Supplementary Material: Improved ArtGAN for Conditional Synthesis of Natural Image and Artwork

August 23, 2018

Abstract

This supplementary file provides additional details which are not covered in the submission due to the page limit. First, list of classes in Wikiart dataset is provided. Next, we present the pseudocode that is used to train the ArtGAN. Then, we list the detailed model configurations of the Generator and Discriminator used in our work to facilitate future reimplementation of our work. Finally, we show more qualitative results for experiments on Wikiart, CIFAR-10, STL-10, CUB-200, and Oxford-102 datasets. Codes are available at https://github.com/cs-chan/ArtGAN.

1 Wikiart Dataset

Table 1 details the members in each of the annotation class in the Wikiart dataset.

	List of Members		
Style	(1) Abstract Expressionism	(2) Action Painting	(3) Analytical Cubism
	(4) Art Nouveau-Modern Art	(5) Baroque	(6) Colour Field Painting
	(7) Contemporary Realism	(8) Cubism	(9) Early Renaissance
	(10) Expressionism	(11) Fauvism	(12) High Renaissance
	(13) Impressionism	(14) Mannerism-Late-Renaissance	(15) Minimalism
	(16) Primitivism-Naive Art	(17)New Realism	(18) Northern Renaissance
	(19) Pointillism	(20) Pop Art	(21) Post Impressionism
	(22) Realism	(23) Rococo	(24) Romanticism
	(25) Symbolism	(26) Synthetic Cubism	(27) Ukiyo-e
Genre	(1) Abstract Painting	(2) Cityscape	(3) Genre Painting
	(4) Illustration	(5) Landscape	(6) Nude Painting
	(7) Portrait	(8) Religious Painting	(9) Sketch and Study
	(10) Still Life		
Artist	(1) Albrecht Durer	(2) Boris Kustodiev	(3) Camille Pissarro
	(4) Childe Hassam	(5) Claude Monet	(6) Edgar Degas
	(7) Eugene Boudin	(8) Gustave Dore	(9) Ilya Repin
	(10) Ivan Aivazovsky	(11) Ivan Shishkin	(12) John Singer Sargent
	(13) Marc Chagall	(14) Martiros Saryan	(15) Nicholas Roerich
	(16) Pablo Picasso	(17) Paul Cezanne	(18) Pierre-Auguste Renoir
	(19) Pyotr Konchalovsky	(20) Raphael Kirchner	(21) Rembrandt
	(22) Salvador Dali	(23) Vincent van Gogh	

Table 1: List of *Style*, *Genre*, and *Artist* in the Wikiart Dataset

2 Algorithm

Algorithm 1 illustrates the training process in our ArtGAN models. The notations are consistent with the submission. In addition, we denote $\mathbf{K} = \{1, \ldots, K\}$ as the set of indices of the classes. Then, the one-hot vector of a sample $\bar{c}_k$ is randomly sampled, where $k \in \mathbf{K}$ and value at position k is set to **one** while the rest of the elements are set to **zero**. Given n samples in a minibatch, $\mathbf{y} = \{y_1, \ldots, y_n\}$ is a vector of the computed adversarial outputs. While, $\mathbf{C} = \{\mathbf{c}_1, \ldots, \mathbf{c}_n\}$ is a set of class prediction.

Algorithm 1 Pseudocode for training ArtGAN

Require: Minibatch size, n, learning rate, λ, and $\mathbf{z}$ vector size, d
Require: Randomly initialize θ_D and θ_G
while condition not met **do**
  Sample $\mathbf{Z} = [\mathbf{z}_1, \ldots, \mathbf{z}_n] \sim \mathcal{N}(0, 1)^{n \times d}$
  Randomly set $\bar{\mathbf{C}} = [\bar{c}_{k_1}, \ldots, \bar{c}_{k_n}]$
  Sample minibatch $\hat{\mathbf{X}} = [\hat{\mathbf{x}}_1, \ldots, \hat{\mathbf{x}}_n]$
  $\mathbf{C}, \mathbf{y} = D(\hat{\mathbf{X}})$
  $\bar{\mathbf{X}} = G(\mathbf{Z}, \bar{\mathbf{C}})$
  $\mathbf{C}', \mathbf{y}' = D(\bar{\mathbf{X}})$
  if use magnified learning **then**
    $\mathbf{R} = Dec(\Phi(\hat{\mathbf{X}}))$
    $\mathbf{R}' = Dec(\Phi(\bar{\mathbf{X}}))$
    $\theta_D = \theta_D - \lambda \frac{\partial \mathcal{L}_{Dae}}{\partial \theta_D}$, $\mathcal{L}_{Dae} \leftarrow \mathbf{y}, \mathbf{C}, \bar{\mathbf{C}}, \mathbf{C}', \mathbf{y}', \mathbf{R}$
    $\theta_G = \theta_G - \lambda \frac{\partial \mathcal{L}_{Gae}}{\partial \theta_G}$, $\mathcal{L}_{Gae} \leftarrow \bar{\mathbf{C}}, \mathbf{C}', \mathbf{y}', \mathbf{R}'$
  else
    $\theta_D = \theta_D - \lambda \frac{\partial \mathcal{L}_D}{\partial \theta_D}$, $\mathcal{L}_D \leftarrow \mathbf{y}, \mathbf{C}, \bar{\mathbf{C}}, \mathbf{C}', \mathbf{y}'$
    $\theta_G = \theta_G - \lambda \frac{\partial \mathcal{L}_G}{\partial \theta_G}$, $\mathcal{L}_G \leftarrow \bar{\mathbf{C}}, \mathbf{C}', \mathbf{y}'$
  end if
end while

3 Network Architectures

This section describes the network architectures used in our experiments. All models used the same denoiser in ArtGAN-DFM, which has the same architecture as Warde et al. [1] by using a Gaussian noise on the inputs, followed by 10 fully connected layers (with $nfm = 1024$ for all intermediate layers). When *categorical autoencoder-based discriminator* is used, the input of the decoder (or denoiser) is the output of the last convolutional layer of the classifier, that is the second last layer of the classifier. The annotations are as follow:

1. conv(nfm, k, s): convolution operation with nfm feature maps, k kernel size, and stride of s, followed by a leaky ReLU with the parameter in negative slope set to 0.2.

2. convBN(nfm, k, s): same as conv(nfm, k, s) except with Batch Normalization between the convolution operation and leaky ReLU.

3. NNupsample(S): Nearest neighbour upsampling with upscale size of S.

4. fc(nfm): fully connected layer with nfm feature maps, followed by a leaky ReLU with the parameter in negative slope set to 0.2.

5. fcBN(nfm): same as fc(nfm) but with Batch Normalization between the fully connected layer and leaky ReLU.

6. Dropout(γ): dropout with $100 \times \gamma\%$ of the neurons dropped.

3.1 CIFAR-10

This section describes the network architectures used on CIFAR-10. Table 2 shows the architecture for the discriminator. Table 3 shows the architectures for the generators.

Table 2: Network architectures of the discriminator used on CIFAR-10, which contains a classifier and a decoder.

Classifier	Decoder
conv(96, 3, 1)	convBN(256, 3, 1)
convBN(96, 3, 2)	
Dropout(0.2)	
convBN(192, 3, 1)	NNupsample(16)
convBN(192, 3, 2)	convBN(128, 3, 1)
Dropout(0.2)	convBN(128, 3, 1)
convBN(256, 3, 1)	NNupsample(32)
convBN(256, 1, 1)	convBN(64, 3, 1)
convBN(512, 1, 1)	conv(3, 3, 1)
fc(10)	

Table 3: Network architectures for the generators (with and without magnified learning) used on CIFAR-10.

Generator	Generator with Magnified Learning
fcBN($512 \times 4 \times 4$)	fcBN($512 \times 4 \times 4$)
NNupsample(8)	NNupsample(8)
convBN(256, 3, 1)	convBN(256, 3, 1)
NNupsample(16)	NNupsample(16)
convBN(128, 3, 1)	convBN(128, 3, 1)
convBN(128, 3, 1)	convBN(128, 3, 1)
NNupsample(32)	NNupsample(32)
convBN(64, 3, 1)	convBN(64, 3, 1)
convBN(3, 3, 1)	convBN(64, 3, 1)
	NNupsample(64)
	convBN(32, 3, 1)
	conv(3, 3, 1)

3.2 STL-10

Table 4 and Table 5 show the network architectures of the discriminator and generator used on STL-10.

Table 4: Network architectures for discriminator (containing a classifier and a decoder) used on STL-10.

Classifier	Decoder
conv(64, 3, 1) convBN(64, 3, 2) Dropout(0.2)	convBN(512, 3, 1)
convBN(128, 3, 1) convBN(128, 3, 2) Dropout(0.2)	NNupsample(16) convBN(256, 3, 1)
convBN(256, 3, 1) convBN(256, 3, 2) Dropout(0.2)	NNupsample(32) convBN(128, 3, 1) convBN(128, 3, 1)
convBN(512, 3, 1) convBN(512, 3, 1)	NNupsample(64) convBN(64, 3, 1) conv(3, 3, 1)
fc(10)	

Table 5: Network architectures for the generators (with and without magnified learning) used on STL-10.

Generator	Generator with Magnified Learning
fcBN($512 \times 4 \times 4$) NNupsample(8) convBN(512, 3, 1)	fcBN($512 \times 4 \times 4$) NNupsample(8) convBN(512, 3, 1)
NNupsample(16) convBN(256, 3, 1)	NNupsample(16) convBN(256, 3, 1)
NNupsample(32) convBN(128, 3, 1) convBN(128, 3, 1)	NNupsample(32) convBN(128, 3, 1) convBN(128, 3, 1)
NNupsample(64) convBN(64, 3, 1) convBN(3, 3, 1)	NNupsample(64 convBN(64, 3, 1) convBN(64, 3, 1)
	NNupsample(128) convBN(32, 3, 1) conv(3, 3, 1)

3.3 Wikiart

All tasks in Wikiart (i.e. *genres*, *styles*, and *artists*) used the same architectures described in Table 6 and Table 7.

Table 6: Network architectures for discriminator (containing a classifier and a decoder) used on Wikiart.

Classifier	Decoder
conv(128, 3, 2) Dropout(0.2)	convBN(512, 3, 1)
convBN(256, 3, 2) Dropout(0.2)	NNupsample(8) convBN(256, 3, 1)
convBN(512, 3, 2) convBN(512, 3, 1) Dropout(0.2)	NNupsample(16) convBN(128, 3, 1)
convBN(1024, 3, 2)	NNupsample(32) convBN(64, 3, 1)
fc(10)	NNupsample(64) convBN(32, 3, 1) conv(3, 3, 1)

Table 7: Network architectures for the generators (with and without magnified learning) used on Wikiart.

Generator	Generator with Magnified Learning
fcBN($512 \times 4 \times 4$) NNupsample(8) convBN(512, 3, 1)	fcBN($512 \times 4 \times 4$) NNupsample(8) convBN(512, 3, 1)
NNupsample(16) convBN(256, 3, 1)	NNupsample(16) convBN(256, 3, 1)
NNupsample(32) convBN(128, 3, 1)	NNupsample(32) convBN(128, 3, 1)
NNupsample(64) convBN(64, 3, 1) convBN(3, 3, 1)	NNupsample(64 convBN(64, 3, 1) convBN(64, 3, 1)
	NNupsample(128) convBN(32, 3, 1) conv(3, 3, 1)

3.4 Oxford-102 flowers and CUB-200 birds

Oxford-102 and CUB-200 datasets share the same network architectures as described in Table 8 and Table 9.

Table 8: Network architectures for discriminator (containing a classifier and a decoder) used on Oxford-102 and CUB-200.

Classifier	Decoder
conv(64, 3, 2) Dropout(0.2)	convBN(512, 3, 1)
convBN(128, 3, 2) Dropout(0.2)	NNupsample(8) convBN(256, 3, 1)
convBN(256, 3, 2) convBN(256, 3, 1) Dropout(0.2)	NNupsample(16) convBN(128, 3, 1)
convBN(512, 3, 2) convBN(512, 3, 1)	NNupsample(32) convBN(64, 3, 1)
fc(K)	NNupsample(64) conv(32, 3, 1) conv(3, 3, 1)

Table 9: Network architectures for the generators (with and without magnified learning) used on Oxford-102 and CUB-200.

Generator	Generator with Magnified Learning
fcBN($512 \times 4 \times 4$) NNupsample(8) convBN(512, 3, 1)	fcBN($512 \times 4 \times 4$) NNupsample(8) convBN(512, 3, 1)
NNupsample(16) convBN(256, 3, 1)	NNupsample(16) convBN(256, 3, 1)
NNupsample(32) convBN(128, 3, 1)	NNupsample(32) convBN(128, 3, 1)
NNupsample(64) convBN(64, 3, 1) convBN(3, 3, 1)	NNupsample(64 convBN(64, 3, 1) convBN(64, 3, 1)
	NNupsample(128) convBN(32, 3, 1) conv(3, 3, 1)

4 More generated samples

4.1 Wikiart

More generated fine-art paintings are visualized in Figure 1, Figure 2, and Figure 3 at high resolution (128×128 pixels).

Figure 1: Generated *genres* images at $\mathbf{128 \times 128}$ pixels. From top to bottom: (1) Abstract painting, (2) Cityscape, (3) Genre painting, (4) Illustration, (5) Landscape, (6) Nude painting, (7) Portrait, (8) Religious painting, (9) Sketch and study, (10) Still life.

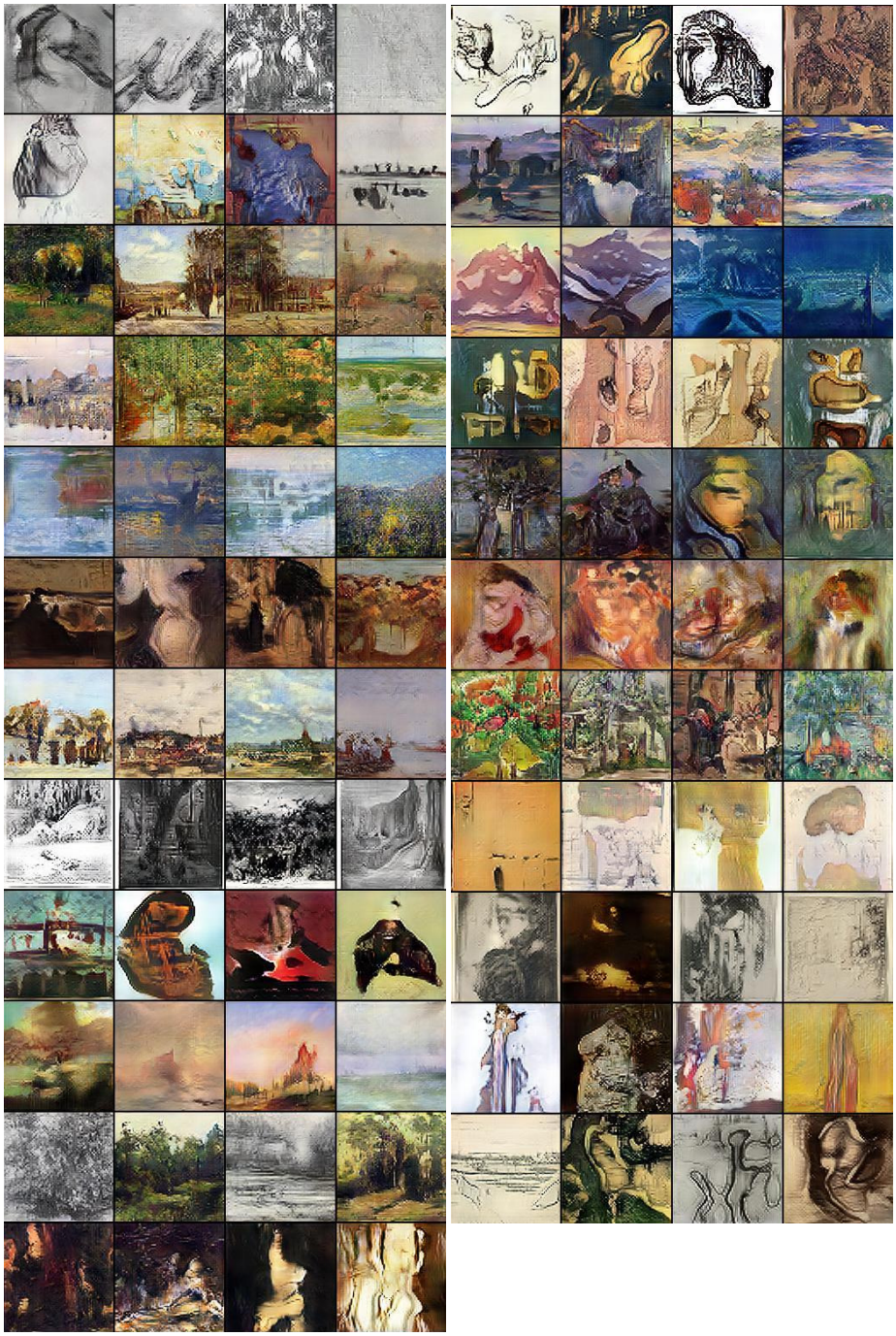

Figure 2: Generated *artists* images at $\mathbf{128} \times \mathbf{128}$ pixels. (**Left**) From top to bottom: (1) Albrecht Durer, (2) Boris Kustodiev, (3) Camille Pissarro, (4) Childe Hassam , (5) Claude Monet, (6) Edgar Degas, (7) Eugene Boudin, (8) Gustave Dore, (9) Ilya Repin, (10) Ivan Aivazovsky, (11) Ivan Shishkin, (12) John Singer Sargent. (**Right**) From top to bottom: (13) Marc Chagall, (14) Martiros Saryan, (15) Nicholas Roerich, (16) Pablo Picasso, (17) Paul Cezanne, (18) Pierre Auguste Renoir, (19) Pyotr Konchalovsky, (20) Raphael Kirchner, (21) Rembrandt, (22) Salvador Dali, (23) Vincent van Gogh.

Figure 3: Generated *styles* images at $\mathbf{128} \times \mathbf{128}$ pixels. (**Left**) From top to bottom: (1) Abstract Expressionism, (2) Action painting, (3) Analytical Cubism, (4) Art Nouveau, (5) Baroque, (6) Color Field Painting, (7) Contemporary Realism, (8) Cubism, (9) Early Renaissance. (**Middle**) From top to bottom: (10) Expressionism, (11) Fauvism, (12) High Renaissance, (13) Impressionism, (14) Mannerism Late Renaissance, (15) Minimalism, (16) Naive Art Primitivism, (17) New Realism, (18) Northern Renaissance. (**Right**) From top to bottom: (19) Pointillism, (20) Pop Art, (21) Post Impressionism, (22) Realism, (23) Rococo, (24) Romanticism, (25) Symbolism, (26) Synthetic Cubism, (27) Ukiyo-e.

4.2 CIFAR-10

Figure 4 shows generated images at 64×64 resolution trained on CIFAR-10.

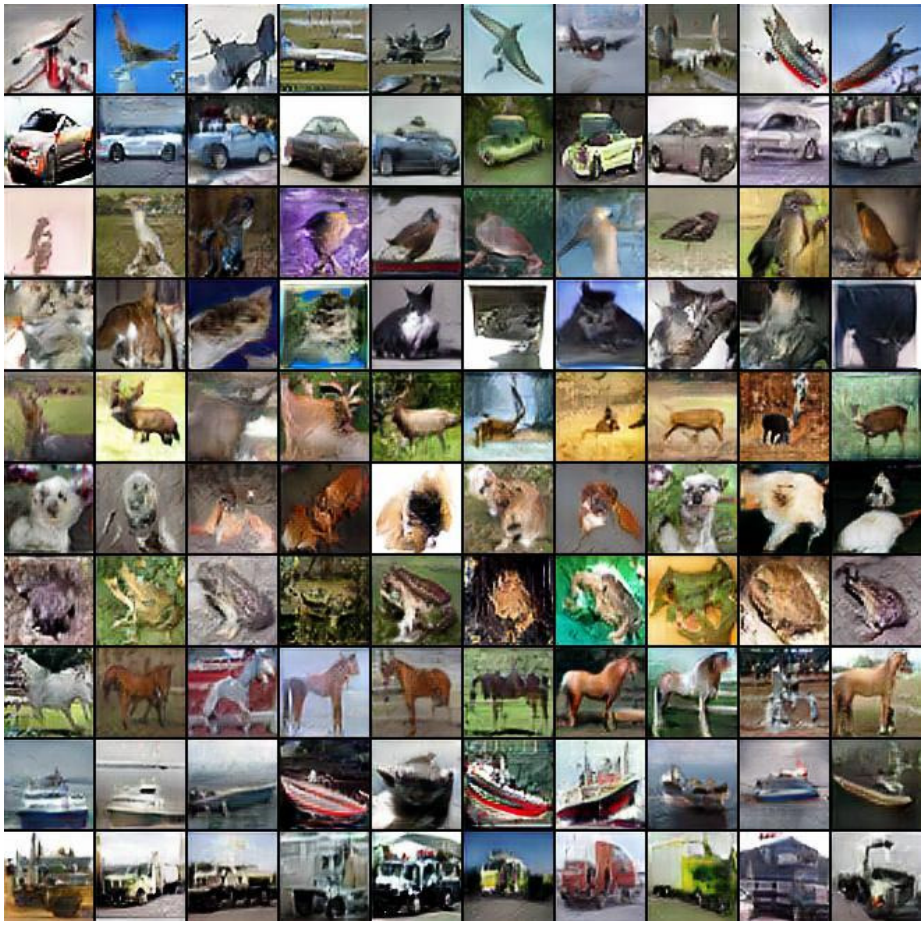

Figure 4: Generated CIFAR-10 images at $\mathbf{64 \times 64}$ pixels. From top to bottom: (1) Airplane, (2) Automobile, (3) Bird, (4) Cat, (5) Deer, (6) Dog, (7) Frog, (8) Horse, (9) Ship, (10) Truck.

4.3 STL-10

Figure 5 shows generated images at resolution of 128×128 pixels trained on STL-10.

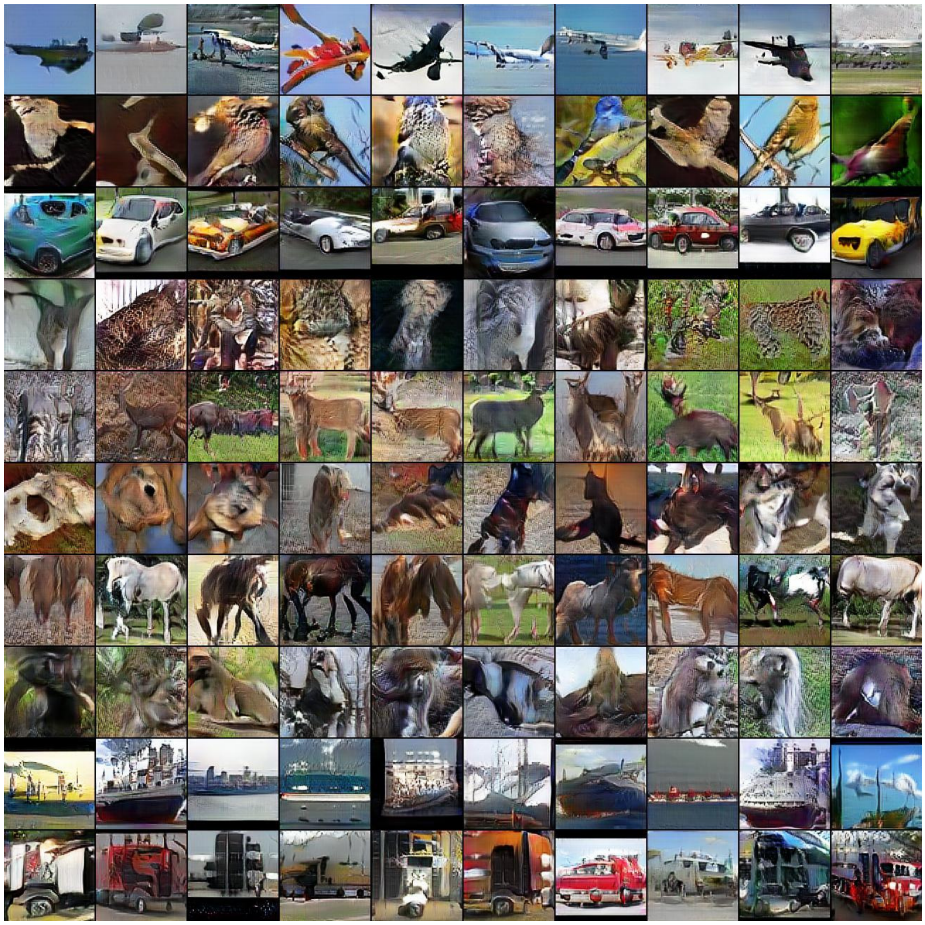

Figure 5: Generated STL-10 images at $\mathbf{128 \times 128}$ pixels. From top to bottom: (1) Airplane, (2) Bird, (3) Car, (4) Cat, (5) Deer, (6) Dog, (7) Horse, (8) Monkey, (9) Ship, (10) Truck.

4.4 CUB-200 birds

Figure 6 shows more generated CUB-200 images. Each sample represents one of the 200 bird species.

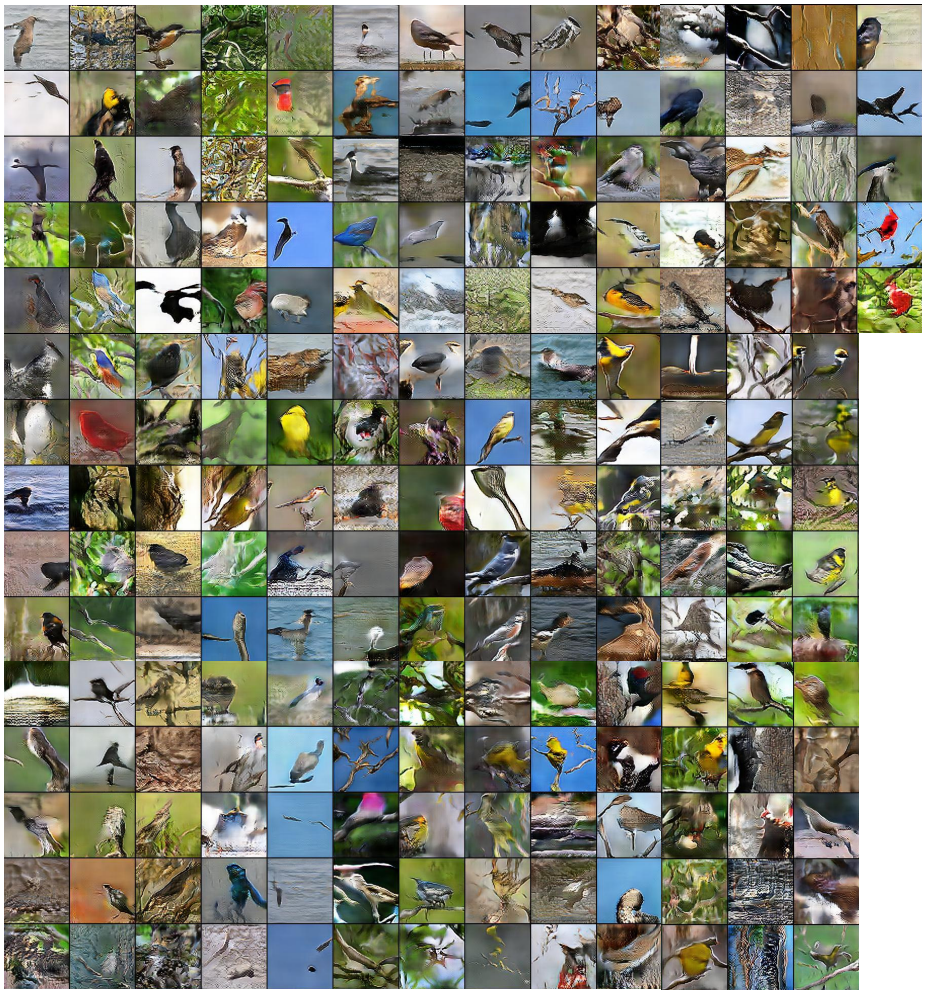

Figure 6: More generated images on CUB-200 birds at $\mathbf{128} \times \mathbf{128}$ pixels.

4.5 Oxford-102 flowers

Figure 7 shows more generated flower images on Oxford-102 at high resolution (128×128 pixels). Each sample represents one flowers species, with a total of 102 types of flowers generated.

Figure 7: More generated images on Oxford-102 flowers at $\mathbf{128 \times 128}$ pixels.

References

[1] D. Warde-Farley and Y. Bengio, "Improving generative adversarial networks with denoising feature matching," in *International Conference on Learning Representations*, 2017. 2