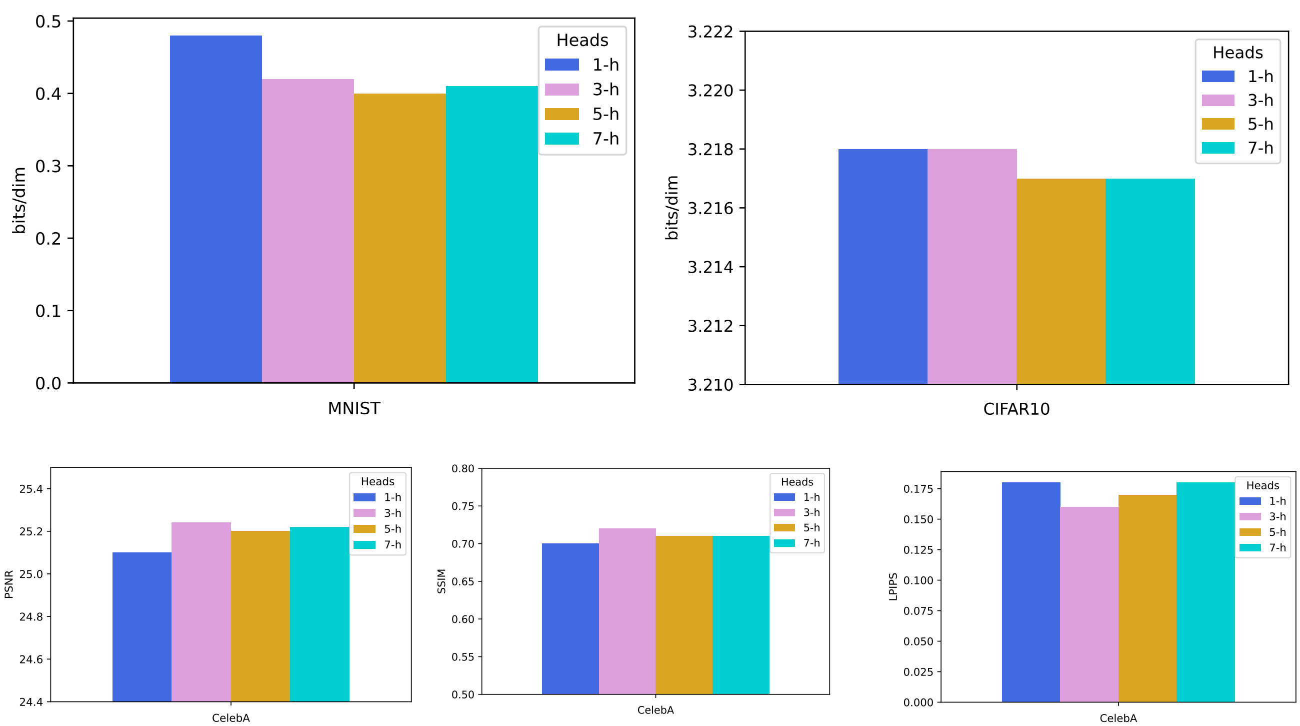


Figure 9. (Better/more visualisation for Fig.(5) in the major paper) Ablation studies of the proposed attentions on different number of attention heads for the proposed iTrans attention (1h: 1head, 3h: 3 heads, 5h: 5 heads, 7h: 7 heads) on the MNIST, CIFAR10 and CelebA datasets. The Bits/dims metric is employed for MNIST and CIFAR10 (top), and LPIPS, PSNR and SSIM scores are reported on CelebA (bottom).

```
class IMAP(nn.Module):
    def __init__(self, input_channels):
        super(IMAP, self).__init__()
        self.input_channels=input_channels
        self.weight = torch.empty([self.input_channels,self.input_channels,1])
        nn.init.kaiming_uniform_(self.weight, a=math.sqrt(5))
        self.weight=torch.nn.Parameter(self.weight).cuda()
        self.bias = torch.empty([self.input_channels])
        fan_in, _ = nn.init._calculate_fan_in_and_fan_out(self.weight)
        bound = 1 / math.sqrt(fan_in)
        nn.init.uniform_(self.bias, -bound, bound)
        self.bias =torch.nn.Parameter(self.bias).cuda()
        self.register_parameter("s",nn.Parameter(torch.randn([1,self.input_channels,1])))
        self.register_parameter("offset",nn.Parameter(torch.ones([1])*8))
        self.pool1 = torch.nn.AvgPool1d(self.input_channels)

    def forward(self, input: torch.Tensor, logdet=0, reverse=False, permute=False):
      if not reverse:
        self.num_channels=input.shape[-1]**2
        self.mask=checkerboard
        B, C, H, W = input.shape
        sig = torch.nn.Sigmoid()
        input_masked = input.view(B, C, H * W) * self.mask
        z = Conv1D(input_masked, self.weight, bias=self.bias) #Conv1d
        z_new = z.transpose(1, 2)
        pool_out = self.pool1(z_new) #Average pooling
        attn_out = (sig(pool_out.squeeze(-1)+self.offset)+1e-5).unsqueeze(1)
        attn_mask = (1 - self.mask) * attn_out + self.mask * (sig(self.s)+1e-5)
        out_new = input * attn_mask.view(B, C,H * W).view(B, C, H, W)
        logdet = logdet + torch.sum((self.input_channels// 2) * (torch.log(sig(pool_out.squeeze(-1)+self.offset)+1e-5)),dim=-1)
        logdet = logdet + torch.sum(torch.log(sig(self.s)+1e-5) * self.mask)
        return out_new,logdet
      else:
        out_new=input
        self.num_channels=input.shape[-1]**2
        self.mask=checkerboard
        B, C, H, W = out_new.shape
        sig = torch.nn.Sigmoid()
        s_sig = sig(self.s)+1e-5
        s_sig_in = torch.ones_like(s_sig) / s_sig
        inp_masked = out_new.view(B, C,H * W) * self.mask * s_sig_in
        out_conv = Conv1D(inp_masked, self.weight, bias=self.bias) #Conv1d
        pool_out = self.pool1(out_conv.transpose(1, 2))  # average pooling
        attn_out = (sig(pool_out.squeeze(2)+self.offset)+1e-5).unsqueeze(1)
        attn_out = torch.ones_like(attn_out) / attn_out
        attn_mask = (1 - self.mask) * attn_out + self.mask * s_sig_in
        input_rev = out_new * (attn_mask.view(B, C,H * W).view(B, C, H, W))
        logdet = logdet - torch.sum((self.input_channels// 2) * (torch.log(sig(pool_out.squeeze(-1)+self.offset)+1e-5)),dim=-1)
        logdet = logdet - torch.sum(torch.log(sig(self.s)+1e-5) * self.mask)
        return input_rev,logdet
```

```
class ITRANS(nn.Module):
    def __init__(self,num_channels):
        super (ITRANS,self).__init__()
        self.c = num_channels
        self.convq1 = torch.nn.Parameter(torch.randn([num_channels,num_channels,1,1]),requires_grad=True)
        self.s = torch.nn.Softmax(dim=-1)
        self.convk1 = torch.nn.Parameter(torch.randn([num_channels,num_channels,1,1]),requires_grad=True)
        self.register_parameter("offset",nn.Parameter (torch.ones([1,1,1])*1.01))
        self.register_parameter("scale", nn.Parameter(torch.ones([1,1,1])*10)
    def forward(self, input: torch.Tensor, logdet=0, reverse=False, permute=False):
        if not reverse:
            p = input.shape[-1]//2
            inp = rearrange(input) #get patches,
            mask = checkerboard
            inp_rev = reverse_rearrange(inp*mask)
            q1 = Conv2d(inp_rev, self.convq1) #Convq
            k1 = Conv2d(inp_rev, self.convk1)#Convk
            full_inp_q1 = rearrange(q1) #get patchwise features:Queries
            full inp_k1 = rearrange(k1)#get patchwise features:Keys
            attn_mask = checkerboard
            attn = (self.s((torch.matmul(full_inp_q1,full_inp_k1.permute(0,2,1))/self.scale)))*attn mask #Compute
                attention
            id = torch.eye(attn.shape[-1]).cuda()*self.offset #identity matrix to ensure proper logdet computation
            logdet_trans = torch.slogdet(attn+id)[1]*p*(p//2)*self.c
            logdet = logdet + logdet_trans
            out_attn = torch.matmul(attn+id,inp*(1-mask))
            out = out_attn*(1 - mask)+inp*mask
            output = reverse_rearrange(out)
        else:
            p = input.shape[-1]//2
            out = rearrange(input)
            mask = checkerboard
            if permute:
                mask = 1-mask
            rev = out*mask
            rev_rearrange = reverse_rearrange(rev)
            q1 = Conv2d(rev_rearrange,self.convq1)
            k1 = Conv2d(rev_rearrange,self.convk1)
            full_inp_q1_rev = rearrange (q1) # get patchwise features: Queries
            full_inp_k1_rev = rearrange (k1) # get patchwise features:Keys
            attn_mask = checkerboard
            attn = (self.s(torch.matmul(full_inp_q1_rev,full_inp_k1_rev.permute(0, 2, 1)/self.scale)))*attn mask
            id = torch.eye(attn.shape[-1]).cuda()*self.offset # identity matrix to ensure proper logdet computation
            logdet_trans = torch.slogdet(attn+id)[1]*p*(p//2)*self.c
            logdet = logdet - logdet_trans
            attn_inv = torch.inverse(attn+id)
            out_attn = torch.matmul(attn_inv,out*(1-mask))
            output = out_attn*(1-mask)+out*mask
            output = reverse_rearrange(output)
        return output, logdet
```

Figure 10. Pseudo code of the proposed AttnFlow-iMap and AttnFlow-iTrans.