

Supplementary Material: PartGAN: Unsupervised Part Decomposition for Image Generation and Segmentation

Yuheng Li¹

li2464@wisc.edu

Krishna Kumar Singh²

krishsin@adobe.com

Yang Xue²

ayxue@ucdavis.edu

Yong Jae Lee¹

yongjaelee@cs.wisc.edu

¹ University of Wisconsin–Madison
Madison, WI, USA

² University of California-Davis
Davis, CA, USA

In this supplementary material, we first discuss the loss functions used in the background, shape, and texture stages. Then, we provide model architecture and training details. Lastly, we provide additional quantitative and qualitative results as well as ablation studies.

1 Training details of the first three stages

Background stage. As mentioned in the main paper, in the background stage, the background generator G_b takes the background code b , and a random noise vector z as input to synthesize a background image \mathcal{B} . We use patches outside the bounding boxes to model the background. Specifically, we use adversarial loss [2] at the patch level to train G_b :

$$\mathcal{L}_b = \min_{G_b} \max_{D_b} E[\log(D_b(x))] + E[\log(1 - D_b(\mathcal{B}))] \quad (1)$$

where D_b is the background discriminator and x are sampled real background patches.

Shape stage. In the shape stage, the network takes in a shape code s and a noise vector z to generate shape foreground and shape mask images. These two images are element-wise multiplied together to form the masked shape image. We want the shape code s (a one-hot code) to control the shape (e.g., duck vs sparrow), and the continuous vector z to control the pose variations associated with the shape. To achieve this, we apply mutual information loss [1] between masked shape image and shape code. Since the number of possible shapes (dozens) is generally much less than all possible poses (unlimited), mutual information loss can force code s to represent shape, leaving z to control the pose.

$$\mathcal{L}_s = \mathcal{L}_{s_info} = \max_{D_s, G_{p1}, G_{p2}} E_{z,s}[\log D_s(s|\mathcal{S}_{f,m})] \quad (2)$$

where $\mathcal{S}_{f,m}$ is masked shape image, and D_s is a network to approximate the posterior $P(s|\mathcal{S}_{f,m})$. G_{p1} and G_{p2} are networks implemented in the shape stage (see Figure 1 in the main paper).

Texture stage. Taking in shape feature F_s from the shape stage and a texture code t determined by the s code used in the last stage (e.g., say $N_s = 20$ and $N_t = 200$, then the texture codes will be evenly divided into 20 shape groups with 10 texture codes in each group, and the sampled t code must come from the corresponding shape group determined by the sampled s code), PartGAN generates the foreground texture in this stage. Texture codes are grouped according to shape code as there are certain textures which are specific to a shape. In this stage, we generate a texture foreground image and a texture mask. The texture mask is used to stitch the generated foreground texture to form the final image \mathcal{T} on which the adversarial loss is applied. In order to make the texture code represent different textures, we maximize mutual information between texture code t and masked texture image (element wise multiplication between texture mask and texture foreground). The losses applied in this stage are:

$$\mathcal{L}_{adv} = \min_{G_t} \max_{D_{adv}} E_x[\log(D_{adv}(x))] + E_{z,b,s,t}[\log(1 - D_{adv}(\mathcal{T}))] \quad (3)$$

where G_t (Clarification: In the main paper, G_C should be G_t in the texture stage of Figure 1) is the texture stage generator, D_{adv} is the final image discriminator trying to distinguish the real image x and generated image \mathcal{T} .

$$\mathcal{L}_t = \mathcal{L}_{t_info} = \max_{D_t, G_t} E_{z,t,s}[\log D_t(t|\mathcal{T}_{f,m})] \quad (4)$$

where $\mathcal{T}_{f,m}$ is masked texture image, and D_t is a network to approximate the posterior $P(t|\mathcal{T}_{f,m})$.

2 Implementation details

We train and generate 128×128 images for all datasets. For datasets where only landmark annotations are available, we crop objects by approximating bounding boxes around the landmarks. For the first three stages, we adopt the same architecture of FineGAN [8]. In our part stage, we use a Unet [2] structure for the part generator and the encoder.

Generative modules. G_b consists of 6 convolutional layers with BatchNorm [14]. Before each convolution layer, we first apply nearest neighbor upsampling to upscale the feature resolution. At the very beginning, we use a fully connected layer to convert input b, z to a vector of length $512 \times 4 \times 4$, then reshape it into a 3D volume with spatial size of 4×4 and 512 channels for the proceeding convolutional layers.

G_{p1} initially has the same structure as G_b . Then the intermediate feature again is concatenated with the shape code s , which is passed through a residual block and a pair of convolutional layers, to get the final shape feature F_s , which has 16 channels. The shape feature is then processed by two different single convolutional layers (which is G_{p2} in Figure 1 from main paper) to generate shape foreground and shape mask.

In G_t , similar to the later part of G_{p1} , shape feature is concatenated with the texture code t , and are processed by a residual block and a pair of convolutional layers to generate a

feature with size $16 \times 128 \times 128$, which is fed into two different single convolutional layers to generate texture foreground and texture mask.

In the part stage, we use a Unet [2] structure for the part generator. It consists of a series of upsampling and downsampling blocks which are implemented by convolutional layer, BatchNorm [2], and ReLU. Specifically we downsample input shape feature (with 16 channels) to the spatial size of 8×8 , while doubling channel size each time. We then perform upsampling by taking in previous upsampling results and resized shape feature. The final output layer has a size of 128×128 with k (number of parts) channels. In order to make sure our generated part masks only focus on foreground objects, we multiply the output from G_{pt} with the texture mask to obtain final part masks. The encoder adopts the same network architecture. For the decoder, we only implement it with three convolutional layers.

Discriminative modules. D_b consists of four convolutional layers, with leaky Relu as the activation function. For the last layer it outputs a 24×24 activation indicating real/fake scores. The network is designed such that each pixel in the output map corresponds to a 34×34 receptive field in the input image. D_s has eight convolutional layers, and it outputs a vector with length of s . Similarly, D_t outputs a vector with length of t . For D_{adv} , it shares all the network with D_t except for the last layer, where it has an independent layer to predict real/fake score for final image \mathcal{T} . For D_b , D_s , and D_t , initial convolutional layers output 64, 32 and 64 channels.

3 Keypoint predictor

In the main paper, in order to demonstrate that we predict consistent parts, we conduct experiments by predicting keypoints using learned part masks. Compared with hard/soft keypoints used in other methods, our part mask representation can preserve richer information. Thus, to take advantage of this richer information, we implement a convolutional predictor to predict the keypoints from the discovered part masks. For this, we use a Unet [2] structure model. We use convolution and transposed convolution to do downsampling and upsampling each time. BatchNorm [2] and ReLU are used after each layer. Here we start our first feature with 32 channels and double the number of channels each time until it reaches 512, and our bottleneck feature has a spatial size of 4×4 . During training, we use random crop as data augmentation.

4 More results and ablation

Comparison with the discriminative model SCOPS [2]. In addition to the generative model baselines, we also compare to the discriminative model of [2], which is a state-of-art self-supervised part segmentation approach. Just like our model, it predicts part masks, so we train a convolutional landmark predictor to regress 2D landmark points using the learned part masks. We use their released model on the CUB dataset. Their normalized error for landmark prediction on CUB is 4.55%, which is slightly better than ours (5.05%). However, their approach requires pre-trained models from ImageNet and is trained with ground truth object masks [2]. In contrast, we only need bounding box supervision. More importantly, [2] is not a generative model, thus it can not generate new images as our model does.

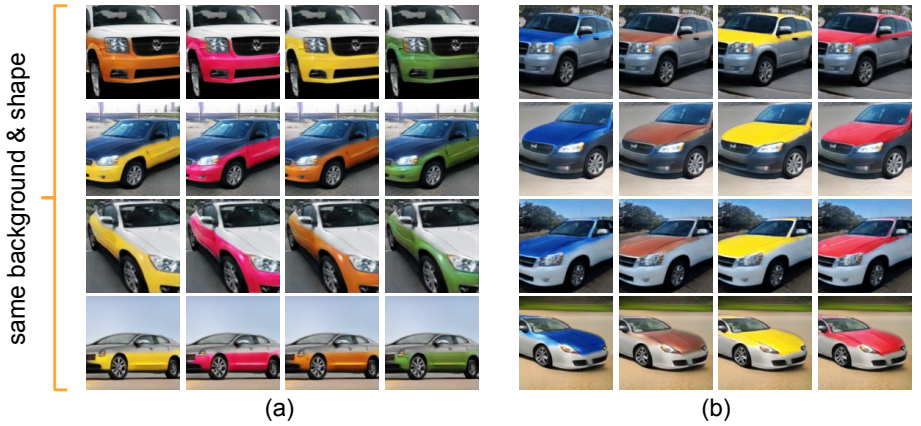


Figure 1: **Unconditional generation on Stanford Car.** See text for details.



Figure 2: **Unconditional generation on Cat head.** See text for details.

Unconditional part generation. Fig. 1 and Fig. 2 show unconditional generation results for cars and cats. The images in each row share the same background, shape, pose codes, and same texture code for all but one part. For the Stanford car dataset, we change the texture code for the (a) lower body, (b) upper body. For the Cat head dataset, we change the texture code for the (a) forehead, (b) ears. Note that each ear is actually learned as a different part as shown in the Figure 2 in the main paper, here we combine the two parts for a better visualization purpose. The results show that our model is able to learn consistent part masks across different instances, and it can also successfully change the texture of a specific region.

Conditional part generation. Fig. 3 and Fig. 4 show part-level texture transfer given real reference images. In each sub-figure, the real reference images on the top provide background, shape, and texture (for all but one part) information, while the reference images on the left provide the texture information of the part to be transferred. In these examples, we transfer the lower and upper region for cars, and ears and forehead region for cats. These results show that PartGAN can accurately locate consistent parts in different real images and transfer part texture successfully.

Better foreground-background disentanglement. Co-occurring context can often mis-

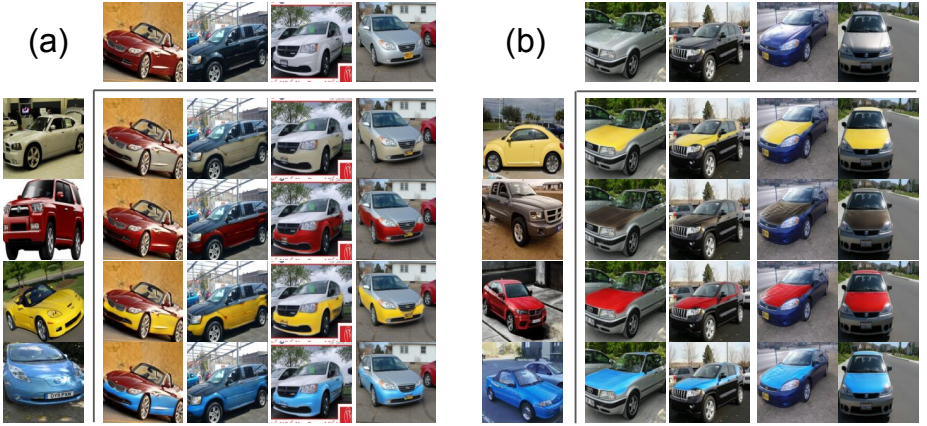


Figure 3: **Conditional generation on Stanford Car.** See text for details.

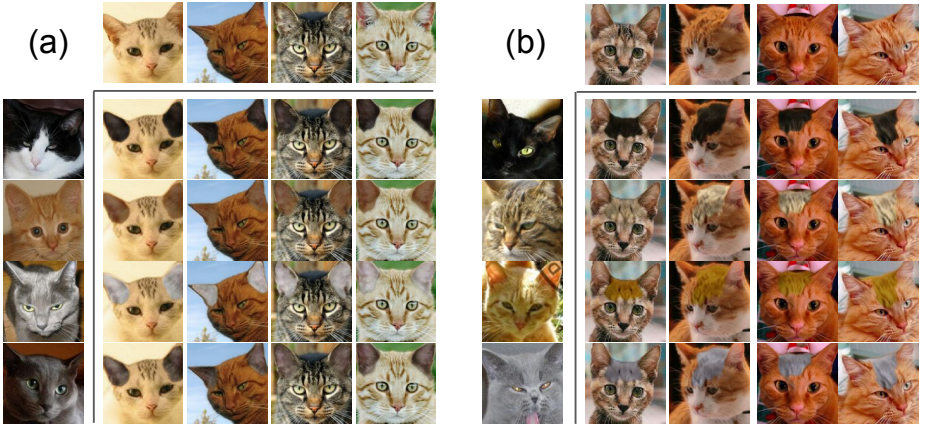


Figure 4: **Conditional generation on Cat head.** See text for details.

takenly be considered as part of the object (e.g., branches often appear with birds). Thus, when transferring texture from one image onto another, prior disentanglement work like MixNMatch [15] can mistakenly change the texture of the co-occurring context. Since PartGAN performs part decomposition and if the co-occurring context is learned as a separate part, in an interactive setting, one can subtract the context part from the texture stage mask (for all images at once) to improve background and foreground separation. For example, in Fig. 5, PartGAN can transfer the texture of the bird from the first column to the second column without changing the texture of the branch (fourth column). While MixNMatch can also do this, it also undesirably changes the texture of the co-occurring branch/shadow as it has no control over parts (third column). In the last two columns, we show how the foreground mask can be refined by removing the context part (cyan color) generated by PartGAN. We also quantitatively evaluate the quality of bird masks on CUB, and find our generated bird masks, after removing the extra context part, has 0.840 pixel-level AP which is significantly higher than MixNMatch’s 0.749 AP.



Figure 5: **Removing co-occurring context.** MixNMatch [5] often mistakenly considers co-occurring context as part of the object. Our model can recognize the context as a separate part, so one can remove the context for more accurate foreground texture transfer.

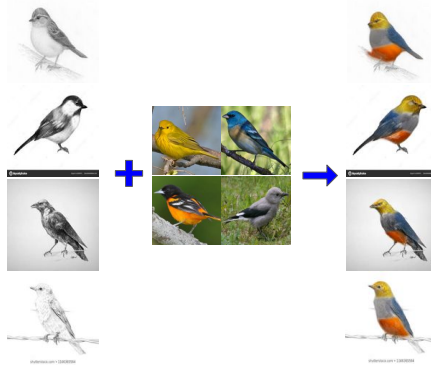


Figure 6: **Sketch to image generation.** In the first column, the four sketch images provide shape information, the middle four real images provide texture information for each part, and the last column are the final generated images.

Sketch-to-Image. Fig. 6 shows an application of PartGAN. It can be used to add texture details to one or multiple parts of sketch images. For example, the four real images in the middle provide texture details of the bird’s head, wing, chest, and belly regions, and the last column shows the texturized sketch images. It is worth noting that our model never saw any sketch images during training, but can still correctly decompose sketch images into parts consistently. This demonstrates the generalization ability of our model.

Video. We provide a video (named 57.mp4) which shows our model’s part decomposition consistency. Specifically, on the left, we interpolate a sequence of images where each time we change a single factor by sampling different latent codes (background b , shape s , texture t , and pose z). On the right, we show the discovered parts for each generated image. This video clearly indicates that our learned parts are consistent across different instances.

Ablation studies. Lastly, we conduct detailed ablation studies where we remove each of the loss functions from our full model. Table 1 shows the landmark prediction error of each ablation model on CUB using the previously described convolution predictor. The larger error of each ablation model shows the importance of each and every loss. Fig. 7 visually demonstrates the necessity of each loss component. Without the merge loss, the learned masks are faint and have gaps between them. Without the concentration loss, a single part can be comprised of two or more disjoint regions. Without the partition loss, different parts blend into each other. The balancing loss prevents parts from being too small.

	Ours	w/o merge	w/o concentration	w/o partition	w/o balance	w/o reconstruction
Error	5.05	7.03	7.85	6.84	7.78	6.95

Table 1: **Quantitative ablation studies** to demonstrate the necessity of each loss. We report landmark prediction error on CUB. See Fig. 7 for visualization results.

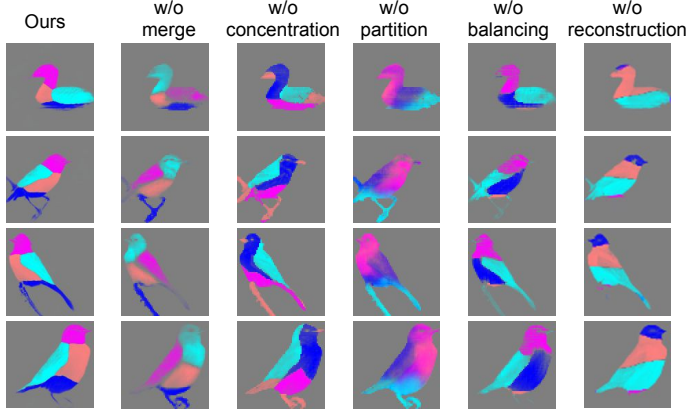


Figure 7: **Qualitative ablation studies.** The images above show that removing any loss (in each column) results in unsatisfactory part decomposition learning.

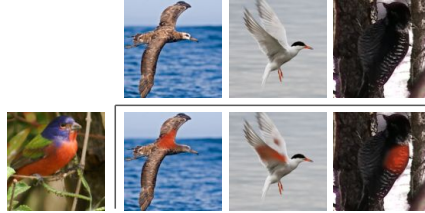


Figure 8: **Failure cases.** In this example, we try to transfer the ‘chest’ region texture from the image on the left to those on the top. Our model is unable to recognize the chest region due to rare pose or lack of foreground detail, leading to incorrect part generation results.

Finally, the model tends to learn parts according to their relative location in images without reconstruction loss.

Apart from evaluating the importance of each loss function, a plausible alternative solution to replace the merge and partition losses is applying the softmax across mask regions. However, we find that this is insufficient and not as explicit as our constraints for obtaining a single location with high probability for a single part mask. Empirically, this alternative solution leads to multiple parts capturing the same region (e.g. two parts with ~ 0.5 probability each).

Failure cases. PartGAN can sometimes mistakenly identify wrong object parts, which leads to unsatisfactory generations as shown in Fig. 8. In this example, we try to transfer the texture in the bird’s chest region of the image on the left to the chest region of the images on the top. Due to imbalanced data (i.e., flying birds are much less frequent than non-flying birds in the training data), similar color between foreground and background, or lack of foreground details, PartGAN fails to recognize the correct object part.

References

- [1] Xi Chen, Yan Duan, Rein Houthoofd, John Schulman, Ilya Sutskever, and Pieter Abbeel. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. In *NeurIPS*, 2016.
- [2] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *NeurIPS*, 2014.
- [3] Wei-Chih Hung, Varun Jampani, Sifei Liu, Pavlo Molchanov, Ming-Hsuan Yang, and Jan Kautz. Scops: Self-supervised co-part segmentation. In *CVPR*, 2019.
- [4] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *ICML*, 2015.
- [5] Yuheng Li, Krishna Kumar Singh, Utkarsh Ojha, and Yong Jae Lee. Mixnmatch: Multi-factor disentanglement and encoding for conditional image generation. In *CVPR*, 2020.
- [6] NVlabs. <https://github.com/nvlabs/scops>. 2019.
- [7] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *MICCAI*, 2015.
- [8] Krishna Kumar Singh, Utkarsh Ojha, and Yong Jae Lee. FineGAN: Unsupervised hierarchical disentanglement for fine-grained object generation and discovery. In *CVPR*, 2019.