Supplementary Material: MixNMatch: Multifactor Disentanglement and Encoding for Conditional Image Generation

Yuheng Li Krishna Kumar Singh Utkarsh Ojha Yong Jae Lee
University of California, Davis

In this supplementary material, we first introduce some key points of our training details. Next, we elaborate on our model's feature mode (second stage) training. Then, in Sec. 3, we discuss the usage of bounding box annotations during training for background generation. In Sec. 4, we provide details on texture disentanglement, and report shape and texture disentanglement results for all 15 keypoints for all methods. Finally, we show more qualitative disentanglement results and discuss the video clips which further demonstrate the disentanglement ability of our model.

1. Training details

We optimize our model using Adam with learning rate 0.0002, $\beta_1 = 0.5$, $\beta_2 = 0.999$ for 600 epochs. Following FineGAN [5], we crop all the images to 1.5× of their available bounding boxes.

As mentioned in the main paper, in our code mode (first stage) training, we use four paired discriminators to help encoders learn disentanglement. For each paired discriminator, there are two initial branches of convolution blocks which process the code and image, respectively. Then, their outputs are concatenated and fed into a series of convolution blocks to predict whether the input image-code pair is real or fake (during training, we set the image-code pair from encoders as real, and the image-code pair from generator as fake). In the code branch, we add Gaussian noise after each activation layer in order to avoid the discriminator from trivially recognizing that the one hot code in image-code pair from generator is a fake (since the encoded code from the encoders will never be one hot). Also, we update the paired discriminator using Wasserstein GAN [1] with gradient penalty.

2. Feature mode details

In feature mode (second stage) training we only train a shape and pose feature extractor S. Concretely, we fix the trained code mode (first stage) MixNMatch generator, and treat it as a real feature distribution provider. We then randomly sample p and z codes from their prior code distribution (categorical and normal distribution, respectively) and also predict p and z using their trained encoders on randomly sampled real images with equal probability. We feed these codes into the fixed parent stage generator to get an intermediate feature $\phi(p, z)$ (we use the feature F_p outputted from generator G_p according to [5]). As this feature is the output of the parent stage generator, it only contains shape and pose information. Thus, by applying an adversarial loss on the feature extractor S to match the distribution of $\phi(p, z)$, we can extract shape and pose information from real images x.

As mentioned in the main paper, we use a patch discriminator for this feature mode (second stage) training; specifically, we use a patch size of 34 x 34. Finally, in order to preserve instance-specific shape and pose details, we also generate fake images using our pretrained MixNMatch generator and compute their $\phi(p, z)$. Then, for each fake image, we input it into the feature extractor S, and apply an L1 loss between the resulting feature and its $\phi(p, z)$. In summary, our loss to train S is:

$$\mathcal{L}_S = \mathcal{L}_{adv} + \mathcal{L}_{L1} \tag{1}$$

where $\mathcal{L}_{adv} = \min_S \max_{D_S} \mathbb{E}_{\phi(p,z)}[\log(D_S(\phi(p, z)))] + \mathbb{E}_x[\log(1 - D_S(S(x)))]$ and $\mathcal{L}_{L1} = |S(G(b, c, p, z)) - \phi(p, z)|$. Here D_S is the feature discriminator.

3. Background modeling

As mentioned in the main paper, we only use bounding box annotations during training to model the background. Since we do not have any background training images without the object-of-interest (e.g., trees without bird), for each training image, we treat patches that are completely outside of the bounding box annotated (object) region as being the "real" background patches. We then train the background generator G_b to generate realistic background images, by applying a patch-level background discriminator D_b using the adversarial loss, following [5].

Once our model is trained, we do not need any bounding box annotations for image generation.

	Deforming AE [4]		SC-GAN [3]		FineGAN [5]		MixNMatch (c)		MixNMatch (f)	
	shape	texture	shape	texture	shape	texture	shape	texture	shape	texture
back	75.08	0.816	27.60	0.679	16.69	0.637	16.52	**0.561**	**13.92**	0.584
beak	62.54	0.707	32.92	0.565	21.16	0.599	21.38	**0.509**	**12.84**	0.526
belly	61.58	0.873	30.86	0.778	19.56	0.683	19.51	**0.633**	**16.92**	0.656
breast	66.93	0.859	33.36	0.757	18.81	0.669	18.25	**0.626**	**15.83**	0.648
crown	81.75	0.773	32.52	0.631	19.31	0.614	18.90	**0.550**	**12.61**	0.564
forehead	70.64	0.759	29.29	0.572	18.67	0.570	18.84	**0.495**	**11.48**	0.510
left eye	66.13	0.809	27.84	0.586	17.87	0.540	17.47	**0.481**	**12.22**	0.508
left leg	70.32	0.800	34.53	0.573	26.03	0.585	24.78	**0.508**	**22.42**	0.537
left wing	68.53	0.809	34.98	0.714	25.40	0.609	24.48	**0.572**	**22.76**	0.612
nape	80.72	0.807	32.05	0.675	18.27	0.613	17.97	**0.566**	**13.76**	0.589
right eye	54.14	0.810	28.53	0.587	17.66	0.533	17.21	**0.478**	**12.01**	0.510
right leg	74.57	0.773	33.23	0.569	24.50	0.583	24.20	**0.505**	**22.36**	0.535
right wing	68.99	0.859	32.28	0.698	23.43	0.592	22.84	**0.561**	**20.00**	0.598
tail	67.42	0.635	42.52	0.591	28.97	0.617	27.52	**0.533**	**22.03**	0.551
throat	80.34	0.792	33.05	0.641	19.29	0.596	18.70	**0.527**	**13.24**	0.554
mean	69.98	0.792	32.37	0.641	21.04	0.602	20.57	**0.540**	**16.29**	0.565

Table 1: **Shape & texture disentanglement.** MixNMatch outperforms strong baselines in terms of both shape or texture disentanglement for all keypoints. (c) is code mode, (f) is feature mode.

4. Shape & texture disentanglement evaluation

We first elaborate on how we evaluate texture disentanglement in Sec. 4.2 of the main paper. Recall that our goal is to take texture and background (codes c, b) from image A, shape and pose (codes p, z) from image B to generate new image C. In order to measure how well texture information is disentangled and preserved in generated image C, we first calculate 50 RGB cluster centers among 50,000 randomly sampled pixels from 1000 images (50 pixels per image) from the CUB dataset [6]. We then fire our pretrained keypoint detector, and crop a 16x16 patch centered on each keypoint from images A and C. For each patch, we compute its histogram representation by assigning each pixel to one of the color centers. Finally, we calculate the χ^2-distance between the L1-normalized color histograms of the patch in image A and corresponding patch in image C. Since images A and B can have different poses and hence occluded parts, we only consider keypoints visible in both images.

Next, in Table 1, we evaluate shape and texture disentanglement for all 15 keypoints. MixNMatch consistently outperforms the baselines for all keypoints. Our feature mode has the best performance for shape disentanglement due to its ability of preserving instance-specific shape and pose details. Our code mode model has the best performance for the texture disentanglement. One reason that the feature mode texture disentanglement result is slightly worse than that of code mode is because MixNMatch in feature mode can sometimes generate suboptimal masks (due to very similar background and object texture in the shape and pose reference images), leading to incomplete image generations.

5. Additional results

In Fig. 1 we encode the z, b, p, c codes from the two real images (first and last columns), linearly interpolate each code, and generate the interpolated images. MixNMatch produces perceptually smooth transitions for each factor, which again suggests that it has learned a highly disentangled latent space [2].

Figs. 2, 3, and 4 show additional disentanglement results of varying each factor for CUB, Dogs and Cars, respectively. These results supplement Fig. 4 from the main paper. In each sub-figure, images in the red boxes are real and we only change one factor indicated in the top left corner for generating the new images.

6. Video results

Finally, we include two videos demonstrating the disentanglement learned by MixNMatch. In MixNMatch.mp4, the four reference images on the top are real images which provide the four factors (background, shape, texture, and pose, respectively). The generated image is shown at the bottom. Each time we change different real reference image(s) and smoothly translate the corresponding factor.

We also animate an object in a still image according to the movement of a different object from a reference video. In the two img2gif files, the frames from the reference video on the top is used to extract the z vector to control object pose and location. On the left, we have a reference image from which shape, background, and texture (p, b, c) information are extracted. These factors are combined by MixNMatch to generate the new images at the bottom.

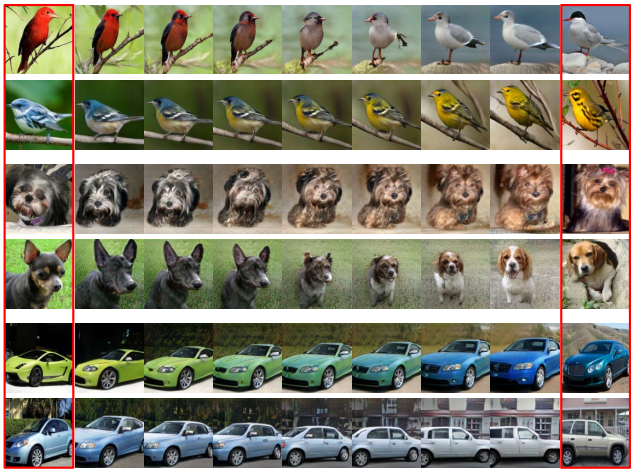


Figure 1: **Latent code interpolation.** Images in the red boxes are real, and intermediate images are generated by linearly interpolating codes predicted by our encoders.

Notice how our generated bird follows the pose of the reference video bird well – e.g., it turns around and lifts its head at the end. These results clearly indicate that our model can correctly disentangle pose information from the real images. Since MixNMatch is not trained on any video data and does not use any temporal information, the generated video can be a bit sensitive and unstable in terms of the bird's shape/size. Still, overall, each generated frame captures the factors from the respective image/video-frame very well to produce a realistic image with the corresponding properties.

References

[1] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. Improved training of wasserstein gans. In *NeurIPS*, 2017.

[2] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. In *CVPR*, 2019.

[3] Hadi Kazemi, Seyed Mehdi Iranmanesh, and Nasser M. Nasrabadi. Style and content disentanglement in generative adversarial networks. In *WACV*, 2018.

[4] Zhixin Shu, Mihir Sahasrabudhe, Riza Alp Güler, Dimitris Samaras, Nikos Paragios, and Iasonas Kokkinos. Deforming autoencoders: Unsupervised disentangling of shape and appearance. In *ECCV*, 2018.

[5] Krishna Kumar Singh, Utkarsh Ojha, and Yong Jae Lee. FineGAN: Unsupervised hierarchical disentanglement for fine-grained object generation and discovery. In *CVPR*, 2019.

[6] Catherine Wah, Steve Branson, Peter Welinder, Pietro Perona, and Serge Belongie. The Caltech-UCSD Birds-200-2011 Dataset. Technical Report CNS-TR-2011-001, 2011.

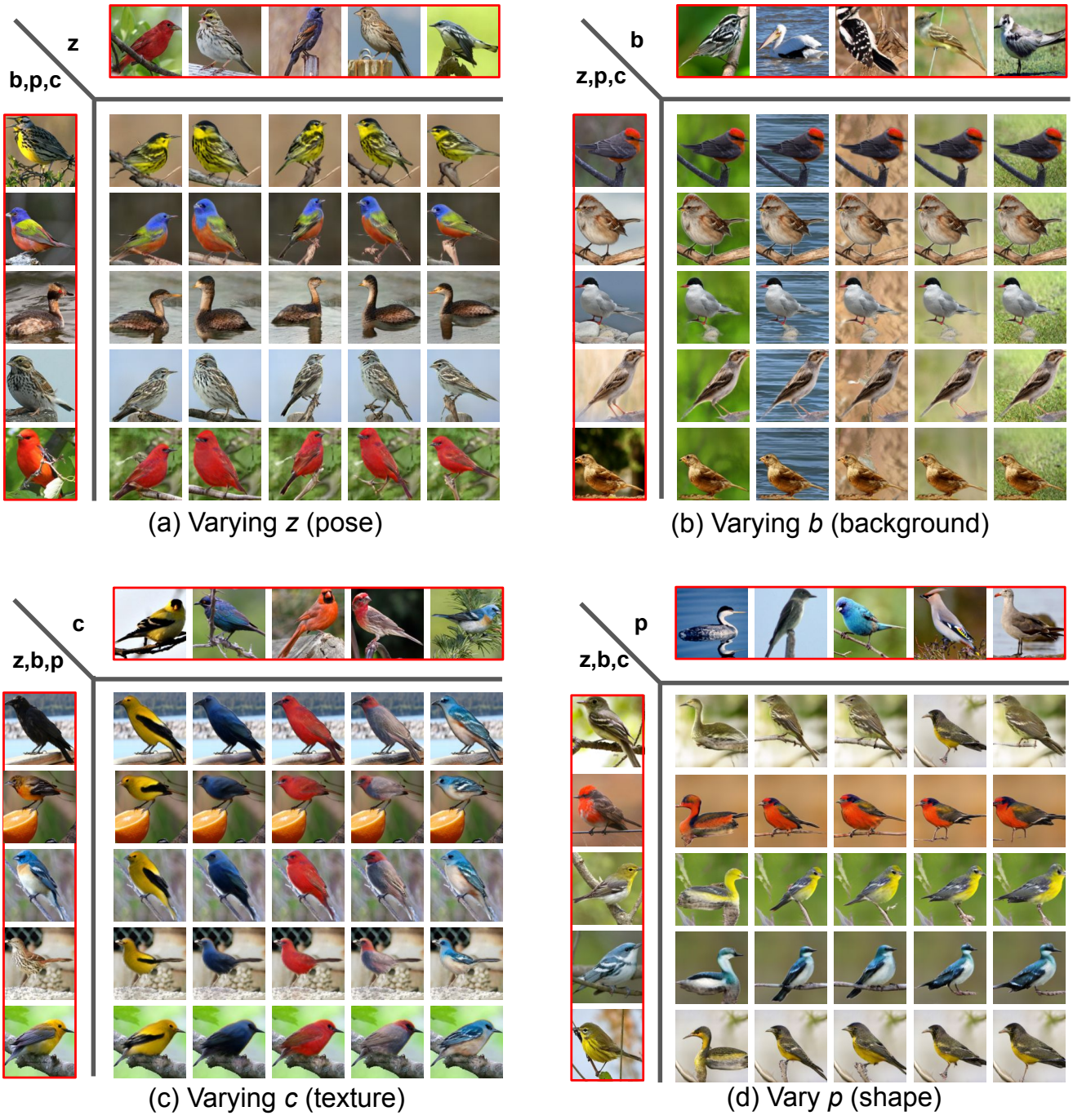
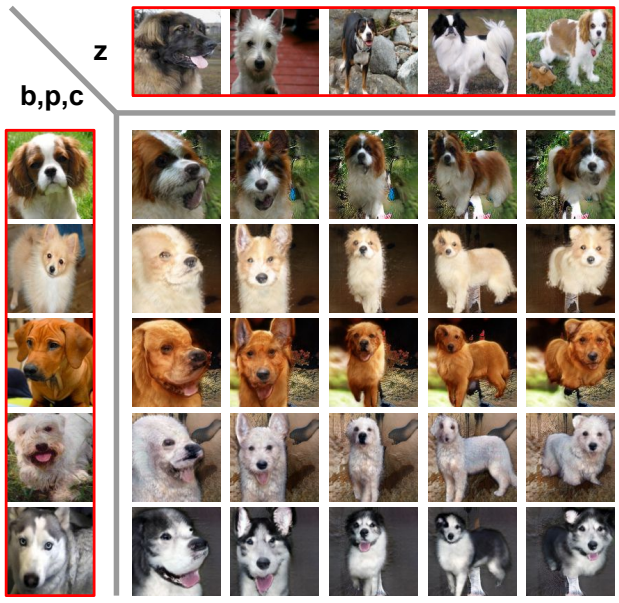
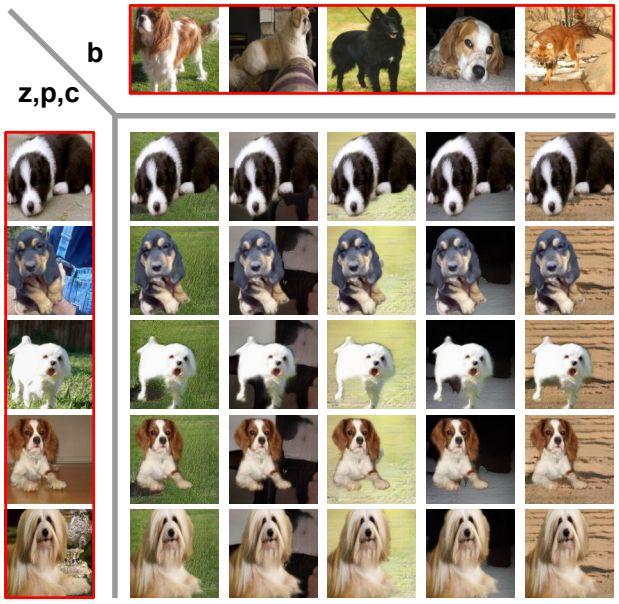


Figure 2: **Varying a single factor.** Real images are indicated with red boxes. For (a-d), the reference images on the left/top provide three/one factors. The center 5x5 images are generations.

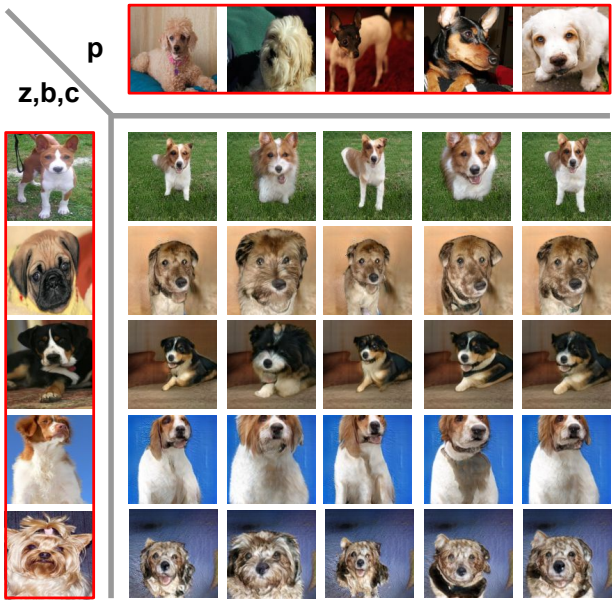


(a) Varying *z* (pose)



(b) Varying *b* (background)

(c) Varying *c* (texture)



(d) Vary *p* (shape)

Figure 3: **Varying a single factor.** Real images are indicated with red boxes. For (a-d), the reference images on the left/top provide three/one factors. The center 5x5 images are generations.

Figure 4: **Varying a single factor.** Real images are indicated with red boxes. For (a-d), the reference images on the left/top provide three/one factors. The center 5x5 images are generations.