

Appendices for RG-Flow

A DETAILS OF THE NETWORK AND THE TRAINING PROCEDURE

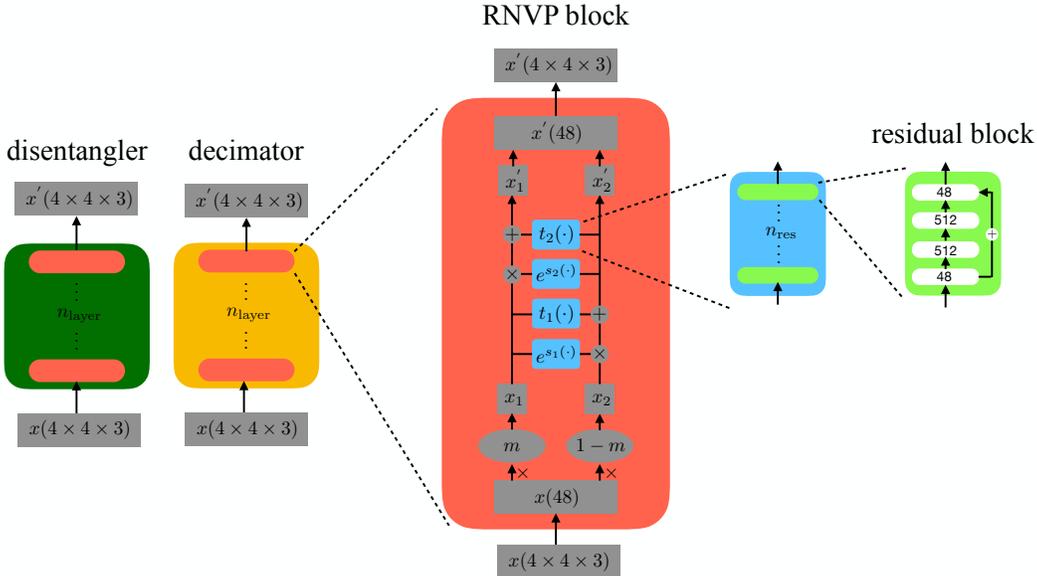


Figure 8: The details of the disentangler/decimator as a bijective map.

As shown in Fig. 2(a), each RG step includes a disentangler (green block) and a decimator (yellow block), which is parameterized by a bijective neural network. The blocks in every horizontal level share parameters, so we view them as a same block. In our reported experiments, disentanglers and decimators in different RG steps do not share parameters. However, we can also make them share parameters, which implements a scale-invariant course-graining process.

For each disentangler or decimator, we split the image into 4×4 patches as shown in Fig. 2(b), stack them along the batch dimension, feed them into the network, and merge the output patches into a new image. After each RG step, the edge length of the image (the number of black lines above yellow blocks) is halved, except for the last (topmost) RG step that decimates all variables.

The choice of the bijective neural networks for the disentanglers and decimators can be versatile, and the performance of RG-Flow is strongly dependent on them. Since tuning the expressive power of those blocks is not the focus of our work, we choose the conventional Real NVP (Dinh et al., 2017) coupling layer to build disentanglers and decimators, which has great expressive power and is easy to invert. Fig. 8 illustrates our implementation. Each RNVP block is shown as a red block. It takes a 4×4 image patch x as input, and split it into x_1 and x_2 using the checkerboard mask m . They are coupled to produce the output x'_2 , using the formula

$$x'_2 = x_2 \odot \exp(s_1(x_1)) + t_1(x_1), \quad (10)$$

where \odot is the element-wise product. $s(\cdot)$ and $t(\cdot)$ are named the scale network and the translation network, which can be arbitrarily complicated to enhance the expressive power, as long as the numbers of input and output variables are the same. Then we use x'_2 to alter x_1 in the similar manner and outputs x'_1 , and combine them to produce the output x' of the RNVP block.

We use residual networks with n_{res} residual blocks as $s(\cdot)$ and $t(\cdot)$, shown as blue blocks in Fig. 8, and we choose $n_{\text{res}} = 4$ in our implementation. Each residual block has 3 linear layers with size $24 \rightarrow 512, 512 \rightarrow 512, 512 \rightarrow 24$. Between linear layers, we insert swish activations (Chen et al., 2019), which is reported to give better results than ReLU and softplus, and its smoothness benefits our further analysis with higher order derivatives. We use Kaiming initialization (He et al., 2016) and weight normalization (Salimans & Kingma, 2016) on the linear layers.

The CelebA dataset contains rich information on different scales, including high-level information like gender and emotion, mid-level one like shapes of eyes and nose, and low-level one like details in hair and wrinkles. Because lower-level RG steps take larger images as input, we heuristically put more parameters in them. The numbers of RNVP blocks in all RG steps are $n_{\text{layer}} = 8, 6, 4, 2$ respectively.

To preprocess the dataset, we use the aligned images from CelebA, crop a 148×148 patch at the center of each image, downscale the patch to 32×32 using bicubic downsampling, and randomly flip it horizontally.

We use AdamW optimizer (Loshchilov & Hutter, 2019) with conventional learning rate 10^{-3} and weight decay rate 5×10^{-5} . To further stabilize training, we use gradient clipping with global norm 1. Between coupling layers, we use checkpointing (Paszke et al., 2019) to reduce memory usage. After using the checkpointing technique, we find the network’s memory consumption is greatly reduced. The maximal batch size can be set to 1024 on a single Nvidia Titan RTX given the current setup, which approximately has a million parameters. In our experiment, the batch size is conventionally set to 64. A training step takes about 1.2 seconds on an Nvidia Titan RTX.

The code for our implementation is at <https://github.com/rgflowopen/rg-flow>

B DETAILS OF THE SYNTHETIC MULTI-SCALE DATASETS

To illustrate RG-Flow’s ability to disentangle representations at different scales and spatially separated representations, we propose two synthetic toy datasets with multi-scale features, named MSDS1 and MSDS2, as shown in Fig. 9. Each dataset contains 10^5 images of 32×32 pixels. In each image, there are 16 ovals with different colors and orientations, and their positions have small random variations to deform the 4×4 grid. In MSDS1, all ovals in an image have almost the same color, while their orientations are randomly distributed. So the color is a global feature in MSDS1, and the orientation is a local feature. In MSDS2, on the contrary, the orientation is a global feature, and the color is a local one.

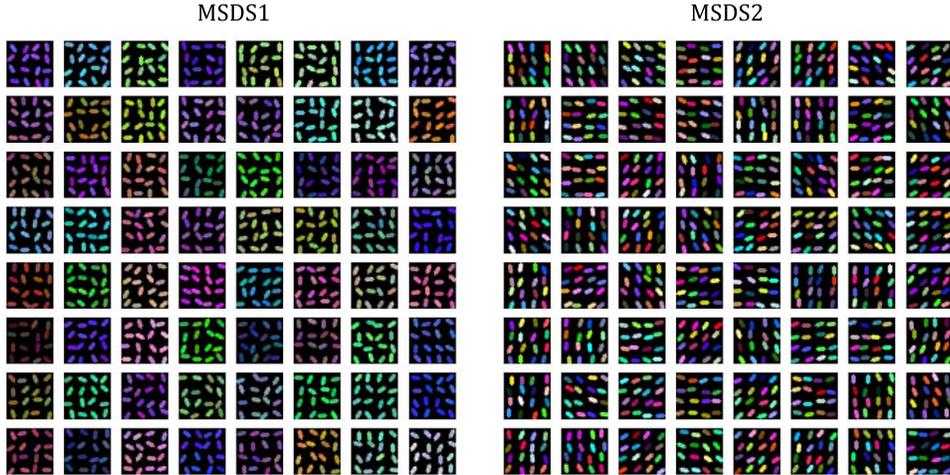


Figure 9: Samples from MSDS1 and MSDS2.

We trained RG-Flow on those datasets, with $n_{\text{layer}} = 4$ and $n_{\text{res}} = 4$, and other hyperparameters described in Appendix A. For comparison, we also trained Real NVP on those datasets, with approximately the same number of trainable parameters. Their generated images are shown in Fig. 10, where we can intuitively find that RG-Flow has learned the characteristics of the two datasets. Namely, in each image in MSDS1 the ovals should have almost the same color, and in MSDS2 the same orientation. In contrast, Real NVP fails to capture the global and local features of those datasets. The metrics of bits per dimension (BPD) and Fréchet Inception distance (FID) are listed in Table 1. Note that FID may not reflect much semantic property for such synthetic datasets.

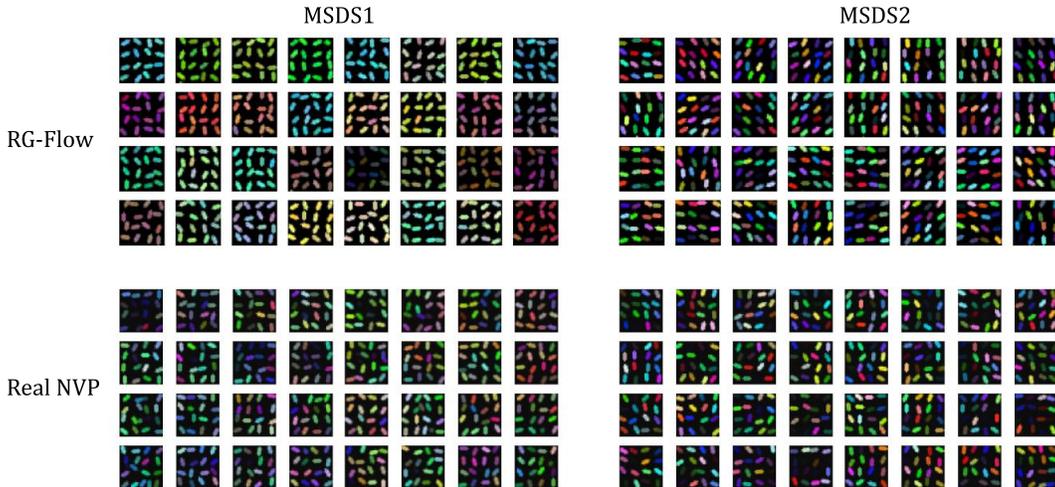


Figure 10: Samples from RG-Flow and Real NVP trained on MSDS1 and MSDS2.

	BPD ↓		FID ↓	
	MSDS1	MSDS2	MSDS1	MSDS2
RG-Flow	0.906	1.01	1.40	2.61
Real NVP	1.07	1.14	50.4	78.0

Table 1: BPD and FID from RG-Flow and Real NVP trained on MSDS1 and MSDS 2.

C DISCUSSION ON REAL NVP

As a comparison to Fig. 4, we plot receptive fields of Real NVP in Fig. 11(a), together with the statistics of their strength in Fig. 11(b). Without local constraints on the bijective maps, there is no generation causal cone for Real NVP and other globally connected multi-scale models, and we cannot find semantic factors separated at different scales in general. In contrast, our RG-Flow can separate semantic factors at different scales, as shown in Fig. 4.

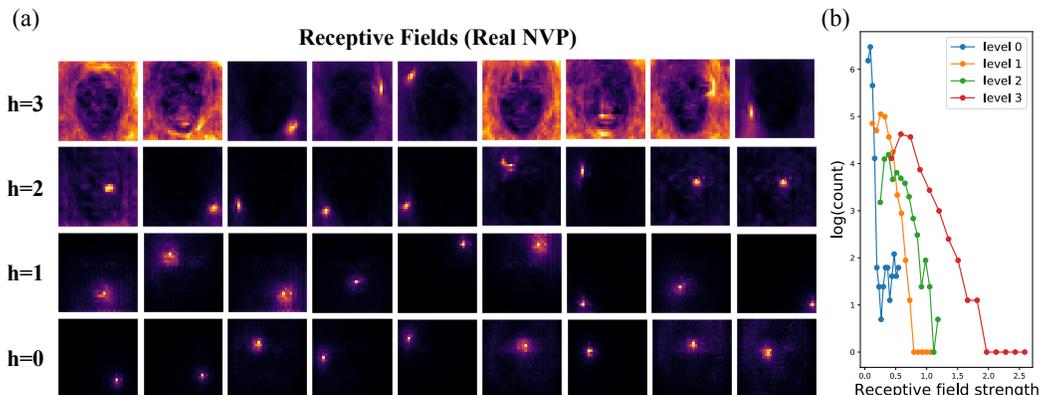


Figure 11: Subplot (a) shows the randomly picked receptive fields from the trained Real NVP on CelebA at different levels. Subplot (b) shows the statistics of the receptive fields' strength.

D IMAGE GENERATION USING EFFECTIVE AND MIXED TEMPERATURES

Just like other generative models, we can define the effective temperature for RG-flow. Our prior distribution is

$$p_Z(\mathbf{z}) = \prod_l p(z_l), \quad (11)$$

where $l = (i, j, h)$ labels every latent variable by its coordinate on the hyperbolic tree, especially h is the RG level. A model with effective temperature T ($T > 0$) changes the prior distribution to $p_{Z,T}(\mathbf{z})$, with

$$p_{Z,T}(\mathbf{z}) \propto (p_Z(\mathbf{z}))^{1/T}. \quad (12)$$

For Laplacian prior, the effective temperature is implemented as

$$p_{Z,T}(\mathbf{z}) = \prod_l \frac{1}{2T} \exp\left(-\frac{|z_l|}{T}\right). \quad (13)$$

Moreover, we can define a mixed temperature model on the hyperbolic tree by

$$p_{Z,\text{mix}}(\mathbf{z}) \propto \prod_l (p(z_l))^{1/T_l}, \quad (14)$$

where T_l can be coordinate-dependent.

In our training procedure, we find the bijective maps on the lowest level take a longer time to converge. Therefore, when plotting the results in Fig. 5, we use the mixed temperature scheme with $T_{h=0} = 0.2, T_{h=1} = T_{h=2} = T_{h=3} = 0.6$, where T_h is the effective temperature on level h . Then we vary each latent variable from 0 to 1.5 if it is in high level or mid level, and from 0 to 6 if it is in low level.

E A TOY MODEL FOR SPARSE PRIOR DISTRIBUTION

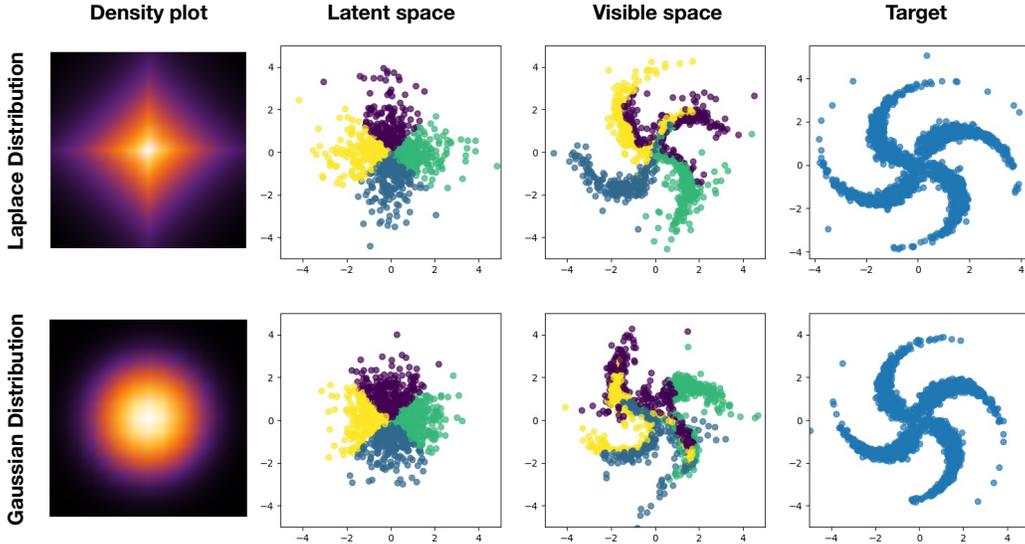


Figure 12: Two-dimensional pinwheel model.

In the first column of Fig. 12, the density plots of Laplacian distribution and Gaussian distribution are shown. As we can see, the Gaussian distribution has rotational symmetry, whereas the Laplacian distribution breaks $SO(2)$ rotational symmetry to C_4 symmetry. A flow-based generative model is trained to match the target distribution, which is a four-leg pinwheel as shown in the last column in Fig. 12. Given either Gaussian distribution or Laplacian distribution as the prior distribution, the model can learn the target distribution. In the second column, we sample 100 points and color them

by their living quadrant in the prior distribution. Then we map them to the visible space by the trained model, as shown in the third column. We see that four quadrants are approximately mapped to the four legs of the pinwheel if Laplacian prior is used. But for the Gaussian case, since it has rotational symmetry, the points in different quadrants are mixed more in the visible space, which makes it harder to interpret the mapping.

F DETAILS OF INPAINTING EXPERIMENTS

For the inpainting experiments shown in Fig. 7, we randomly choose a corrupted region of 10×10 pixels on the ground truth image, marked as the red patch in the second row of Fig. 7. We generate an image x_g from latent variables z_g , and use its corresponding region to fill in the corrupted region. Then we map the filled image x_f back to the latent variables z_f , and compute the log-likelihood. To recover the ground truth image, we optimize z_g to maximize the log-likelihood.

For RG-Flow, we only variate the latent variables living inside the inference causal cone, which is about 1200 out of 3072 latent variables. For the constrained Real NVP, we randomly pick the same amount of latent variables and allow them to variate, and we find it fails to inpaint the image in general. As a check, we find Real NVP can successfully inpaint the images if we optimize all latent variables, as shown in the last row of Fig. 7.

We use the conventional Adam optimizer to do the optimization. During the optimization procedure, we find the optimizer can be trapped in local minima. Therefore, for all experiments, we first randomly draw 200 initial samples of latent variables that are allowed to variate, then pick the one with the largest log-likelihood as the initialization.

To quantitatively assess the quality of inpainted images, we compute the peak signal-to-noise ratio (PSNR) of them against the ground truth images, and take the average over the 15 samples shown in Fig. 7. To further incorporate the semantic properties in the assessment, we can also use the Inception-v3 network (Szegedy et al., 2016). We first scale the images to 299×299 , then feed them into the Inception-v3 network, extract the features from the “pool3” layer (as in FID score), and compute the PSNR. The results are listed in Table 2.

	PSNR \uparrow	Inception-PSNR \uparrow
RG-Flow	38.7	24.9
Constrained Real NVP	23.2	15.7
Real NVP	37.0	24.8

Table 2: PSNR and Inception-PSNR of inpainted images.

G EXPERIMENTS ON OTHER DATASETS

For CIFAR-10 dataset, we use the same hyperparameters as described in Appendix. A. For 3D Chair dataset, we use $n_{\text{layer}} = 8$ for all RG steps, because there is not so much low-level information as in CelebA and CIFAR-10. We also trained Real NVP on those datasets for comparison, with approximately the same number of trainable parameters. The metrics of bits per dimension (BPD) and Fréchet Inception distance (FID) are listed in Table. 3.

	BPD \downarrow			FID \downarrow		
	CelebA	CIFAR-10	3D Chair	CelebA	CIFAR-10	3D Chair
RG-Flow	3.47	3.35	0.930	31.3	98.5	36.2
Real NVP	3.36	3.61	0.933	11.1	126	73.8

Table 3: Bits per dimension (BPD) from RG-Flow and Real NVP trained on various datasets.

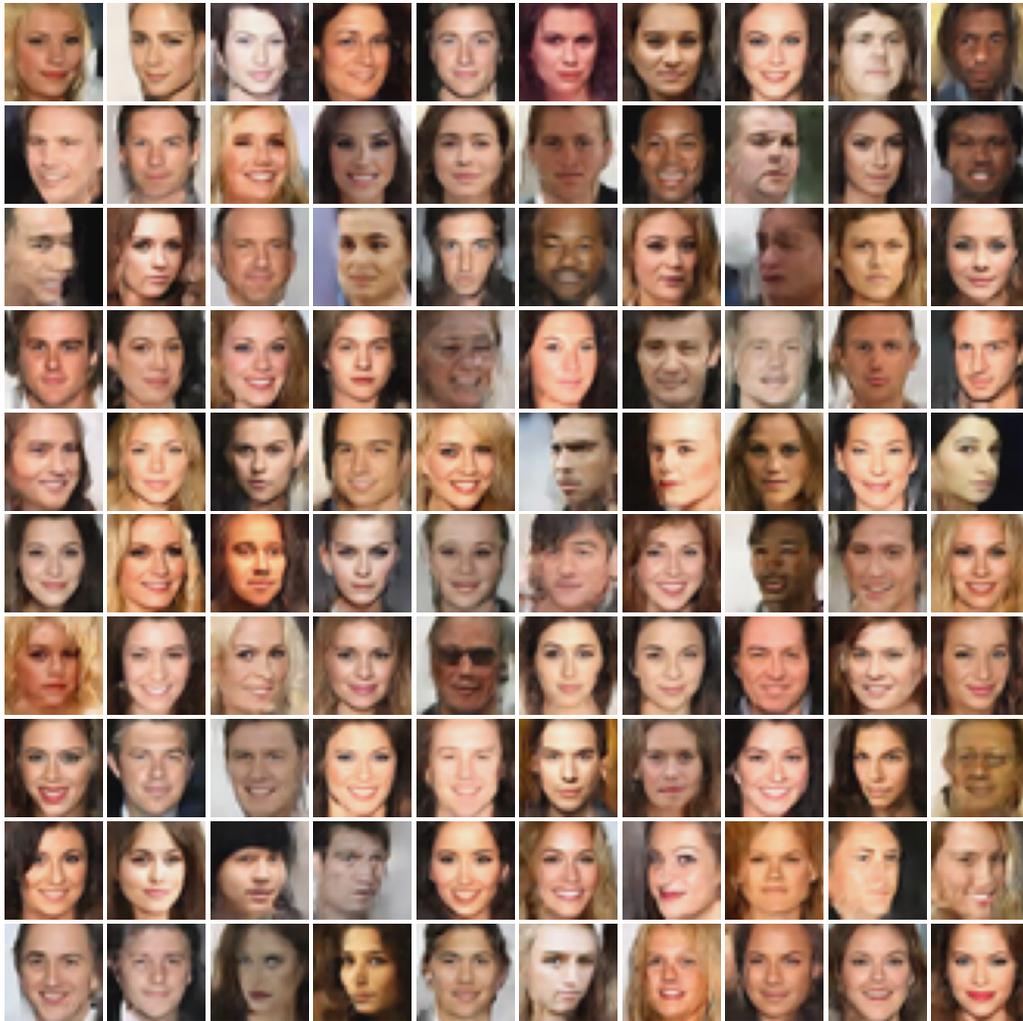


Figure 13: Samples from RG-Flow trained on CelebA dataset. We use $T = 0.9$ when sampling.

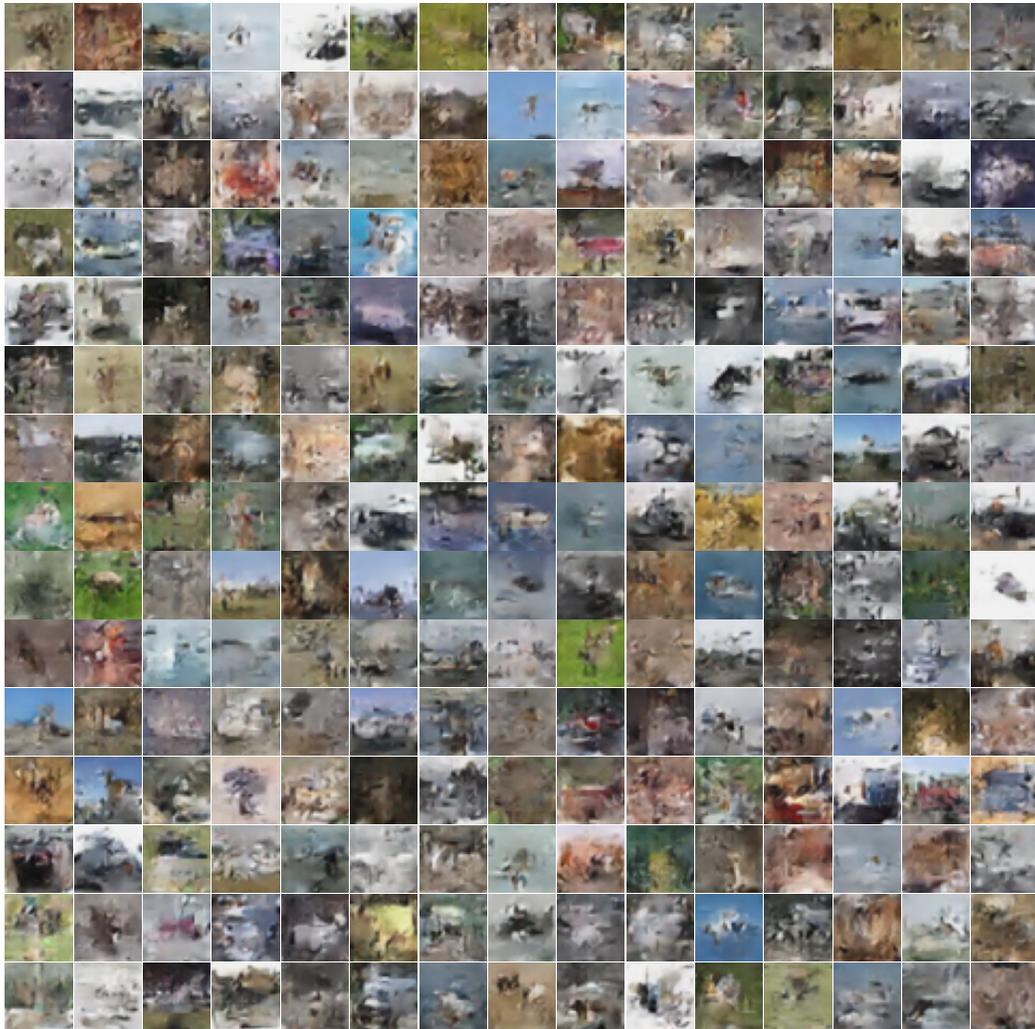


Figure 14: Samples from RG-Flow trained on CIFAR-10 dataset.



Figure 15: Samples from RG-Flow trained on 3D Chair dataset.

H RECEPTIVE FIELDS OF THE LATENT REPRESENTATIONS

More examples of randomly picked receptive fields are plotted in Fig. 16, Fig. 17, Fig. 18, and Fig. 19. For better visualization, we normalize all receptive fields' strength to one.



Figure 16: Receptive fields of high-level latent variables ($h = 3$).

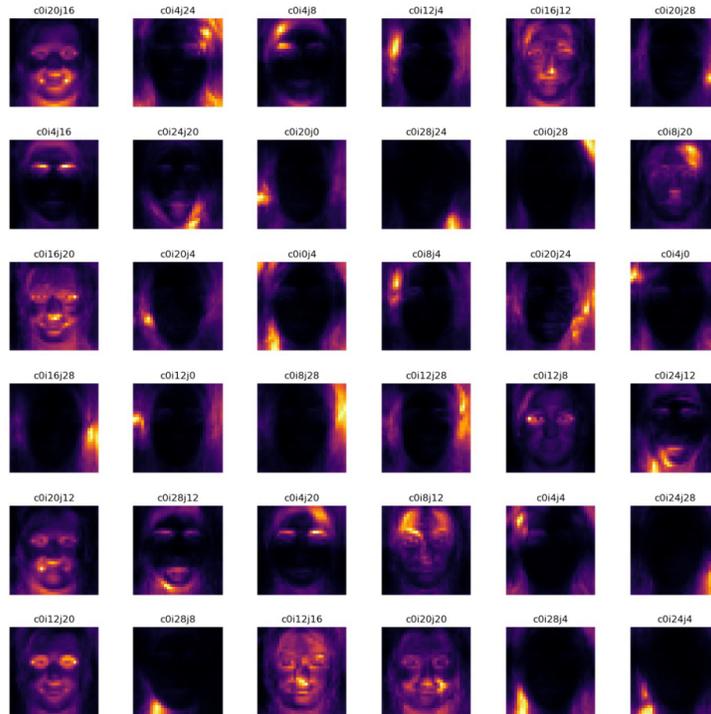


Figure 17: Receptive fields of mid-level latent variables ($h = 2$).

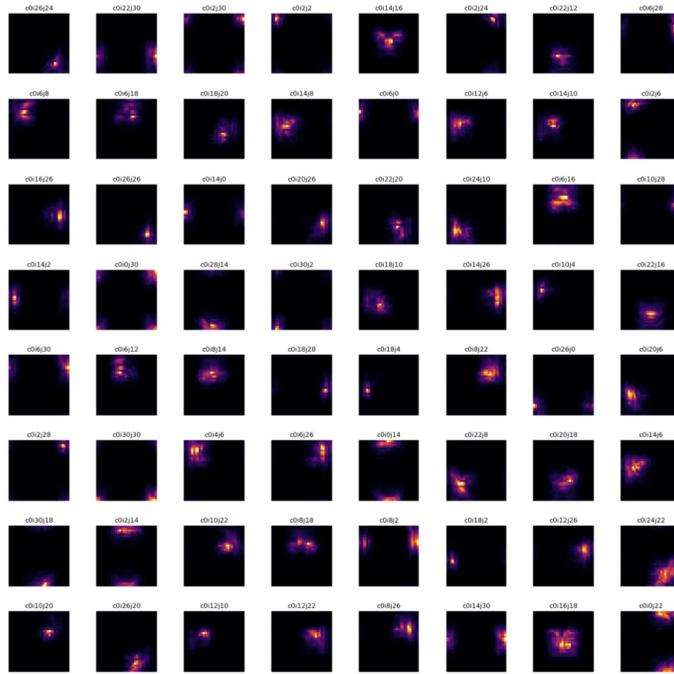


Figure 18: Receptive fields of low-level latent variables ($h = 1$).

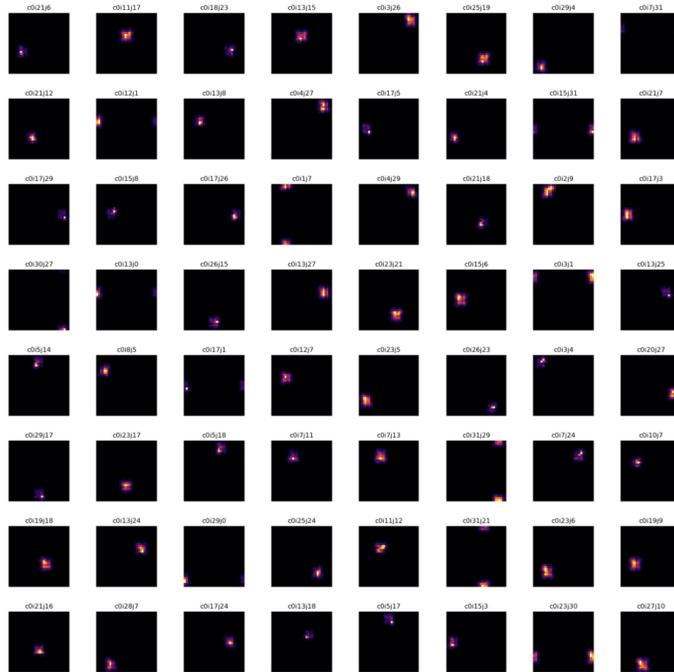


Figure 19: Receptive fields of the lowest-level latent variables ($h = 0$).