LR-GAN: LAYERED RECURSIVE GENERATIVE ADVERSARIAL NETWORKS FOR IMAGE GENERATION

Jianwei Yang*
Virginia Tech
Blacksburg, VA
jw2yang@vt.edu

Anitha Kannan
Facebook AI Research
Menlo Park, CA
akannan@fb.com

Dhruv Batra* and Devi Parikh*
Georgia Institute of Technology
Atlanta, GA
{dbatra, parikh}@gatech.edu

ABSTRACT

We present LR-GAN: an adversarial image generation model which takes scene structure and context into account. Unlike previous generative adversarial networks (GANs), the proposed GAN learns to generate image background and foregrounds separately and recursively, and stitch the foregrounds on the backgrounds in a contextually relevant manner to produce a complete natural image. For each foreground, the model learns to generate its appearance, location and shape. The whole model is unsupervised, and is trained in an end-to-end manner with conventional gradient descent methods. The experiments demonstrate that LR-GAN can generate more natural images with objects that are more human recognizable than baseline GAN models.

1 INTRODUCTION

Generative adversarial networks (GANs) (Goodfellow et al., 2014) have shown significant promise as generative models for natural images. A flurry of recent work has proposed improvements over the original GAN work for image generation (Radford et al., 2015; Denton et al., 2015; Salimans et al., 2016; Chen et al., 2016; Zhu et al., 2016; Zhao et al., 2016), multi-stage image generation including part-based models (Im et al., 2016; Kwak & Zhang, 2016), image generation conditioned on input text or attributes (Mansimov et al., 2015; Reed et al., 2016b;a), image generation based on 3D structure (Wang & Gupta, 2016), and even video generation (Vondrick et al., 2016).

While the holistic 'gist' of images generated by these approaches is beginning to look natural, there is clearly a long way to go. For instance, the foreground objects in these images tend to be deformed, blended into the background, and not look realistic or recognizable.

One fundamental limitation of these methods is that they attempt to generate images without modeling that images are 2D projections of a 3D visual world, which has a lot of structure in it. This manifests as structure in the 2D images that capture this world. One example of this structure is that images tend to have a background, and foreground objects are placed in this background in contextually relevant ways.

We develop a GAN model that explicitly encodes this structure. Our proposed model generates images in a recursive fashion: it firsts generates a background, and then conditioned on the background generates a foreground along with a mask and an affine transformation that together define how the background and foreground should be composed to obtain a complete image. Conditioned on this composite image, a second foreground (and an associated mask and affine transformation) are generated, and so on. As a byproduct in the course of recursive image generation, our approach generates foreground-background masks in a completely unsupervised way, without access to *any* object masks for training. Note that decomposing a scene into foreground-background layers is a classical ill-posed problem in computer vision. By explicitly factorizing appearance and transformation, LR-GAN encodes natural priors about foreground objects in images that the same foreground can be 'pasted' under different affine transformations. We show that the absence of these object priors result in degenerate foreground-background decompositions.

*Work was done while visiting Facebook AI Research.

We show qualitatively (via samples) and quantitatively (via human studies on Amazon Mechanical Turk) that LR-GANl generates images that globally look natural *and* contain object structures in them that are realistic and recognizable by humans as semantic entities We find that LR-GAN generates foreground objects that are contextually relevant to the generated background (sheep on grass, airplanes in skies, ships in water, cars on streets, etc.). We propose a new quantitative measure to evaluate the quality of generated images. By this metric, as well as existing metrics in literature, our approach outperforms existing GAN approaches. We show that LR-GAN is capable of recursively generating multiple foreground entities in an image. We evaluate our approach on a synthetic MNIST dataset containing two digits, CIFAR, and CUB (birds).

2 Related Work

Early work in parametric texture synthesis was based on a set of hand-crafted features (Portilla & Simoncelli, 2000). Recent improvements in image generation using deep neural networks mainly fall into one of the two stochastic models: variational autoencoders (VAEs) (Kingma et al., 2016) and generative adversarial networks (Goodfellow et al., 2014). VAEs pair a top-down probabilistic generative model with a bottom up recognition network for amortized probabilistic inference. Both networks are jointly trained to maximize a variational lower bound on the data likelihood. GANs consist of a generator and a discriminator in a minmax game with the generator aiming to fool the discriminator with its samples with the latter aiming to not get fooled.

Sequential models have been pivotal for improved image generation using variational autoencoders: DRAW (Gregor et al., 2015) uses attention based recurrence conditioning on the canvas drawn so far. In (Eslami et al., 2016), a recurrent generative model that draws one object at a time to the canvas was used as the decoder in VAE. These methods are yet to show scalability to natural images. Early compelling results using GANs used sequential coarse-to-fine multiscale generation and class-conditioning Denton et al. (2015). Since then, improved training schemes (Salimans et al., 2016) and better convolutional structure has improved the generation results Radford et al. (2015) using GAN. PixelRNN (van den Oord et al., 2016) is also recently proposed that sequentially generates pixels, one at a time along the two spatial dimensions.

In this paper, we combine the merits of sequential generation with the flexibility of GAN. Our model for sequential generation imbibes the recursive structure that more naturally mimics object composition by inferring three components: appearance, shape, and transformation. We show that the resulting images are both qualitatively and quantitatively better. The closely related work combining recursive structure with GAN is that of Im et al. (2016) but it does not explicitly model object composition and follows a similar paradigm as by Gregor et al. (2015). Another closely related is the work of Kwak & Zhang (2016). It combines recursive structure and alpha blending. However, our work differs in three main ways: (1) We have explicit generators for foreground transformations, that provide significant advantage for natural images, as evidenced by our ablation studies. (2) Our shapes generator is separate from the appearance generator. This factored representation allows more flexibility in the generated scenes. (3) Our recursive framework generates subsequent objects conditioned on the generated canvas *and* previously generated object. This allows for explicit contextual modeling among generated elements in the scene. See Figure 17 for multiple contextually relevant foregrounds generated for the same background, or Figure 4 for meaningful placement of two MNIST digits relative to each.

Models that provide supervision to image generation using conditioning variables have also been proposed: Style/Structure GANs (Wang & Gupta, 2016) learn separate generative models for style and structure that are then composed to generate images. In Reed et al. (2016a), GAN based image generation is conditioned on text and the region in the image where the text manifests, specified during training via keypoints or bounding boxes. While not the focus of our work, the model proposed in this paper can be easily extended to take into account these forms of supervision.

3 Preliminaries

3.1 Generative Adversarial Networks

Generative Adversarial Networks (GANs) consist of a generator G and a discriminator D that are simultaneously trained with competing goals: The generator G is trained to generate samples that

can 'fool' a discriminator D, while the discriminator is trained to classify its inputs as either real (coming from the training dataset) or as fake (coming from the samples of G). This competition leads to a minmax formulation with a value function given by

$$\min_{\theta_G}\max_{\theta_D}\Big(\mathrm{E}_{\boldsymbol{x}\sim p_{data}(x)}[\log(D(\boldsymbol{x};\theta_D))]+\mathrm{E}_{\boldsymbol{z}\sim p_z(z)}[\log(1-D(G(\boldsymbol{z};\theta_G);\theta_D))]\Big), \tag{1}$$

where $\boldsymbol{z}$ is a random sample from a standard multivariate Gaussian or a uniform distribution $p_z(z)$, $G(\boldsymbol{z};\theta_G)$ maps $\boldsymbol{z}$ to the data space, $D(\boldsymbol{x})$ is the discriminator's probability estimate that $\boldsymbol{x}$ is real.

The advantage of the GAN formulation is that it lacks an explicit loss function and instead uses the discriminator to optimize the generative model. The discriminator, in turn, only cares whether the sample it receives is on the data manifold, and not whether it exactly matches a particular training example (as opposed to losses such as MSE). Hence, the discriminator provides a gradient signal *only* when the generated samples do not lie on the data manifold so that the generator can readjust its parameters accordingly. This form of training a generative model enables learning the data manifold of the training set and not just optimizing to reconstruct the dataset, as in autoencoder and it variants. This makes GAN attractive for learning rich generative models of multimodal data such as images.

While the GAN framework is largely agnostic to the choice of G and D, it is clear that generative models with the 'right' inductive biases will be more effective in learning from the gradient information (Denton et al., 2015; Im et al., 2016; Gregor et al., 2015; Reed et al., 2016a; Yan et al., 2015). With this motivation, we propose a generator that models image generation via a recurrent process – in each time step of the recurrence, an object with its own appearance and shape is generated and warped to compose an image in layers. We provide details of the the full model in §4.

3.2 Layered structure of image

An image taken from 3D world is typically with layered structure. One way of representing an image layer is by its appearance and shape. As an example, an image $\boldsymbol{x}$ with two layers, foreground $\boldsymbol{f}$ and background $\boldsymbol{b}$ is described through

$$\boldsymbol{x}=\boldsymbol{f}\odot\boldsymbol{m}+\boldsymbol{b}\odot(1-\boldsymbol{m}), \tag{2}$$

where $\boldsymbol{m}$ is the mask depicting the shapes of image layers, and $\odot$ the element wise multiplication operator. Some existing methods assume the access to the shape of the object either during training (Isola & Liu, 2013) or *both* at train and test time (Reed et al., 2016a; Yan et al., 2015). Representing image in layered structure is even straightforward for video with moving objects (Darrell & Pentland, 1991; Wang & Adelson, 1994; Kannan et al., 2005). (Vondrick et al., 2016) generates videos by separately generating a fixed background and moving foregrounds. At the most recent, Kwak & Zhang (2016) attempts to generate layered images with appearance and shape without extra priors.

Another stream is modeling the layered structure with object appearance and pose as

$$\boldsymbol{x}=ST(\boldsymbol{f},\boldsymbol{a})+\boldsymbol{b}, \tag{3}$$

where $\boldsymbol{a}$ is the affine transformation representing scale, location and rotation; ST is the spatial transformation operator. Several works fall into this group Roux et al. (2011); Huang & Murphy (2015); Eslami et al. (2016). In (Huang & Murphy, 2015), images are decomposed into layers of objects with specific poses in a variational autoencoder framework, while the number of objects is adaptively estimated in (Eslami et al., 2016).

In this paper, we simultaneously model all three dominant factors of variation: appearance $\boldsymbol{f}$, shape $\boldsymbol{m}$ and pose $\boldsymbol{a}$. Our generative model for images uses a layered composition that takes into account all these three factors for image generation. We will elaborate it in the following section.

4 Layered Recursive GAN (LR-GAN)

The basic structure of LR-GAN is similar to GAN: it consists of a discriminator and a generator that are simultaneously trained using the minmax formulation of GAN, as described in §.3.1. The key innovation of our work is the layered recursive generator, which is what we describe in this section.

The generator in LR-GAN is recursive in that the image is constructed recursively using a recurrent network. Layered in that each recursive step composes an object layer that is 'pasted' on the image

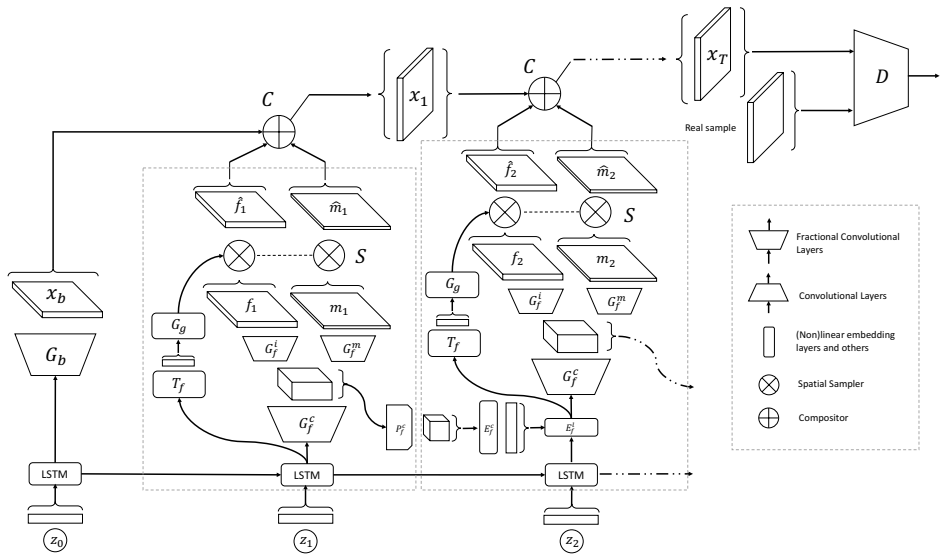

Figure 1: Layered Recursive GAN (LR-GAN) architecture, unfolded it to three timesteps.

generated so far via §3.2. Each object layer t is parameterized by the following three constituents – 'canonical' appearance $\boldsymbol{f}_t$, shape (or mask) $\boldsymbol{m}_t$, and affine transformation parameters $\boldsymbol{a}_t$ for warping the object before pasting in the image composition.

Fig. 1 shows the architecture of the LR-GAN with the generator architecture unrolled for generating background $\boldsymbol{x}_0$ ($\doteq \boldsymbol{x}_b$) and next two object layers $\boldsymbol{x}_1, \boldsymbol{x}_2$. At each time step t, the generator composes the next image $\boldsymbol{x}_t$ via the following recursive computation:

$$\boldsymbol{x}_t = \underbrace{ST(\boldsymbol{m}_t, \boldsymbol{a}_t)}_{\text{affine transformed mask}} \odot \underbrace{ST(\boldsymbol{f}_t, \boldsymbol{a}_t)}_{\text{affine transformed appearance}} + \underbrace{(1 - ST(\boldsymbol{m}_t, \boldsymbol{a}_t)) \odot \boldsymbol{x}_{t-1}}_{\text{pasting on image composed so far}}, \quad \forall t \in [1, T] \quad (4)$$

where $ST(\diamond, \boldsymbol{a}_t)$ is a spatial transformer layer (Jaderberg et al., 2015) that outputs the affine transformed version of $\diamond$ with $\boldsymbol{a}_t$ indicating parameters of the affine transformation.

Note that our proposed model has an explicit transformation variable $\boldsymbol{a}_t$ that is used to warp the object. This enables learning a canonical object representation that can be re-used to generate scenes where the object occurs as mere transformations of it, such as different scales or rotations. By factorizing the transformation and appearance, the object generator can focus on separately capturing regularities in appearance and shape that 'make up' an object. We will demonstrate in our experiments that removing this factorization from the model leads to it spending its capacity in variability that may not solely be about the object. We study this characteristic in detail in Section 5.2. As a teaser, see the comparative Figure 3 comparing learned object appearance-shape decomposition when LR-GAN is trained with and without explicit modeling of transformations.

4.1 Details of Generator architecture

Fig. 1 shows our LR-GAN architecture in detail – we use different shapes to indicate different kinds of layers (convolutional, fractional convolutional, (non)linear, etc), as indicated by the legend. Our model consists of two main pieces – a background generator G_b and a foreground generator G_f. G_b and G_f do not share parameters with each other. G_b computation happens only once, while G_f is recurrent over time, i.e., all object generators share the same parameters. In the following, we will introduce each module and connections between them.

Temporal Connection. LR-GAN has two kinds of temporal connections – informally speaking, one on 'top' and one on 'bottom'. The 'top' connections perform the act of sequentially 'pasting' object layers (Eqn. 4). As shown, we use a pooling layer P_f^c and a fully-connected layer E_f^c to extract the information from previous generated object response map. By this way, the model is able to know

what object has been generated. The 'bottom' connections are a unique feature of our model – an LSTM on the noise vectors $\boldsymbol{z}_0, \boldsymbol{z}_1, \boldsymbol{z}_2$. Intuitively, this noise-vector-LSTM provides information to the foreground generator about what else has been generated in past.

Background Generator. The background generator G_b is purposely kept simple. It takes the hidden state of noise-vector-LSTM h_l^0 as the input and passes it to a number of fractional convolutional layers (also called 'deconvolution' layer in some papers). Implementation details such as the number of layers, non-linearities *etc.* can be found in the Appendix. The output of background generator $\boldsymbol{x}_b$ will be used as the canvas for the following generated foregrounds.

Foreground Generator. Recall that G_f generates a canonical appearance, a mask, and an affine transformation for both. Correspondingly, G_f consists of three sub-modules, G_f^c, which is a common 'trunk' whose outputs are shared by G_f^i and G_f^m. G_f^i is used to generate the foreground appearance $\boldsymbol{f}_t$, while G_f^m generates the mask $\boldsymbol{m}_t$ for the foreground. All three sub-modules consists of one or more fractional convolutional layers combined with batch-normalization and nonlinear layers. The generated foreground and mask have the same spatial size as the background. The mask has just one channel, whose values are range in $(0, 1)$ after being passed through a sigmoid layer.

Spatial Transformer. To spatially transform the appearance and mask, we firstly need to estimate the transformation matrix. As in Jaderberg et al. (2015), we predict the transformation matrix with a linear layer T_f that has six-dimensional outputs. Then based on the predicted transformation parameters, we use a grid generator G_g to generate the corresponding sampling coordinates in the input for each location at the output. The generated foreground appearance and mask share the same transformation matrix, and thus the same sampling grid. Given the grid, the sampler S will simultaneously sample the $\boldsymbol{f}_t$ and $\boldsymbol{m}_t$ to obtain $\hat{\boldsymbol{f}}_t$ and $\hat{\boldsymbol{m}}_t$, respectively. Different from Jaderberg et al. (2015), our sampler here normally performs downsampling, since the the foreground typically has smaller size than the background. Pixels in $\hat{\boldsymbol{f}}_t$ and $\hat{\boldsymbol{m}}_t$ that are from outside the extent of $\boldsymbol{f}_t$ and $\boldsymbol{m}_t$ are set to zero. Finally, $\hat{\boldsymbol{f}}_t$ and $\hat{\boldsymbol{m}}_t$ are sent to the compositor C to combining the canvas $\boldsymbol{x}_{t-1}$ and $\hat{\boldsymbol{f}}_t$ through layered composition with weight given by $\hat{\boldsymbol{m}}_t$ (Eqn. 4).

Pseudo-code for our approach and detailed model configuration are provided in the Appendix.

4.2 New Evaluation Metrics

Several metrics have been proposed to evaluate GANs, such as Gaussian parzen window (Goodfellow et al., 2014), Generative Adversarial Metric (GAM) (Im et al., 2016) and 'Inception Score' (Salimans et al., 2016). The common goal is to measure the similarity between the generator distribution $P_g(\boldsymbol{x}) = G(\boldsymbol{z}; \theta_z)$ and the real data distribution $P(\boldsymbol{x})$. Most recently, 'Inception Score' has been used in several works (Salimans et al., 2016; Zhao et al., 2016). However, it can be easily fooled by generating centers of categories.

In addition to these metrics (which we report in our experiments), we present two new metrics based on the following intuition – a sufficient (but not necessary) condition for closeness of $P_g(\boldsymbol{x})$ and $P(\boldsymbol{x})$ is closeness of $P_g(\boldsymbol{x}|y)$ and $P(\boldsymbol{x}|y)$, i.e., generator and real distributions conditioned under some variable of interest y.

One way to obtain this variable of interest y is via human annotation. Specifically, given the samples from $P_g(\boldsymbol{x})$ and $P(\boldsymbol{x})$, we ask people to label the category of the samples. Note that such human annotation is often easier than comparing samples from the two distributions (e.g., because there isn't a 1:1 correspondence between samples to conduct forced-choice tests).

Now, we need to verify whether the generator and real distributions are similar in each category. Clearly, directly comparing the distributions $P_g(\boldsymbol{x}|y)$ and $P(\boldsymbol{x}|y)$ may be as difficult as comparing $P_g(\boldsymbol{x})$ and $P(\boldsymbol{x})$. Fortunately, we can use Bayes rule and compare $P_g(y|\boldsymbol{x})$ and $P(y|\boldsymbol{x})$, which is a much easier task. We can simply train a discriminative model (e.g., a convolutional neural network) on the samples from $P_g(\boldsymbol{x})$ and $P(\boldsymbol{x})$ (together with the human annotations about categories of these samples). With a slight abuse of notation, we use $P_g(y|\boldsymbol{x})$ and $P(y|\boldsymbol{x})$ to denote probability outputs from these two classifiers (trained on generated-samples vs trained on real-samples). We can then use these two classifiers to compute the following two evaluation metrics:

Adversarial Accuracy: Computes the classification accuracies achieved by these two classifiers on a validation set, which can be the training set or another set of real images.

Adversarial Divergence: Computes the KL divergence between $P_g(y|\boldsymbol{x})$ and $P(y|\boldsymbol{x})$. The lower the adversarial divergence, the closer two distributions are. The low bound for this metric is exactly zero, which means $P_g(y|\boldsymbol{x}) = P(y|\boldsymbol{x})$ for all samples in the validation set.

As discussed above, we need human efforts to label the generated samples. Fortunately, we can further simplify this. Based on the labels on training data, we split the training data into categories, and train one generator for each category. With all these generators, we generate samples of all categories. This strategy will be used in our experiments.

5 EXPERIMENT

To verify the effectiveness of our model, we conduct qualitative and quantitative evaluations on three datasets: 1) MNIST (LeCun et al., 1998); 2) CIFAR-10 Krizhevsky & Hinton (2009); 3) CUB-200 Welinder et al. (2010). To add variability to the MNIST images, we randomly scale (factor of 0.8 to 1.2) and rotate ($-\frac{\pi}{4}$ to $\frac{\pi}{4}$) the images and then add a random value of [0, 200] to each transformed image. The resultant images are 48×48. We clamp the pixel values at 255. Images are then rescaled to 32×32. Each image thus has a different background grayscale value and a different transformed digit as foreground. We call this the **MNIST-ONE** dataset (single digit on a gray background). We also generate **MNIST-TWO** containing two digits on a grayscale background. We randomly select two images of digits and perform similar transformations as described above. We put one on the left and the other on the right side of a 78×78 gray background. We resize the whole image to 64×64.

We develop LR-GAN based on open source code[1]. We assume the number and maximal size of objects is known. Therefore, for MNIST-ONE, MNIST-TWO, CIFAR-10, and CUB-200, our model has 2, 3, 2, and 2 timesteps respectively. We set the minimal allowed scaling values in affine transforamtion to be 1.2 for all datasets except for MNIST-TWO, which is set to 2 (objects are smaller in MNIST-TWO[2]). In LR-GAN, the background generator and foreground generator have similar architectures. One difference is that the number of channels in the background generator is half of the one in the foreground generator. We compare our results to that of DCGAN (Radford et al., 2015). Note that LR-GAN with a timestep of 1 corresponds to DCGAN. This allows us to run controlled experiments. In both generator and discriminator, all the (fractional) convolutional layers have 4×4 filter size with stride 2. As a result, the number of layers in the generator and discriminator automatically adapt to the size of training images. Please see the Appendix (Section 6.2) for details about the configurations. We use three metrics for quantitative evaluation, including 'Inception Score' (Salimans et al., 2016) and the proposed Adversarial Accuracy, Adversarial Divergence.

5.1 QUALITATIVE RESULTS

In Figure 2, we show the generated samples on CIFAR-10 and CUB-200. MNIST results are shown in the next subsection. As we can see from the images, the compositional nature of our model results in the images being free of blending artifacts. For CIFAR-10, we can see the horses and cars with clear shapes. For CUB-200, the birds shape tends to be even sharper. More results are shown in the Appendix.

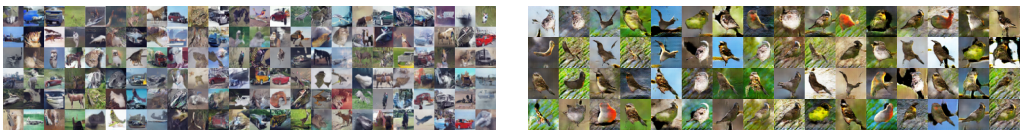

Figure 2: Generated samples for CIFAR-10 dataset (left) and CUB-200 (right) using our model.

5.2 IMPORTANCE OF TRANSFORMATION

We verify the importance of the transformations in our model. As seen in Figure 3, removing the spatial transformer results in our model finding a degenerate composition that is not layered. Similar results are also found on CUB-200 and CIFAR-10 in Figure 20 in the Appendix.

[1] https://github.com/soumith/dcgan.torch

[2] Scale corresponds to the size of the target canvas – larger the scale, larger the canvas, and smaller the relative size of the object in the canvas.

Figure 3: Comparison of intermediate output of our model without spatial transformation (left) and with spatial transformation (right). For each model from left to right, sets of 6 columns are generated backgrounds, generated foregrounds, generated masks, and the final composed images

5.3 MULTI-DIGITS MNIST

Figure 4 shows the generation results, along with intermediate results, from our model trained on the **MNIST-TWO** dataset. We can see that the model is able to generate background and the two foreground objects separately. The foreground generators tends to generate a single digit at each timestep. It also captures the context information from the previous time steps. When the first digit is placed to the left, the second one tends to be placed on the right, and vice versa. Several more results (including human studies) on MNIST-ONE can be found in the Appendix (Section 6.3).

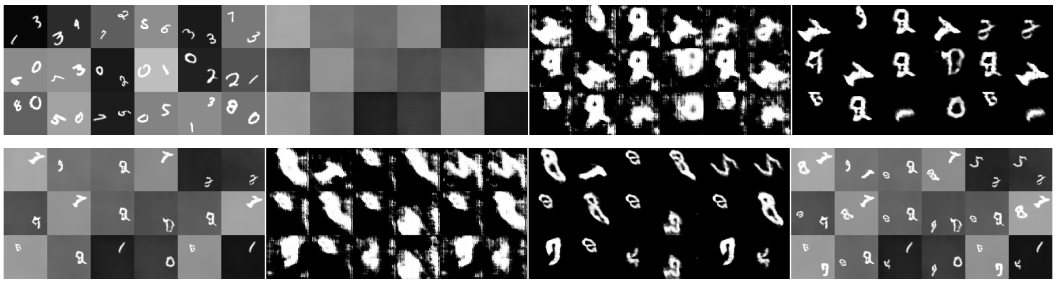

Figure 4: (Intermediate) outputs of our model on MNIST-TWO. Top left to bottom right (row major): training images, $\boldsymbol{x}_b$, $\boldsymbol{f}_1$, $\boldsymbol{m}_1$, $\boldsymbol{x}_1$, $\boldsymbol{f}_2$, $\boldsymbol{m}_2$, and $\boldsymbol{x}_2$.

5.4 CUB-200

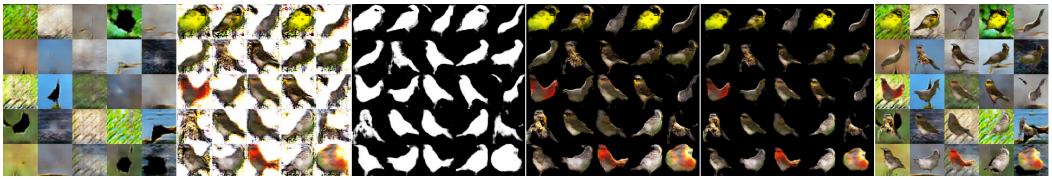

Figure 5: From left to right, set of 5 columns are generated backgrounds, generated foregrounds, generated masks, foreground carved out by masks, carved foregrounds after spatial transformation, and final composed images.

Here, we study the effectiveness of our model trained on the CUB-200 dataset of birds. In Figure 5, we show a random set of generated images, along with the intermediate outputs of the model. While being completely unsupervised, the model, for a large fraction of the samples, is able to successfully factor out the foreground and the background. This is also evident from the generated masks that are predominantly bird-like.

We do a comparative study on Amazon Mechanical Turk (AMT) between DCGAN and our approach to quantify relative visual quality of the generated images. For this task, we first generated 1000 samples from both the models. Then, we performed perfect matching between the two image sets using the Hungarian algorithm on $L2$ norm in the pixel space as the distance. This resulted in 1000 image pairs. For each image pair, 9 judges are asked to choose the one that is more realistic. Based on majority vote, our generated images are selected 68.4% times, compared with 31.6% times for DCGAN. This demonstrates that our model has generated more realistic images than DCGAN. We can attribute this difference to our model's ability to generate foreground pixels separately from the background pixels, enabling stronger edge cues. See Figure 6.

Figure 6: Top three rows are samples generated by DCGAN; Bottom three rows are samples generated by our model. These are paired according to the perfect matching (see text for details).

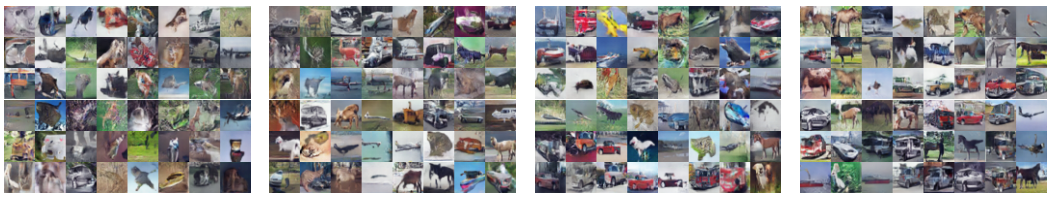

Figure 7: Top three rows are samples generated by DCGAN; Bottom three rows are from our method. From left to right: increasing human recognizability.

5.5 CIFAR-10

We now qualitatively and quantitatively evaluate our model on CIFAR-10, which contains multiple object categories and also various backgrounds.

Comparison of image generation quality: We conduct AMT studies to compare the fidelity of image generation. Towards this goal, we generate 1000 images from DCGAN and from our model. We ask five judges to label each image to either belong to one of the 10 categories or as 'non recognizable' or 'recognizable but not belonging to the listed categories'. We then assign each image a quality score between [0,5] that captures the number of judges that agree with the majority choice. Figure 7 shows the images generated by both approaches, ordered by increasing quality score. We merged images with quality level 0 (all judges said non-recognizable) and 1 together, and similarly images at levels 4 and 5. Visually, the generated samples by our method have clearer boundaries and object structures. We also computed the fraction of non-recognizable images: Our approach had a 10% absolute drop in non-recognizability rate (67.3% for ours vs. 77.7% for DCGAN). For reference, 11.4% of real CIFAR images were categorized as non-recognizable.

Quantitative evaluation on generators: We evaluate the generators based on three metrics: 1) 'Inception Score'; 2) Adversarial Accuracy; 3) Adversarial Divergence. We remove the top layer from the discriminator used in our model, and then append two fully connected layers on the top of it to obtain a classifier model. This classifier model is used across all three metrics. For 'Inception Score', we train the classifier using the training samples of CIFAR-10 based on the annotations, and then compute the score for generated samples from DCGAN and our model, respectively. To obtain the Adversarial Accuracy and Adversarial Divergence scores, we train one generator on each of 10 categories for DCGAN and our model, respectively. Then, we use these generators to generate samples for different categories. Given these generated samples, we train the classifiers for DCGAN model and ours. Along with the classifier trained on the real samples, we compute the Adversarial Accuracy and Adversarial Divergence on the training samples. In Table 1, we report the inception scores, adversarial accuracy and adversarial divergence for comparison. We can see that our model outperforms DCGAN across the board.

Quantitative evaluation on discriminators: We evaluate the discriminator as a extractor for deep representations. Specifically, we use the output of the last convolutional layer in the discriminator as features. We perform a 1-NN classification on the test set given the full training set. Cosine similarity is used as the metric. On the test set, our model achieves 62.09%$\pm$0.01% compared to DCGAN's 56.05%$\pm$0.02%.

Table 1: Quantitative comparison between DCGAN and our model on CIFAR-10.

Training Data	Real Images	DCGAN	Ours
Inception Score	7.23±0.09	5.69±0.07	6.11±0.06
Adversarial Accuracy	83.33±0.08	37.81±0.02	44.22 ±0.08
Adversarial Divergence	0	7.58±0.04	5.57±0.06

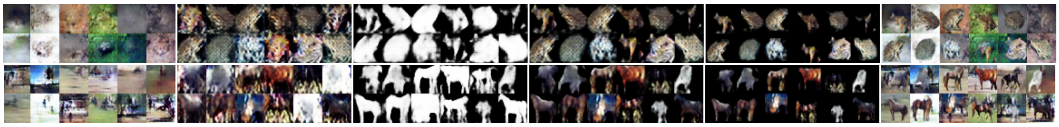

Figure 8: From left to right, similar layout to Figure 5. Top two rows are *frog* and bottom are *horse*.

Diversity in scenes: We also show the efficacy of our approach to generate diverse foregrounds conditioned on fixed background noise vector. The results in Figure 17 in Appendix showcase that the foreground generator generates objects that are compatible with the background. This shows that the model has captured contextual dependencies between the image layers.

Category specific models: The objects in CIFAR-10 exhibit huge variability in shapes. That can partly explain why some of the generated shapes are not as compelling. To test this hypothesis, we reuse the generators trained for each of 10 categories used on our metrics. Fig. 8 shows results for categories, 'frogs' and 'horses'. We can see that the model is now able to generate object-specific appearances and shapes, similar in vein to our results on the CUB-200 dataset. More results are available in the Appendix in Fig. 16.

As discussed above, training category-specific models helps generate crisper masks of the foreground object. Future work involves training a discriminator that tries to differentiate between these category-specific masks and the category-independent masks of our unsupervised model. This may encourage the unsupervised model to generate more object-like masks, for categories not covered in the supervised training data.

REFERENCES

Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. *arXiv preprint arXiv:1606.03657*, 2016.

Trevor Darrell and Alex Pentland. Robust estimation of a multi-layered motion representation. *IEEE Workshop on Visual Motion*, 1991.

Emily L Denton, Soumith Chintala, Rob Fergus, et al. Deep generative image models using a laplacian pyramid of adversarial networks. In *Advances in neural information processing systems*, pp. 1486–1494, 2015.

S. M. Ali Eslami, Nicolas Heess, Theophane Weber, Yuval Tassa, Koray Kavukcuoglu, and Geoffrey E. Hinton. Attend, infer, repeat: Fast scene understanding with generative models. *CoRR*, abs/1603.08575, 2016.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems*, pp. 2672–2680, 2014.

Karol Gregor, Ivo Danihelka, Alex Graves, Danilo Jimenez Rezende, and Daan Wierstra. Draw: A recurrent neural network for image generation. *arXiv preprint arXiv:1502.046239*, 2015.

Gary B. Huang, Manu Ramesh, Tamara Berg, and Erik Learned-Miller. Labeled faces in the wild: A database for studying face recognition in unconstrained environments. Technical Report 07-49, University of Massachusetts, Amherst, October 2007.

Jonathan Huang and Kevin Murphy. Efficient inference in occlusion-aware generative models of images. *CoRR*, abs/1511.06362, 2015.

Daniel Jiwoong Im, Chris Dongjoo Kim, Hui Jiang, and Roland Memisevic. Generating images with recurrent adversarial networks. *arXiv preprint arXiv:1602.05110*, 2016.

Phillip Isola and Ce Liu. Scene collaging: Analysis and synthesis of natural images with semantic layers. In *IEEE International Conference on Computer Vision*, pp. 3048–3055, 2013.

Max Jaderberg, Karen Simonyan, Andrew Zisserman, and koray kavukcuoglu. Spatial transformer networks. In *Advances in Neural Information Processing Systems 28*, pp. 2017–2025, 2015.

Anitha Kannan, Nebojsa Jojic, and Brendan Frey. Generative model for layers of appearance and deformation. *AISTATS*, 2005.

Diederik P Kingma, Tim Salimans, and Max Welling. Improving variational inference with inverse autoregressive flow. *arXiv preprint arXiv:1606.04934*, 2016.

Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. 2009.

Hanock Kwak and Byoung-Tak Zhang. Generating images part by part with composite generative adversarial networks. *arXiv preprint arXiv:1607.05387*, 2016.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Elman Mansimov, Emilio Parisotto, Jimmy Lei Ba, and Ruslan Salakhutdinov. Generating images from captions with attention. *arXiv preprint arXiv:1511.02793*, 2015.

Javier Portilla and Eero P Simoncelli. A parametric texture model based on joint statistics of complex wavelet coefficients. *International journal of computer vision*, 40(1):49–70, 2000.

Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

Scott Reed, Zeynep Akata, Santosh Mohan, Samuel Tenka, Bernt Schiele, and Honglak Lee. Learning what and where to draw. *arXiv preprint arXiv:1610.02454*, 2016a.

Scott Reed, Zeynep Akata, Xinchen Yan, Lajanugen Logeswaran, Bernt Schiele, and Honglak Lee. Generative adversarial text to image synthesis. *arXiv preprint arXiv:1605.05396*, 2016b.

Nicolas Le Roux, Nicolas Heess, Jamie Shotton, and John Winn. Learning a generative model of images by factoring appearance and shape. *Neural Computation*, 23:593–650, 2011.

Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. *arXiv preprint arXiv:1606.03498*, 2016.

Aäron van den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. Pixel recurrent neural networks. *CoRR*, abs/1601.06759, 2016.

Carl Vondrick, Hamed Pirsiavash, and Antonio Torralba. Generating videos with scene dynamics. *arXiv preprint arXiv:1609.02612*, 2016.

John Wang and Edward Adelson. Representing moving images with layers. *IEEE Transactions on Image Processing*, 1994.

Xiaolong Wang and Abhinav Gupta. Generative image modeling using style and structure adversarial networks. *arXiv preprint arXiv:1603.05631*, 2016.

P. Welinder, S. Branson, T. Mita, C. Wah, F. Schroff, S. Belongie, and P. Perona. Caltech-UCSD Birds 200. Technical Report CNS-TR-2010-001, California Institute of Technology, 2010.

Xinchen Yan, Jimei Yang, Kihyuk Sohn, and Honglak Lee. Attribute2image: Conditional image generation from visual attributes. *CoRR*, abs/1512.00570, 2015.

Junbo Zhao, Michael Mathieu, and Yann LeCun. Energy-based generative adversarial network. *arXiv preprint arXiv:1609.03126*, 2016.

Jun-Yan Zhu, Philipp Krähenbühl, Eli Shechtman, and Alexei A Efros. Generative visual manipulation on the natural image manifold. In *European Conference on Computer Vision*, pp. 597–613. Springer, 2016.

6 APPENDIX

6.1 ALGORITHM

Algo. 1 illustrates the generative process in our model. $g(\star)$ evaluates the function g at $\star$. $\circ$ is a composition operator that composes its operands so that $f \circ g(\star) = f(g(\star))$.

Algorithm 1 Stochastic Data Generation

$\boldsymbol{z}_0 \sim \mathcal{N}(0, I)$
$\boldsymbol{x}_0 = G_b(\boldsymbol{z}_0)$ ▷ background generator
$\boldsymbol{h}_l^0 \leftarrow \mathbf{0}$
$\boldsymbol{c}_l^0 \leftarrow \mathbf{0}$
for $t \in [1 \cdots T]$ **do**
  $\boldsymbol{z}_t \sim \mathcal{N}(0, I)$
  $\boldsymbol{h}_l^t, \boldsymbol{c}_l^t \leftarrow \text{LSTM}([\boldsymbol{z}_t, \boldsymbol{h}_l^{t-1}, \boldsymbol{c}_l^{t-1}])$ ▷ pass through LSTM
  if t = 1 **then**
    $\boldsymbol{y}_t \leftarrow \boldsymbol{h}_l^t$
  else
    $\boldsymbol{y}_t \leftarrow E_f^l([\boldsymbol{h}_l^t\ \boldsymbol{h}_f^{t-1}])$ ▷ pass through non-linear embedding layers E_f^l
  end if
  $\boldsymbol{s}_t \leftarrow G_f^c(\boldsymbol{y}_t)$ ▷ predict shared cube for G_f^i and G_f^m
  $a_t \leftarrow T_f(\boldsymbol{y}_t)$ ▷ object transformation
  $\boldsymbol{f}_t \leftarrow G_f^i(\boldsymbol{s}_t)$ ▷ generate object appearance
  $\boldsymbol{m}_t \leftarrow G_f^m(\boldsymbol{s}_t)$ ▷ generate object shape
  $\boldsymbol{h}_f^t \leftarrow E_f^c \circ P_f^c(\boldsymbol{s}_t)$ ▷ predict shared represenation embedding
  $\boldsymbol{x}_t \leftarrow ST(\boldsymbol{m}_t, \boldsymbol{a}_t) \odot ST(\boldsymbol{f}_t, \boldsymbol{a}_t) + (1 - ST(\boldsymbol{m}_t, \boldsymbol{a}_t)) \odot \boldsymbol{x}_{t-1}$
end for

6.2 MODEL CONFIGURATIONS

Table 2 lists the configuration of our model for different datasets. Based on the same notation used in (Zhao et al., 2016), the architectures for the different datasets are:

- MNIST-ONE: $\boldsymbol{G}_b$: (256)4c-(128)4c2s-(64)4c2s-(3)4c2s; $\boldsymbol{G}_f^c$: (512)4c-(256)4c2s-(128)4c2s; $\boldsymbol{G}_f^i$: (3)4c2s; $\boldsymbol{G}_f^m$: (1)4c2s; D: (64)4c2s-(128)4c2s-(256)4c2s-(256)4p4s-1
- MNIST-TWO: $\boldsymbol{G}_b$: (256)4c-(128)4c2s-(64)4c2s-(32)4c2s-(3)4c2s; $\boldsymbol{G}_f^c$: (512)4c-(256)4c2s-(128)4c2s-(64)4c2s; $\boldsymbol{G}_f^i$: (3)4c2s; $\boldsymbol{G}_f^m$: (1)4c2s;D: (64)4c2s-(128)4c2s-(256)4c2s-(512)4c2s-(512)4p4s-1
- CUB-200: $\boldsymbol{G}_b$: (512)4c-(256)4c2s-(128)4c2s-(64)4c2s-(3)4c2s; $\boldsymbol{G}_f^c$: (1024)4c-(512)4c2s-(256)4c2s-(128)4c2s; $\boldsymbol{G}_f^i$: (3)4c2s; $\boldsymbol{G}_f^m$: (1)4c2s;D: (128)4c2s-(256)4c2s-(512)4c2s-(1024)4c2s-(1024)4p4s-1
- CIFAR-10: $\boldsymbol{G}_b$: (256)4c-(128)4c2s-(64)4c2s-(3)4c2s; $\boldsymbol{G}_f^c$: (512)4c-(256)4c2s-(128)4c2s; $\boldsymbol{G}_f^i$: (3)4c2s; $\boldsymbol{G}_f^m$: (1)4c2s D: (64)4c2s-(128)4c2s-(256)4c2s-(256)4p4s-1

The generator in DCGAN is the same as $[\boldsymbol{G}_f^c, \boldsymbol{G}_f^i]$. The dimension of all hidden vectors is 100. Table 2 we also compare the number of parameters between DCGAN and our models. The numbers before '/' are our model, after '/' are DCGAN.

6.3 RESULTS ON MNIST-ONE

We now report results on MNIST-ONE. Fig. 9 shows intermediate outputs of our model after training. As we can see, our model can learn to disentangle the foreground from background nearly perfectly. Though initial values of the mask cover the range in (0, 1), after training, the masks are nearly binary and accurately carve out the digits from the generated foreground.

We also conduct human studies. For human studies, we generate 1,000 images using both our model and DCGAN. As references, we also include 1000 real images. Then we ask the users on Amazon Mechanical Turk (AMT) to label each image to be one of the digits (0-9). We also provide them an option 'non recognizable' in case the generated image does not seem to contain a digit. Each image was judged by 5 unique workers. If an image is recognized to be the same digit by all 5 users, it is assigned the highest level of quality. If a digit is not recognizable according to all users, it is assigned a quality level of 0. Fig. 10 (left) shows the number of images assigned to all six quality levels. As expected, the real images have many samples with high quality levels. Compared to DCGAN, our model generated more samples with high quality levels. In Fig. 10 (right), we show the number of images which are recognized as each of the digit categories (0-9). For qualitative comparison, we show examples images from each quality level (11). From left to right, the quality level increases from 0 to 5. As expected, the images with higher quality levels are more clear.

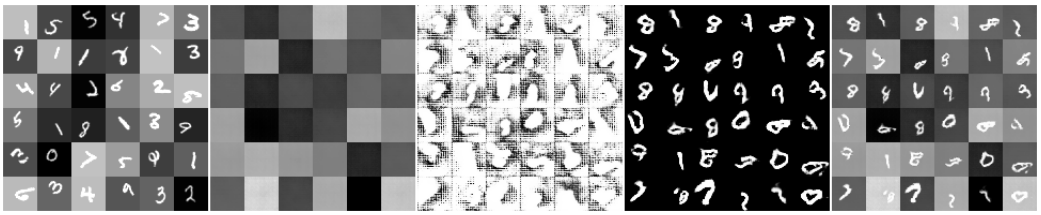

Figure 9: Outputs of our model on the MNIST-ONE dataset. From left to right: training images, backgrounds generated at the first timestep, foregrounds generated at the second timestep, masks generated at the second timestep, and composed images.

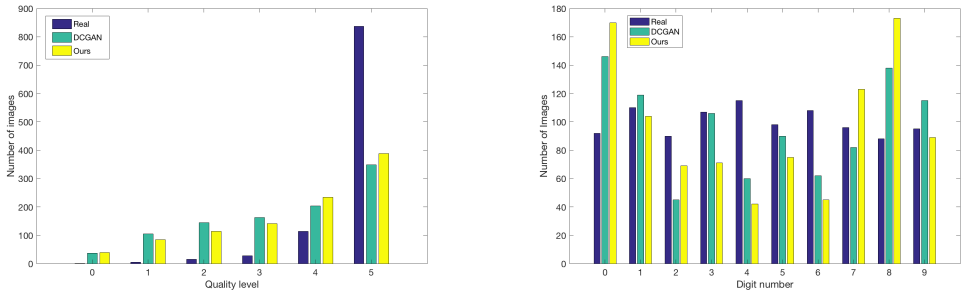

Figure 10: Left: distribution of quality level; Right: distribution of recognized digit categories.

For quantitative evaluation, we use three metrics as for CIFAR-10. To compute the inception score, as in (Zhao et al., 2016), we first train a classifier based on all training samples. Then, we use the classifier to obtain the probability distribution $P(y|\boldsymbol{x})$ and $P(y)$ for all generated samples from our model and DCGAN, respectively. After that, the expectation of KL divergence is computed using

Table 2: Model configurations on different datasets.

Dataset	MNIST-one	MNIST-two	CIFAR-10	CUB-200
Image Size	32	64	32	64
#Images	60,000	60,000	50,000	5,994
#Timesteps	2	3	2	2
#Parameters	5.25M/4.11M	7.53M/6.33M	5.26M/4.11M	27.3M/6.34M

Figure 11: Top three rows are samples generated by DCGAN. Bottom three rows are samples generated from our method. From left to right: increasing quality levels as determined via human studies (see text for details).

$E_{\boldsymbol{x}} KL(P(y|\boldsymbol{x})||P(y))$. To obtain the Adversarial Accuracy and Divergence scores, we first train 10 generators for 10 digits categories separately, and then use the generated samples to train a classifier. As shown in Table 3, our model has a higher inception score than DCGAN. Also, our model has a slightly higher adversarial accuracy than DCGAN, and lower adversarial divergence than DCGAN.

Table 3: Quantitative comparison on MNIST-ONE.

Training Data	Real Images	DCGAN	Ours
Inception Score	9.15±0.04	6.42±0.03	7.15±0.04
Adversarial Accuracy	95.22 ± 0.25	26.12 ± 0.07	26.61 ± 0.06
Adversarial Divergence Score	0	8.47 ± 0.03	8.39 ± 0.04

6.4 More results on CUB-200

We show more results on CUB-200 in Fig. 12. In particular, notice the crisp bird-like masks generated automatically by our approach in an unsupervised manner.

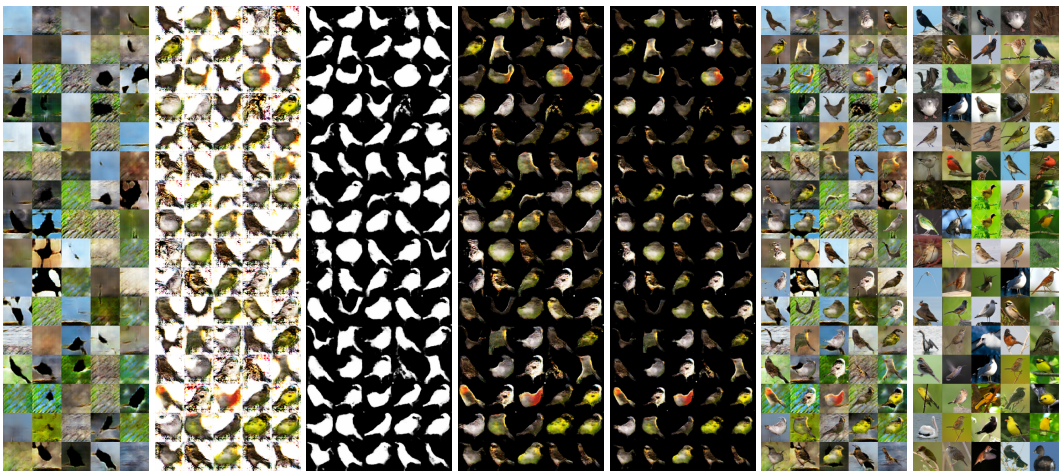

Figure 12: From left to right, each 5 columns are generated background, generated foreground, generated masks, foreground carved out by masks, carved foregrounds after spatial transformation. The sixth and seventh columns are final composed images and the nearest neighbor real images.

From the generated background images in Figure 12, we can find black holes with bird shapes. Ideally, our model is able to learn to generate foregrounds which can cover the holes so that the whole image looks real. However, since we set a minimal constraint on the object scaling factor (1.2 in our experiment), the bigger holes can not be completely covered for those bird shapes with larger scale. To verify this, we set the minimal scaling factor to 1.1, which allows to generate bigger foreground objects. The results are shown in Fig. 13. As expected, the holes in background images disappear.

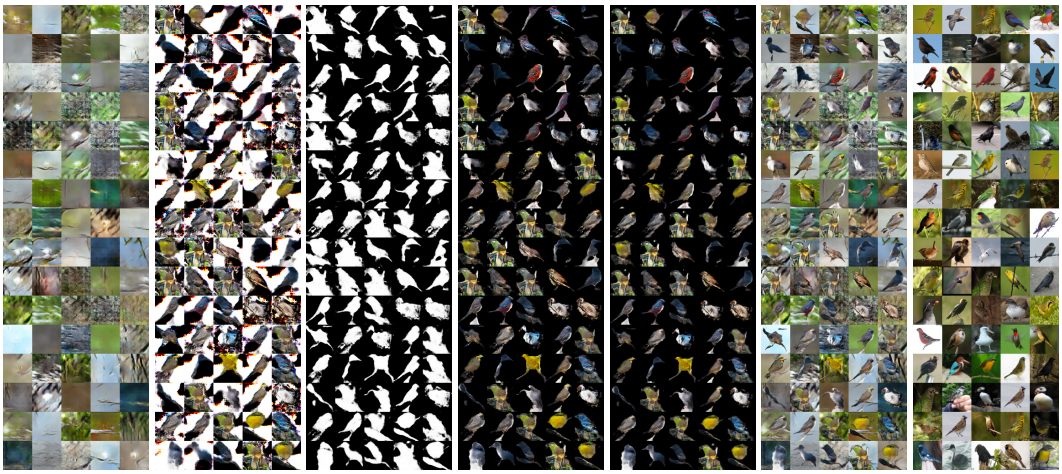

Figure 13: Experimental results on CUB-200 when setting minimal scale be 1.1. From left to right, images are shown in the same layout as Figure 12.

6.5 More results on CIFAR-10

6.5.1 Qualitative Results

In Fig. 14, we show more results on CIFAR-10. The last five columns also show the training images that are closest to the generated images (cosine similarity in pixel space). Note that our model does not seem to be memorizing the training data. Similar to CUB-200, we also change the minimal object scaling factor to 1.1, and obtained the results as shown in Fig. 15.

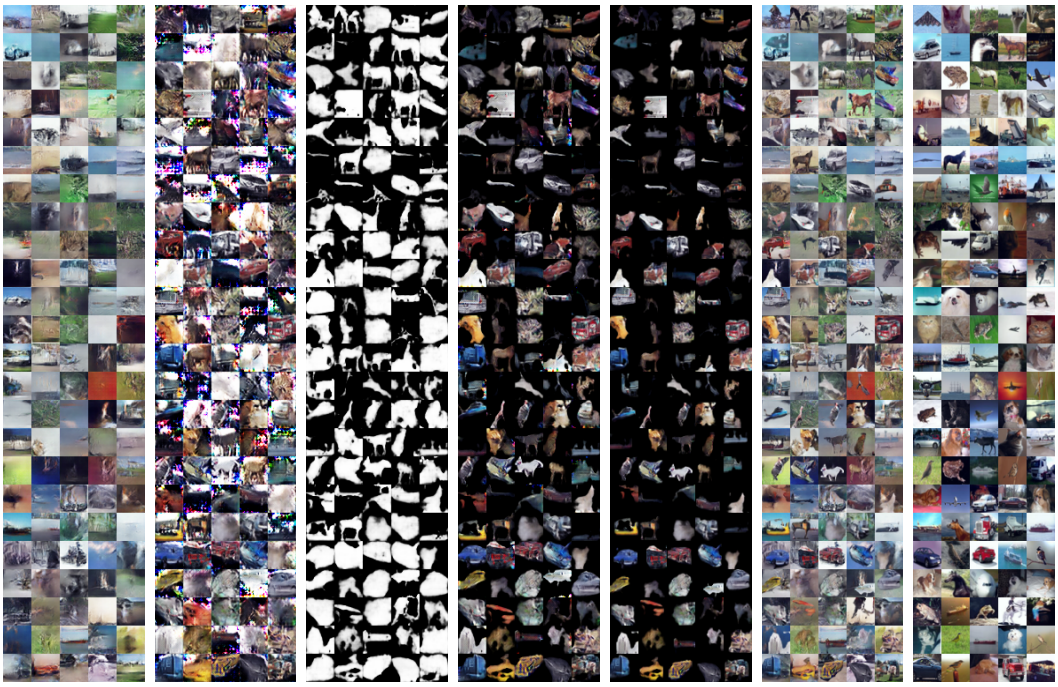

Figure 14: Sets of 5 columns from left to right: generated background, generated foreground, generated masks, foreground carved out by masks, carved foregrounds after spatial transformation, final composed images, nearest neighbor training images to generated images.

Figure 15: Experimental results with minimal object scaling factor 1.1, From left to right, the layout is same to Figure 14.

6.5.2 CATEGORY-SPECIFIC GENERATION

We now show results on category-specific generation (Fig. 16). We train a separate model for each category in CIFAR. When objects occupy a large portion of the image, the background and foreground are less distinguishable. However, for categories such as horse and ship, the backgrounds and foregrounds are disentangled well.

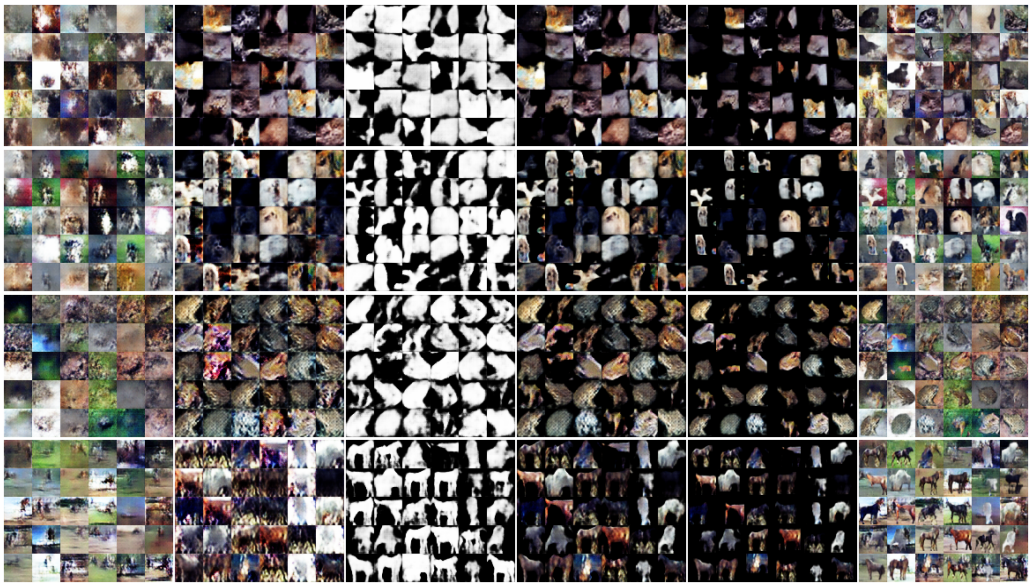

Figure 16: Sets of 6 columns from left to right: generated background, generated foreground, generated masks, foreground carved out by masks, carved foregrounds after spatial transformation, final composed images. From top to bottom, the categories are cat, dog, frog, and horse.

6.5.3 WALKING IN THE LATENT SPACE

Similar to DCGAN, we also show results by walking in the latent space. Note that our model has two or more inputs. So we can walk along any of them or their combination. In Fig. 17, we generate multiple foregrounds for the same fixed generated background. We find that our model consistently generates contextually compatible foregrounds. For example, for the grass-like backgrounds, the foreground generator generates horses and deer, and airplane-like objects for the blue sky.

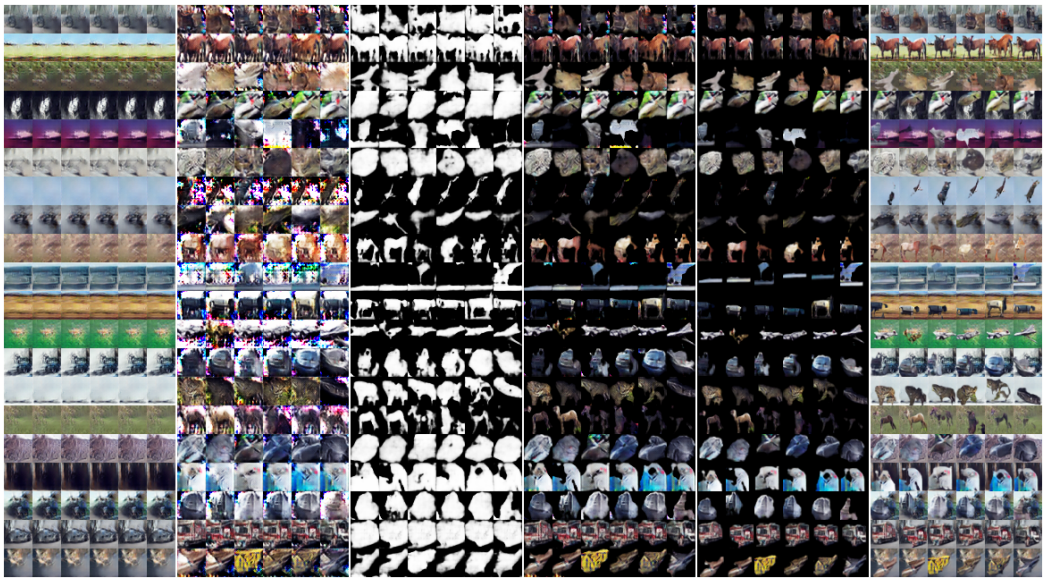

Figure 17: Set of 6 columns from left to right: generated background, generated foreground, generated masks, foreground carved out by masks, carved out foregrounds after spatial transformation, and final composed images. Each row has the same background, but different foregrounds.

6.5.4 WORD CLOUD BASED ON HUMAN STUDY

As we mentioned above, we conducted human studies on CIFAR-10. Besides asking people to select a name from a list for an image, we also conducted another human study where we ask people to use one word (free-form) to describe the main object in the image. Each image was 'named' by 5 unique people. We generate word clouds for real images, images generated by DCGAN and our model (see Fig. 18).

Figure 18: Left to right: word cloud for real images, images generated by DCGAN, images generated by LR-GAN.

6.6 RESULTS ON LFW FACE DATASET

We conduct experiment on face images in LFW dataset (Huang et al., 2007). Different from previous works which work on cropped and aligned faces, we directly generate the original images which contains a large portion of backgrounds. This configuration helps to verify the efficiency of LR-GAN to model the object appearance, shape and pose. In Fig. 19, we show the (intermediate)

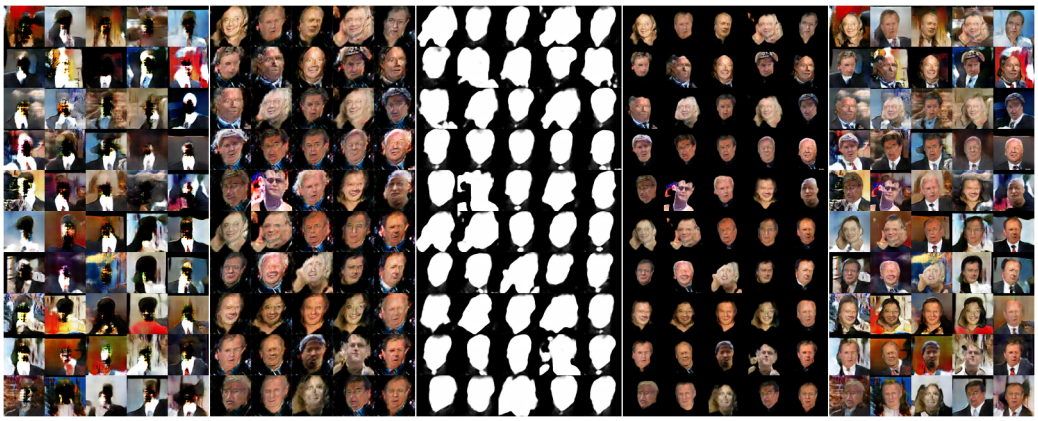

Figure 19: Results on LFW. Set of 5 columns from left to right: generated background, generated foreground, generated masks, carved out foregrounds after spatial transformation, and final composed images.

outputs of LR-GAN. Surprisingly, without any supervisions, the model generated background and faces in separate steps, and the generated masks accurately depict face shapes. Moreover, the model learns where to place the generated faces so that the whole image looks natural. For comparison, please refer to (Kwak & Zhang, 2016) which does not model the transformation.

6.7 Importance of Transformations

Figure 20: Generation results without transformation. From top to bottom, the datasets are CUB-200, CIFAR-10 and LFW. From left to right, they are generated backgrounds, foregrounds, masks and final images. For comparison, in the last column we also show final generated images (of course, not for the same decomposition since we are sampling) using a model with transformations.

We show a comparison between our model with spatial transformation and without spatial transformation in Fig. 20. As we can see, without transformation, the generated backgrouds, foregrounds and masks degenerate. The backgrounds and foregrounds are mixed. As a result, the final generated images also degenerate to some extent. For CUB-200, the final generated images have some blend-

ing between birds and backgrounds. This is particularly the case for those images without bird-shape masks. For CIFAR-10, a number of generated masks are inverted. In this case, the background images are carved out as the foreground objects. The foreground generator takes almost all the duty to generate the final images, which make it harder to generate images as clear as the model with transformation. As for LFW, the background generator generates images contain blurred heads, while the foreground generator generates images with backgrounds. As a result, the final generated images have many blending faces. From these comparions, we qualitivelty demonstrate the importance of performing transformation on the generated foregrounds. Another merit of using transformation is that the intermediate outputs of the model are more interpretable and faciliate to the downstreaming tasks, such as scene paring, which is demonstrated in Sec. 6.10.

6.8 Importance of Shapes

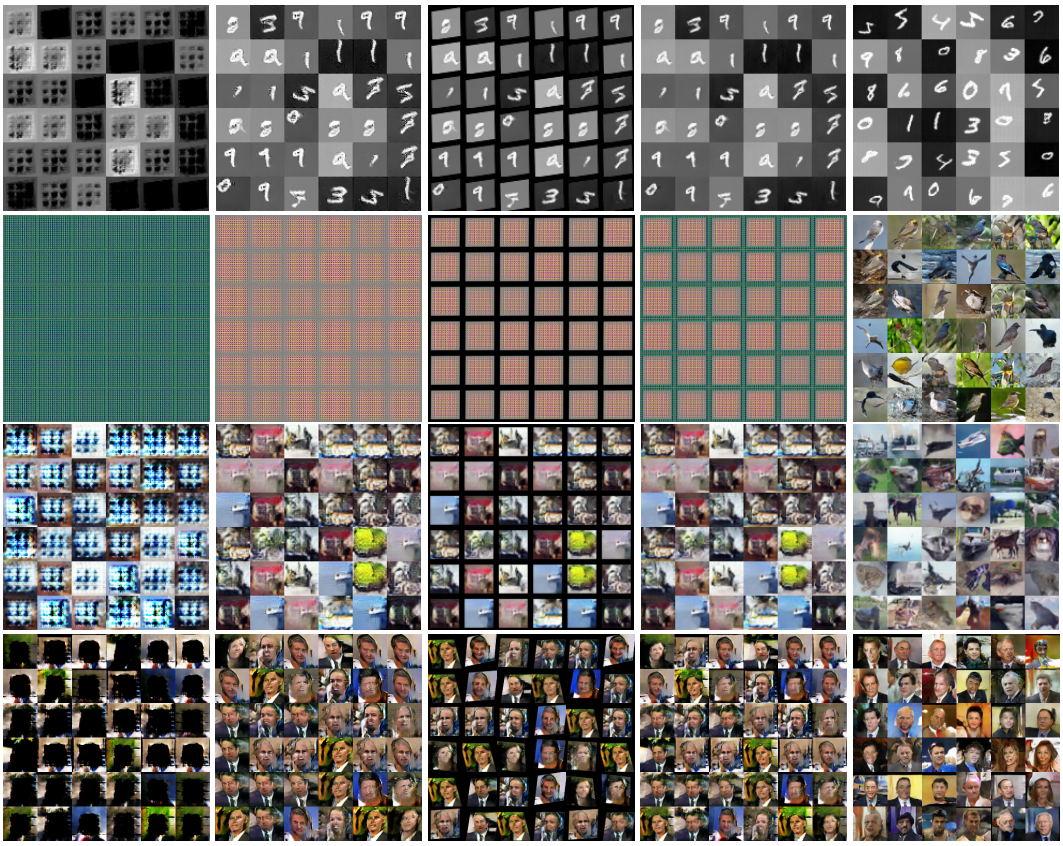

Figure 21: Generation results without mask generator. From top to bottom, the datasets are MNIST-ONE, CUB-200, CIFAR-10 and LFW. From left to right, they are generated backgrounds, foregrounds, transformed foregrounds and final images. For comparison, in the last column we also show final generated images using a model with mask generator.

We perform another ablation study by removing the mask generator to qualify the importance of object shapes. In this case, the generated foreground is put on top of the generated background after being transformed. There is no alpha blending between the foregrounds and backgrounds. The generation results for four datasets, MNIST-ONE, CUB-200, CIFAR-10 and LFW are shown in Fig. 21. As we can, though the model works well for generation of MNIST-ONE, it failed to generate reasonable images across three other datasets. Particularly, the training does not even converge for CUB-200. Based on these results, we qualitatively demonstrate that mask generator in our model is fairly important to obtain plausible results, especially for realistic images.

6.9 STATISTICS ON TRANSFORMATION MATRICES

In this part, we obtain the statistics on the transformation matrices learned in our model for different datasets, including MNIST-ONE, CUB-200, CIFAR-10 and LFW. We used affine transformation in our model. So there are 6 parameters, scaling at x coordinate (s_x), scaling at y coordinate (s_y), translation at x coordinate (t_x), translation at y coordinate (t_y), rotation at x coordinate (r_x) and rotation at y coordinate (r_y),. In Fig. 22, we show the histograms on different parameters for different datasets. According to the histograms, the transformation layer learns scaling, translation and rotation on all datasets. For different datasets, the learned transformation have different patterns. We suspect this is mainly determined by the configurations of objects in the images. For example, on MNIST-ONE, all six parameters have some fluctuations since we put the digits randomly in the backgrounds. For the other three datasets, the scalings converge to single value since the object sizes do not vary much, and the variations on rotation and translation suffice to generate realistic images. Specifically, we can find the generator merely uses translation on x coordinate for generating CUB-200. This makes sense since birds in the images have similar scales, orientations but various horizontal locations. For CIFAR-10, since there are 10 different object categories, the configurations are more diverse, hence the generator uses all parameters for generation except for the scaling. For LFW, since faces have similar configurations, the learned transformations have less fluctuation as well. As a result, we can see that LR-GAN indeed learns the transformations on the foreground to generate images.

6.10 CONDITIONAL IMAGE GENERATION

Considering our model can generate object-like masks (shapes) for images, we conducted an experiment to evaluate whether our model can be potentially used for image segmentation and object detection. We make some changes to the model. For the background generator, the input is a real image instead of a random vector. Then the image is passed through an encoder to extract the hidden features, which replaces the random vector $\boldsymbol{z}_0$ and then fed to the background generator. For the foreground generator, we subtract the image generated by the background generator from the input image to obtain a residual image. Then this residual image is fed to the same encoder to get the hidden features, which are used as the input for foreground generator. In our conditional model, we want to reconstruct the image, so we add another reconstruction loss along with the adversarial loss. We train this conditional model on CIFAR-10. The (intermediate) outputs of the model is shown in Fig. 23. Interestingly, the model successfully learned to decompose the input images to background and foreground.

We also run the conditional LR-GAN on LFW dataset. As we can see in Fig. 24, the object generator automatically and consistently learned to generate the face regions, even though there are large portion of background in the input images. In other words, the conditional LR-GAN successfully learned to detection faces in images. We suspect this success is due to that it has low cost for the generator to generate similar images, and thus converge to the case that the first generator generate background, and the second generator generate face images.

Based on these experiments, we argue that our model can be possibly used for image segmentation and object detection in a generative and unsupervised manner. One future work would be verifying this by applying it to high-resolution and more complicate datasets.

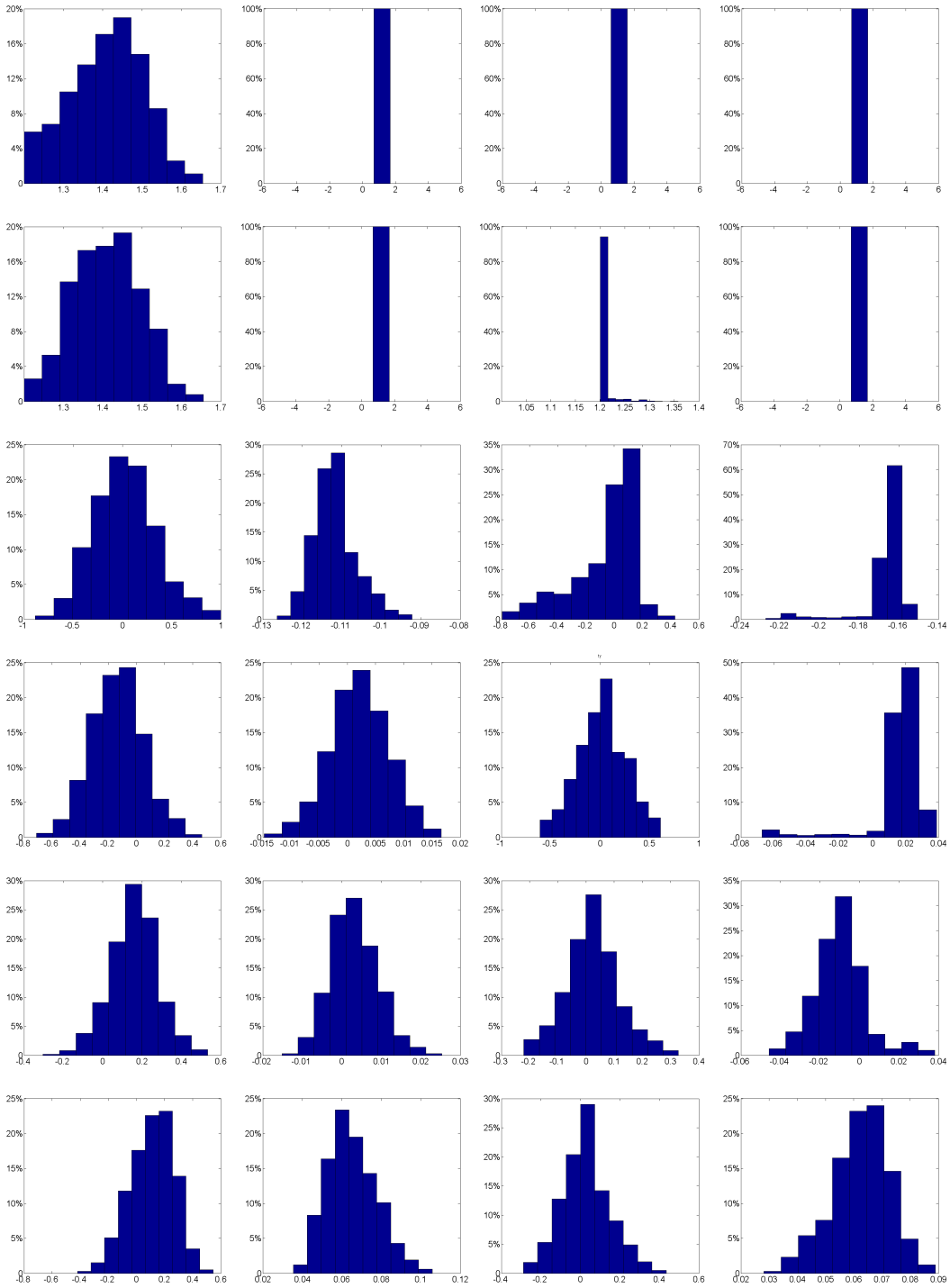

Figure 22: Histograms on transformation parameters. From left to right, the datasets are MNIST-ONE, CUB-200, CIFAR-10 and LFW. From top to bottom, they are s_x, s_y, t_x, t_y, r_x and r_y.

Figure 23: Set of 5 columns from left to right: input image, generated background, generated foreground, generated masks, foreground carved by masks, carved foregrounds after spatial transformation, and final composed (reconstructed) images on CIFAR-10 dataset.

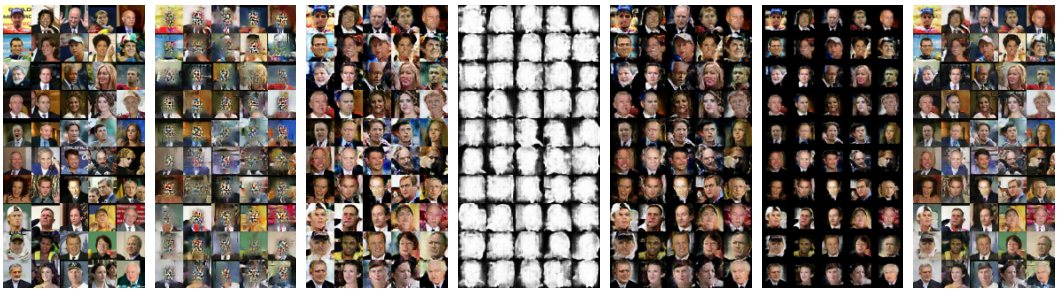

Figure 24: Conditional generation on LFW dataset. Set of 5 columns with the same layout to Fig. 23.