GANSeg: Learning to Segment by Unsupervised Hierarchical Image Generation (Supplementary Material)

Xingzhe He Bastian Wandt Helge Rhodin
University of British Columbia
{xingzhe, wandt, rhodin}@cs.ubc.ca

A. Additional Results

To support the representative nature of the results shown in the main paper, we sample hundreds of images at random to evaluate both the quality of our generator as well as the keypoint detector applied to real images. Figures 8, 9, 10, and 11 show the randomly generated images along with the corresponding points and masks. Figures 12, 13, 14, and 15 show the detected masks on real test images sampled at random.

B. Implementation Details

B.1. Integrated Network Architecture

We provide a complete overview of the network architecture in Figure 1. It combines the individual modules introduced in the main paper.

B.2. SPADE ResBlock

Figure 2 (top) illustrates the SPADE ResBlock [13]. To be self-contained, we summarize the implementation of SPADE here. SPADE takes two feature maps as input. We denote the two feature maps as $\mathbf{F}_{\text{input}} \in \mathbb{R}^{D_{\text{input\_emb}} \times H \times W}$ and $\mathbf{F}_{\text{style}} \in \mathbb{R}^{D_{\text{style\_emb}} \times H \times W}$. We first use BatchNorm [6] to normalize $\mathbf{F}_{\text{input}}$ followed by two convolutions to map $\mathbf{F}_{\text{style}}$ to the new mean $\boldsymbol{\beta} \in \mathbb{R}^{D_{\text{input\_emb}} \times H \times W}$ and new standard deviation $\boldsymbol{\gamma} \in \mathbb{R}^{D_{\text{input\_emb}} \times H \times W}$ for the normalized $\mathbf{F}_{\text{input}}$. Although we use BatchNorm to normalize $\mathbf{F}_{\text{input}}$ in batch and channel dimension, we apply the generated $\boldsymbol{\beta}$ and $\boldsymbol{\gamma}$ to denormalize every individual element.

B.3. AdaIN ConvBlock

Figure 2 (bottom) illustrates the AdaIN ConvBlock [5]. AdaIN takes a feature map and a style vector as input. We denote the feature map as $\mathbf{F}_{\text{input}} \in \mathbb{R}^{D_{\text{input\_emb}} \times H \times W}$ and the style vector as $\mathbf{v}_{\text{style}} \in \mathbb{R}^{D_{\text{style\_emb}}}$. Unlike the original paper [5], which uses InstanceNorm [16], we found it beneficial in our case to use BatchNorm [6] to normalize $\mathbf{F}_{\text{input}}$. We use two fully connected layers to map $\mathbf{v}_{\text{style}}$ to the new mean $\boldsymbol{\beta} \in \mathbb{R}^{D_{\text{input\_emb}}}$ and new standard deviation $\boldsymbol{\gamma} \in \mathbb{R}^{D_{\text{input\_emb}}}$ for the normalized $\mathbf{F}_{\text{input}}$. As for the SPADE block, we use BatchNorm to normalize $\mathbf{F}_{\text{input}}$ in batch and channel dimension. For denormalization, $\boldsymbol{\beta}$ and $\boldsymbol{\gamma}$ are generated for each channel and broadcast to the spatial size of the feature map.

B.4. Hyperparameters

We use the Adam optimizer [11] with a learning rate of 0.0001 for the generator $\mathcal{G}$ and 0.0004 for the discriminator $\mathcal{D}$, both with $\beta_1 = 0.5$, $\beta_2 = 0.9$. We set the gradient penalty coefficient $\lambda_{\text{gp}} = 10$ for the discriminator. In every experiment, we update the generator 30,000 times. The learning rate for DeepLab is 0.0003, with $\beta_1 = 0.9$, $\beta_2 = 0.999$. DeepLab is trained for 10,000 iterations. The feature map resolutions are $(32^2, 32^2, 64^2, 128^2)$, respectively. We set $n_{\text{per}} = 4$ for all experiments. The penalty coefficients for each experiment is listed in Table 1. For Taichi, we set the vertical position of the background to be always in the center and only sample the horizontal position uniformly.

	CelebA	Taichi	CUB	Flower
λ_{con}	10	30	10	30
λ_{area}	1	1	1	1

Table 1. **Penalty coefficients** used in the main paper.

C. Image Quality

To quantify image quality, we report the FID score [15] for the models used in the main paper: 21.42 for CelebA-in-the-wild, 116.62 for Taichi, 38.74 for CUB, and 38.41 for Flowers. Note that FID numbers are not comparable across datasets. The FID is calculated by 50,000 generated images and the corresponding original dataset as established by [9]. Note that image quality is not our primary goal but it is still important since it influences how well the learned detector will generalize. The better the quality, the smaller the domain gap between generated and real images.

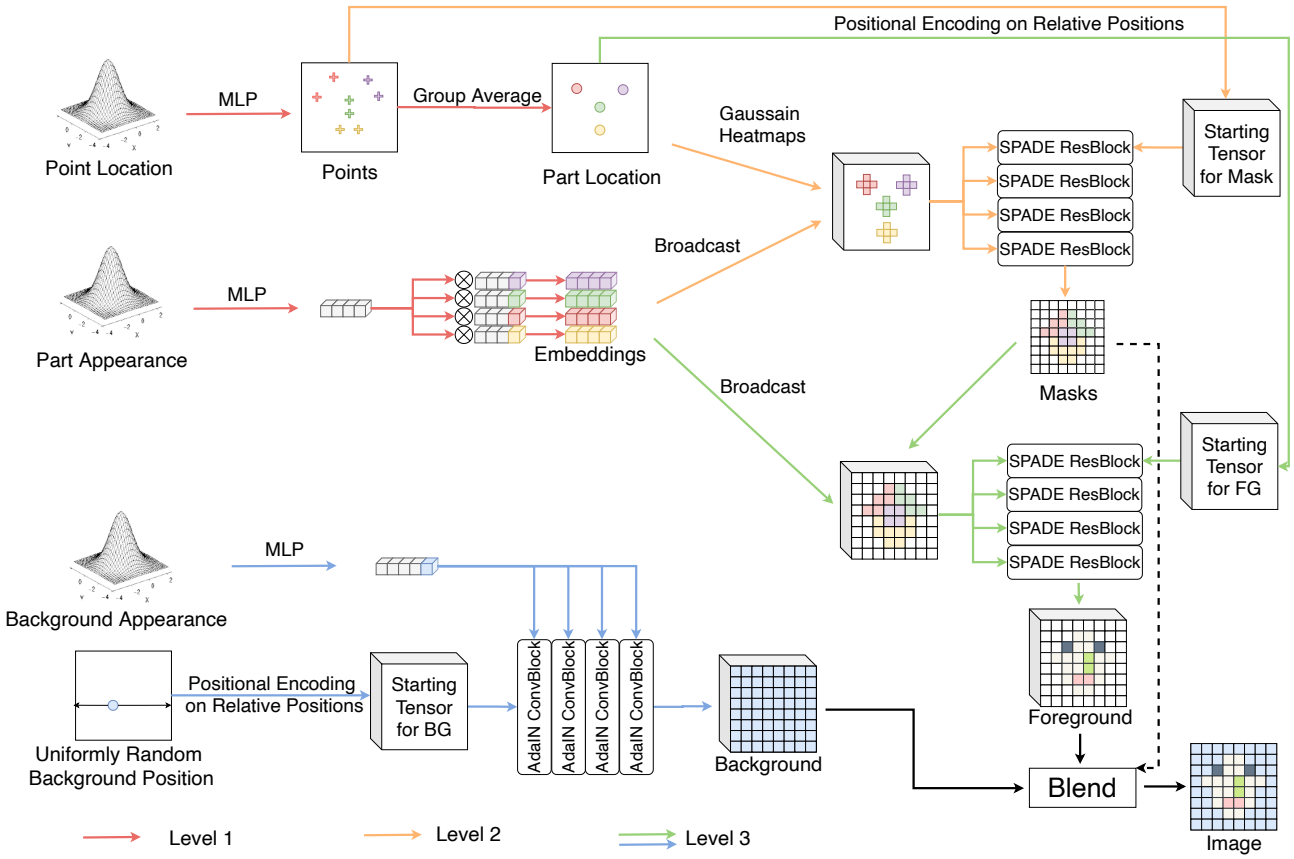

Figure 1. **Integrated network architecture** for better understanding of our data flow.

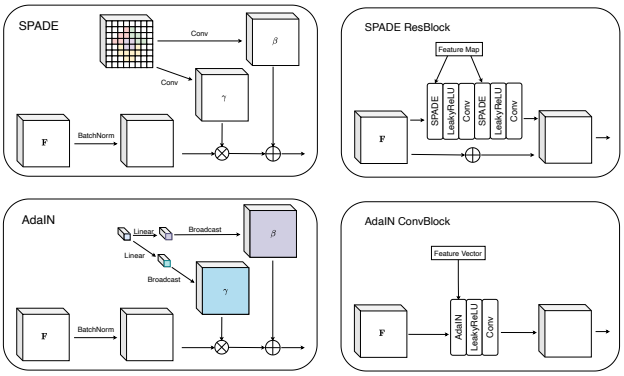

Figure 2. **SPADE ResBlock.** (top) and **AdaIN ConvBlock** (bottom).

D. Embedding Visualization

We generated 320 images for each dataset and provide the T-SNE visualization of their embedding vectors in Figure 3, including the background as a part. The embeddings from different parts are well separated in all datasets, indicating that our model learns unique embeddings for each part. Since overlapping part embeddings would mean that the same image features could be generated by multiple parts, the separation explains the good consistency of parts across images.

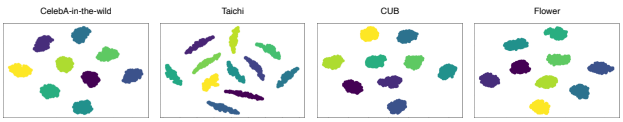

Figure 3. **Embedding visualization.**

E. Theoretical Analysis

In the following, we explain in detail how and why the proposed GAN architecture provides translation equivariance. On top of the translation equivariance of convolutions, we address equivariances in the point- and part-conditioned image generation.

Intuition. Consider a greyscale image, with a single point on it. Assume the image gets darker to the point. How should the image change if we move the point one pixel step to the right? We illustrate it in Figure 4 (Left). The left part of the image w.r.t. the point gets brighter and the right part gets darker. More specifically, if the point moves one pixel to the right, the pixels on the grid take the value of their left neighbor. The continuous case (1D for simplicity) is shown in Figure 4 (Middle). We assume that the point x controls the translation of the function f. If we move the x to $x + \Delta t$, the function f is also shifted by Δt. Then for each fixed position p, the value changes from $f(p)$ to $f(p - \Delta t)$. We desire a network architecture that fulfills this equivariance in the vicinity of each point.

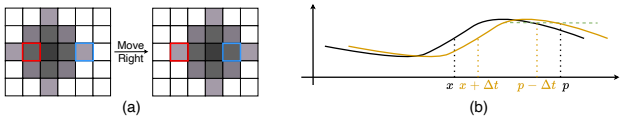

Figure 4. **Intuition.** (a) We move the points to the right by one pixel. The left part of the point (red box) becomes brighter and the right part of the point (blue box) becomes darker. (b) Continuous 1D case. (c) Rotation of parts can be achieved by translating points.

Formulation. Formally, let $\mathbf{I} \in \mathbb{R}^{H \times W}$ denote a feature map (or an image), $\mathbf{x} \in [-1, 1] \times [-1, 1]$ denote the point location, and $\mathbf{p} \in [-1, 1] \times [-1, 1]$ denote the pixel location. We have

$$\mathbf{I}(\mathbf{p} - \Delta\mathbf{t}, \mathbf{x}) = \mathbf{I}(\mathbf{p}, \mathbf{x} + \Delta t). \tag{1}$$

If we divide both sides by $\Delta\mathbf{t}$ (both axis independently), and let $\Delta\mathbf{t} \to \mathbf{0}$, we get the equation for the motion of a single point

$$-\frac{\partial \mathbf{I}(\mathbf{p}, \mathbf{x})}{\partial \mathbf{p}} = \frac{\partial \mathbf{I}(\mathbf{p}, \mathbf{x})}{\partial \mathbf{x}}. \tag{2}$$

This equation is the first integral of a homogeneous linear partial differential equation [17, 18]. Its solution is in the form of $\mathbf{I}(\mathbf{p} - \mathbf{x})$, where $\mathbf{I}$ can be any function. In other words, whenever $\mathbf{p}$ (or $\mathbf{x}$) shows up, it must be in a combination with $\mathbf{x}$ (or $\mathbf{p}$) as $\mathbf{p} - \mathbf{x}$ or $\mathbf{x} - \mathbf{p}$.

If there exist multiple points, each pixel may be influenced by multiple points. Assuming K points $\mathbf{x}_1, ..., \mathbf{x}_K \in [0, 1]^2$ gives

$$\mathbf{I}_k(\mathbf{p} - \Delta\mathbf{t}, \mathbf{x}_1, ..., \mathbf{x}_K) = \mathbf{I}_k(\mathbf{p}, \mathbf{x}_1, ..., \mathbf{x}_k + \Delta t, ..., \mathbf{x}_K), \tag{3}$$

where $k = 1, ..., K$.

We assume additivity of $\mathbf{I}_k$, which models the additivity of the generated feature maps in our convolutional generator well,

$$\mathbf{I} = \sum_{k=1}^{K} \mathbf{I}_k, \tag{4}$$

and write

$$\mathbf{I}(\mathbf{p} - \Delta\mathbf{t}, \mathbf{x}_1, ..., \mathbf{x}_K) = \sum_{k=1}^{K} \mathbf{I}_k(\mathbf{p}, \mathbf{x}_1, ..., \mathbf{x}_k + \Delta t, ..., \mathbf{x}_K). \tag{5}$$

As before, we divide both sides by $\Delta\mathbf{t}$ (both axis independently), and let $\Delta\mathbf{t} \to \mathbf{0}$. We get

$$\frac{\partial \mathbf{I}(\mathbf{p}, \mathbf{X})}{\partial \mathbf{p}} = -\sum_{k=1}^{K} \frac{\partial \mathbf{I}_k(\mathbf{p}, \mathbf{X})}{\partial \mathbf{x}_k}. \tag{6}$$

where $\mathbf{X} = \{\mathbf{x}_1, ..., \mathbf{x}_K\}$ for simplicity. Similarly, the solution to this equation is in the form of $\mathbf{I}(\mathbf{p} - \mathbf{x}_1, ..., \mathbf{p} - \mathbf{x}_k)$. Therefore, in theory, if we build a neural network that takes the difference between the points and the pixels as input, i.e., $(\mathbf{p} - \mathbf{x}_1, ..., \mathbf{p} - \mathbf{x}_k)$, this equation is automatically satisfied. Hence, unlike recent trends of adding absolute coordinates to networks [2, 3, 7, 19], we need to remove the absolute coordinate and point information from the model.

Background Handling. We generate foreground and background independently as foreground and background are not additive. The background exists on all pixels of the image, just with parts occluded by the foreground. Thus the movement of the background will not affect the foreground pixels. Mathematically, if only a non-empty subset of the K points influence the pixel $\mathbf{p}$, Equation 6 still holds because $\frac{\partial \mathbf{I}(\mathbf{p}, \mathbf{x}_1, ..., \mathbf{x}_K)}{\partial \mathbf{x}_k} = 0$ if $\mathbf{p}$ is not influenced by the point $\mathbf{x}_k$. However, if $\mathbf{p}$ belongs to the background (no point affects the pixel $\mathbf{p}$), the RHS becomes zero while the LHS does not (unless the background has only a single color). Therefore, we need to separate foreground and background and blend them at the end, to be able to assume additivity only on the foreground where it is meaningful.

Convolution. We use convolutions in our network since they are translation equivariant. In formulation, a neighbor of $\mathbf{p}$ has value $\mathbf{I}(\mathbf{p} - \mathbf{x}_1 + \Delta\mathbf{p}, ..., \mathbf{p} - \mathbf{x}_k + \Delta\mathbf{p})$ where $\Delta\mathbf{p}$ is a step to the neighbor. Note that $\mathbf{I}(\mathbf{p} - \mathbf{x}_1 + \Delta\mathbf{p}, ..., \mathbf{p} - \mathbf{x}_k + \Delta\mathbf{p})$ itself can be written as a function of $(\mathbf{p} - \mathbf{x}_1, ..., \mathbf{p} - \mathbf{x}_K)$, which makes our desired Equation 6 hold.

Why to use a GAN? Unlike most previous work which uses auto-encoders, we first use a GAN to generate points, masks, and images, and train a segmentation network to obtain segments. A core reason for us is that GANs can effectively prevent leaking absolute position [8]. As pointed out by [1, 7, 10, 19], a convolution (kernel size larger than 1)

Method	CelebA ↓	CUB ↑	Flowers ↑	Taichi (MAE) ↓	Taichi (IoU) ↑
fixed σ	26.32%	0.467	0.697	657.61	0.7452
original	**6.18%**	**0.629**	**0.739**	**417.17**	**0.8538**

Table 2. **Quantitative Ablation Tests on Number of Parts**. The metric for each dataset in this table follows the main paper.

with zero padding implicitly encodes absolute grid position. The more layers and downsampling layers, the larger is the region around the boundary that is affected by leaking absolute position [1]. However, if we were to remove the zero-padding, using only valid convolution, encoder will perform much worse at the boundary of images [12]. GANs, however, usually do not have downsampling but multiple upsampling layers. If we maintain a fixed margin around the feature map and crop after each upsampling [8], we can effectively prevent leaking absolute position and work with cropped and uncropped datasets. Moreover, autoencoders require to learn the encoder and decoder together, which implies a larger memory footprint and they generally lack behind in image generation quality.

F. Additional Ablation Tests

F.1. Replacing the GAN with an Auto-encoder

We tried two kinds of auto-encoders to replace our Point Generator (Level 1 of the hierarchy) and Mask Generator (Level 2). In the first one, we train a U-Net [14] to obtain points $\{\mathbf{x}_k^1, ...\mathbf{x}_k^{n_{\text{per}}}\}_{k=1}^K$, and two separate ResNet-18 [4] to extract part appearance vector $\mathbf{w}^{\text{dynamic}}$, and background appearance vector $\mathbf{w}_{\text{bg}}$ along with background position $\mathbf{u}_{\text{bg\_pos}}$, to feed into our Mask Generator that is left unchanged. In the second version, we directly use DeepLab V3 to generate masks $\mathbf{M}$, and then feed it to our Foreground and Background generator. We use the mean squared image reconstruction error to train this auto-encoder. We observe that all of the tested auto-encoders give trivial solutions, as shown in Figure 6. Hence, we decided to use a pure GAN setup.

F.2. Using Fixed Standard Deviation

Instead of estimating from points, we directly set a fixed standard deviation σ for each part. The fixed standard deviation is calculated by the average of the σ_k from our pre-trained model. We sample 5000 images. The average for all datasets are 0.00725 (CelebA-in-the-wild), 0.010 (Taichi), 0.0016 (CUB), and 0.0065 (Flower). The quantitative results are shown in Table 2 and the qualitative results are shown in Figure 5. The results show that fixing σ harms the performance severely. It is important to have various σ for each part and each object.

n_{per}	CelebA ↓	CUB ↑	Flowers ↑	Taichi (MAE) ↓	Taichi (IoU) ↑
3	**6.08%**	0.644	0.696	594.73	0.5863
4	6.18%	0.629	0.739	**417.17**	**0.8538**
6	8.84%	**0.682**	0.715	434.47	0.8336
8	13.02%	0.641	**0.744**	467.69	0.7792

Table 3. **Ablation Tests on Number of Points per Part**. The number of parts and the metric for each dataset in this table follow the main paper. We use $n_{\text{per}} = 4$ in the main paper.

Method	CelebA ↓	CUB ↑	Flowers ↑	Taichi (MAE) ↓	Taichi (IoU) ↑
K=1	56.23%	0.631	0.570	748.30	0.7895
K=4	12.26%	0.607	**0.767**	652.87	0.8176
K=8	6.18%	0.629	0.739	**420.95**	**0.8481**
K=12	**6.17%**	0.663	0.708	450.51	0.7939
K=16	6.71%	**0.692**	0.712	441.04	0.7475
K=32	7.00%	0.380	0.530	1605.84	0.1758

Table 4. **Quantitative Ablation Tests on Number of Parts**. The metric for each dataset in this table follows the main paper.

F.3. Number of Points per Part

We test on the influence of the number of points per part n_{per}. We found that the network always collapses if $n_{\text{per}} \leq 2$, and collapses occasionally if $n_{\text{per}} = 3$. Table 3 shows that the optimal n_{per} differs among different datasets. We choose $n_{\text{per}} = 4$ in our main paper for consistency.

F.4. Number of Parts

We test how the number of parts affects segmentation. Although the Area Loss is vital, its coefficient is not sensitive to the numbers of parts K. Thus for simplicity, we set $\lambda_{\text{area}} = 1$ as in the main paper. However, we found that small number of parts with large Concentration Loss coefficient causes the trivial or sub-optimal solutions. Therefore, we use the formula

$$\lambda_{\text{con}}(K) = C_{\text{con}} K \tag{7}$$

The constant C_{con} varies between datasets. In our experiments, we set 1.25 for Celeba-in-the-wild, 3 for Taichi, 1.25 for CUB, and 3.75 for Flower.

We show quantitative results in Table 4 and qualitative results in Figure 7. We found our model starts to degenerate when $K = 16$. Even though the IoU on CUB is higher, the mask degenerates, which is still a state-of-the-art foreground/background segmentation model but not a good part model. If $K = 32$, the model fails to distinguish the foreground from the background.

References

[1] Bilal Alsallakh, Narine Kokhlikyan, Vivek Miglani, Jun Yuan, and Orion Reblitz-Richardson. Mind the pad – {cnn}s can develop blind spots. In *International Conference on Learning Representations*, 2021. 3, 4

Figure 5. **Ablation Test on Estimated σ versus Fixed σ.** Examples are given for four different datasets.

Figure 6. **Ablation Test on replacing the GAN with an autoencoder.** Detected points and masks are overlayed on the corresponding images. In this experiment, points move to degenerate positions and scales. Masks are inaccurate.

[2] Xiangxiang Chu, Zhi Tian, Bo Zhang, Xinlong Wang, Xiaolin Wei, Huaxia Xia, and Chunhua Shen. Conditional positional encodings for vision transformers. *arXiv preprint arXiv:2102.10882*, 2021. 3

[3] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2020. 3

[4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 4

[5] Xun Huang and Serge Belongie. Arbitrary style transfer in real-time with adaptive instance normalization. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1501–1510, 2017. 1

[6] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015. 1

[7] Md Amirul Islam, Sen Jia, and Neil DB Bruce. How much position information do convolutional neural networks encode? *arXiv preprint arXiv:2001.08248*, 2020. 3

[8] Tero Karras, Miika Aittala, Samuli Laine, Erik Härkönen, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Alias-free generative adversarial networks. *arXiv preprint arXiv:2106.12423*, 2021. 3, 4

[9] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4401–4410, 2019. 1

[10] Osman Semih Kayhan and Jan C van Gemert. On translation invariance in cnns: Convolutional layers can exploit absolute spatial location. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14274–14285, 2020. 3

[11] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015. 1

[12] Rito Murase, Masanori Suganuma, and Takayuki Okatani. How can cnns use image position for segmentation? *arXiv preprint arXiv:2005.03463*, 2020. 4

[13] Taesung Park, Ming-Yu Liu, Ting-Chun Wang, and Jun-Yan Zhu. Semantic image synthesis with spatially-adaptive normalization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2337–2346, 2019. 1

[14] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015. 4

[15] Maximilian Seitzer. pytorch-fid: FID Score for PyTorch. https://github.com/mseitzer/pytorch-fid, August 2020. Version 0.1.1. 1

[16] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Instance normalization: The missing ingredient for fast stylization. *arXiv preprint arXiv:1607.08022*, 2016. 1

[17] Shiying Xiong and Yue Yang. The boundary-constraint method for constructing vortex-surface fields. *Journal of Computational Physics*, 339:31–45, 2017. 3

[18] Shiying Xiong and Yue Yang. Identifying the tangle of vortex tubes in homogeneous isotropic turbulence. *Journal of Fluid Mechanics*, 874:952–978, 2019. 3

[19] Rui Xu, Xintao Wang, Kai Chen, Bolei Zhou, and Chen Change Loy. Positional encoding as spatial inductive bias in gans. *arXiv preprint arXiv:2012.05217*, 2020. 3

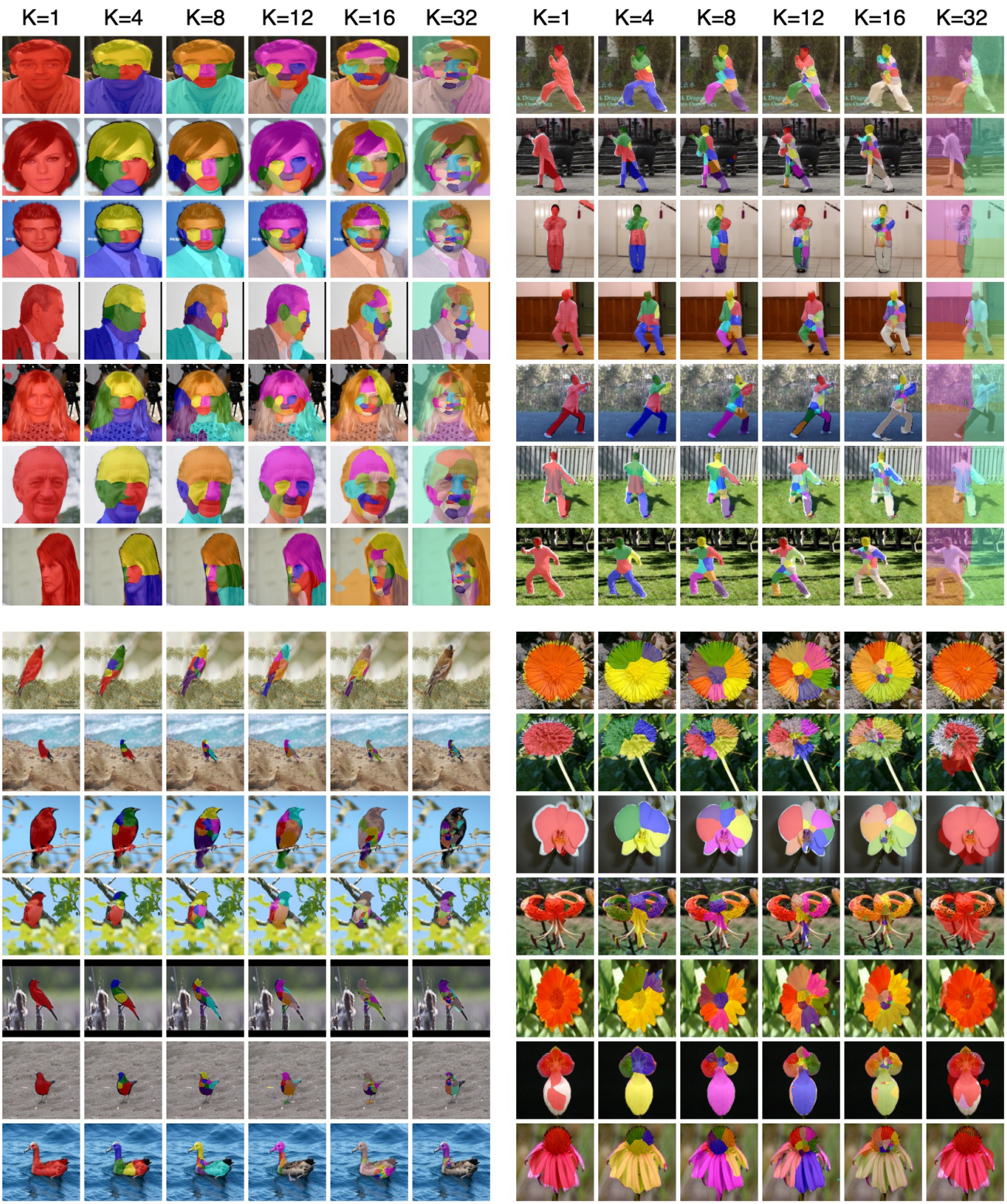

Figure 7. **Ablation Test on the number of parts.** Seven examples are tested for each dataset.

Figure 8. **90 generated samples from CelebA-in-the-wild,** with the generated image-points-masks pairs overlaid.

Figure 9. **90 generated samples from Taichi,** with image-points-masks pairs overlaid.

Figure 10. **90 generated samples from CUB,** with image-points-masks pairs overlaid.

Figure 11. **90 generated samples from Flowers,** with image-points-masks pairs overlaid.

Figure 12. **120 samples from CelebA-in-the-wild,** with detected masks overlaid.

Figure 13. **120 samples from Taichi,** with detected masks overlaid.

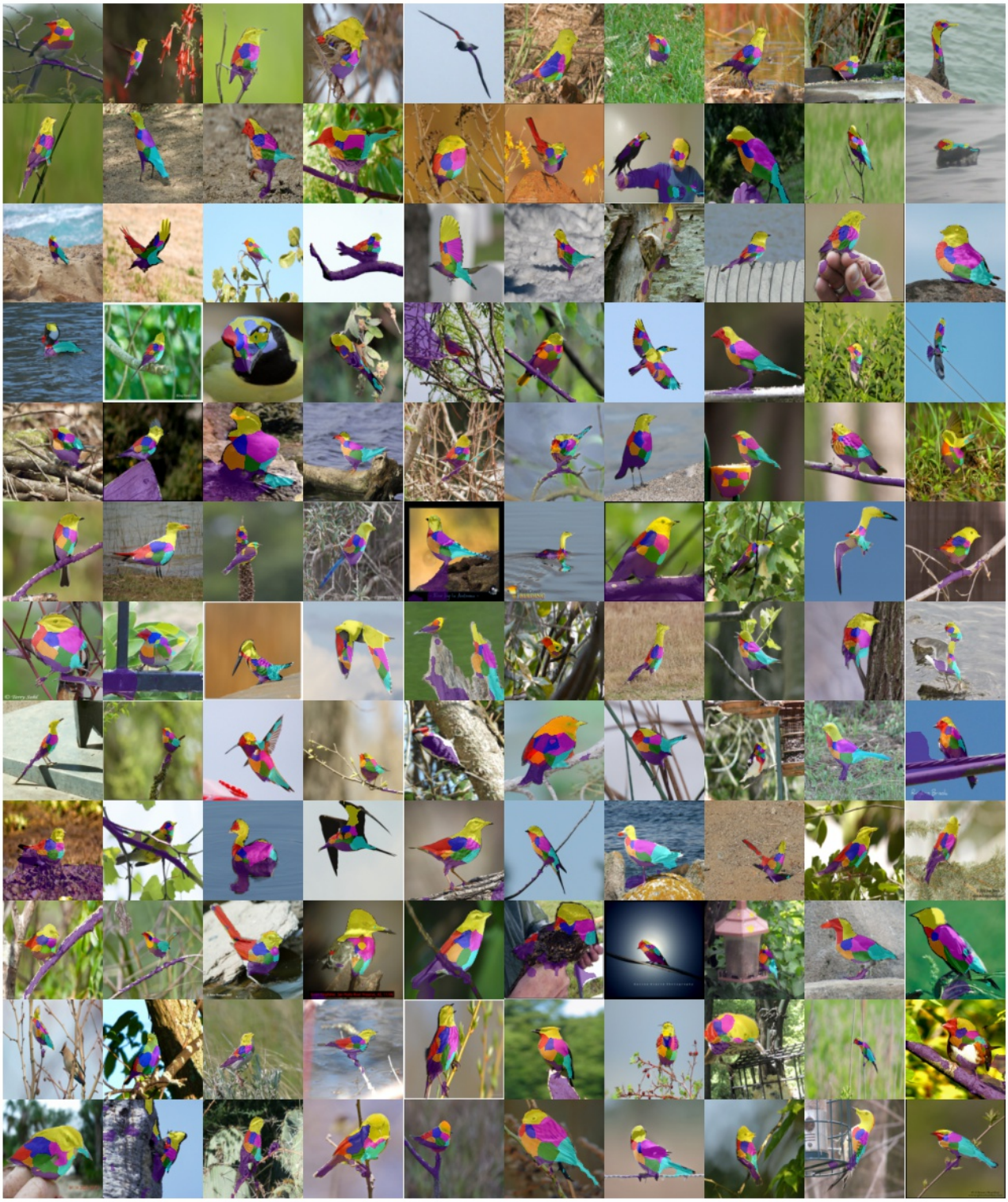

Figure 14. **120 samples from CUB,** with detected masks overlaid.

Figure 15. **120 samples from Flower,** with detected masks overlaid.