STYLEBANK: AN EXPLICIT REPRESENTATION FOR NEURAL IMAGE STYLE TRANSFER

b09902128 黃宏鈺 b09902135 賴豐彰

{b09902128, b09902135}@csie.ntu.edu.tw

1. ABSTRACT

We reproduced StyleBank[1], a kind of neural network composed of auto-encoder and multiple convolution filter banks, for image style transfer. Different from original work, we try to deepen it with DenseNet[2], and solve checkerboard artifacts and broken margin with Guided Filter[3][4].

2. INTRODUCTION

Nowadays, we can easily observe that more and more filters for photos have appeared. Thus, we are curious about how these filters work behind. Afterward, we found that there is a technique called style transfer. Thus this work was born.

StyleBank[1], a kind of neural network, is made for image style transfer. Its feature is that we do not need such complicated model such as GAN[5]. What we need is only an auto-encoder armed with multiple convolution filter banks. Another advantage is that once the auto-encoder was trained. We can fix it and change the banks in this model, creating a new style. Besides the original work, we also try to apply DenseNet[2] on it, to have a deeper train. Additionally, we also discover the checkerboard artifacts and broken margin problem in the original work. We then apply Guided Filter[3][4] to it, and solve it delicately.

3. BACKGROUND

3.1. Dense Net

We use DenseNets[2] in our encoder owing to following advantages: they alleviate the vanishing-gradient problem, strengthen feature propagation, encourage feature reuse, and substantially reduce the number of parameters. In our implementation, a DenseBlock can be present as following :
$D(in, n, g) = \{L_i = C_1(in + i \times g, 4g) \rightarrow C_3(4g, g) | i = 0, 1, ..., n-1\}$

$$\begin{cases} x_i & = L_i([x_0, x_1, .., x_i]) \\ x_0 & = input \end{cases} \rightarrow D(input) = [x_0, ..., x_n]$$

Where $C_3(in, out)$ is kernel size-3, stride-1, padding-1 convolution layer of input channel in and output channel out, $C_1(in, out)$ is kernel size-1, stride-1, padding-0 convolution layer of input channel in and output channel out.

3.2. Guided Filter

Guided filter[4] is a edge-preserving filter with linear time complexity. With a guided image I and input image p, guided filter trying to get output image q where

$$\begin{cases} q_i & = aI_i + b \\ q_i & = p_i - n_i \end{cases} \Rightarrow n_i = p_i - aI_i + b$$

By minimizing the cost function :

$$E(a_k, b_k) = \sum_{i \in w_k} ((a_k I_i + b_k - p_i)^2 + \epsilon a_k^2)$$

where w_k is a window center on kth pixel, ϵ is used to penalize big α. In our implementation, we use r represent radius of the window.

4. STYLEBANK NETWORKS

4.1. Network Architecture

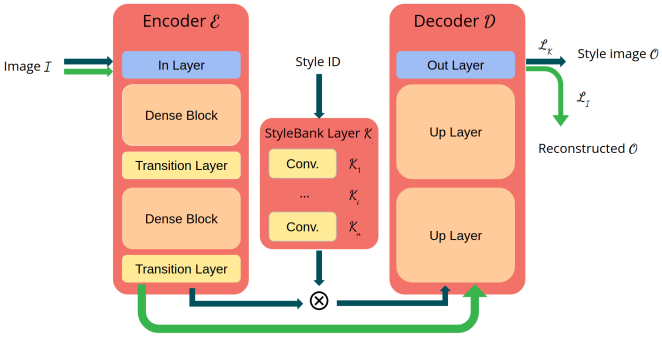


Figure 1.

Figure 1. shows our network architecture, which is roughly same as original StyleBank[1] containing three

modules : image encoder $\mathcal{E}$, *StyleBank* layer $\mathcal{K}$, and image decoder $\mathcal{D}$. There are two learning branches : auto-encoder (*i.e.* $\mathcal{E} \to \mathcal{D}$) and styling (*i.e.* $\mathcal{E} \to \mathcal{K} \to \mathcal{D}$), and both share the same encoder $\mathcal{E}$ and decoder $\mathcal{D}$.
For our network $\mathcal{N}$, the *content image* $\mathcal{I}$ and the *index of style* $\mathcal{ID}$ is required as input. First the image is transform into multi-layer features F by encoder $\mathcal{E}$:$F = \mathcal{E}(\mathcal{I})$. In auto-encoder branch, decoder $\mathcal{D}$ directly take features F as input trying to produce an image O that is as close as possible to input image $\mathcal{I}$ *i.e.* $O = \mathcal{D}(F) \to \mathcal{I}$
In styling branch, *StyleBank* layer $\mathcal{K}$ is added between $\mathcal{D}$ and $\mathcal{E}$. For n styles, each style will respectively correspond to a multi-layer convolution network in $\mathcal{K}$ *i.e.* $\mathcal{K} = \{\mathcal{K}_i | i = 1, 2, ..., n\}$. Transformed features $\tilde{F}_i$ for ith style is obtained by feeding features F into $\mathcal{K}_i$. Finally, taking $\tilde{F}_i$ as input of $\mathcal{D}$, we can get styled image O_i $i, e.$ $O_i = \mathcal{D}(\tilde{F}_i)$.
Our network $\mathcal{N}$ can be presented as follow :

$$\begin{cases} \mathcal{N}(\mathcal{I}) & = & \mathcal{D}(\mathcal{E}(\mathcal{I})) & = & O \to \mathcal{I} \\ \mathcal{N}(\mathcal{I}, \mathcal{ID}) & = & \mathcal{D}(\mathcal{K}_{\mathcal{ID}} \otimes \mathcal{E}(\mathcal{I})) & = & O_i \end{cases}$$

4.2. Encoder and Decoder

4.2.1. Encoder

We use dense block and transition layer from DenseNet[2] in our encoder to avoid gradient vanishing and gradient explored while deepening the module. Also, we hope dense block can transmit more information of sallow layers to compressed features. As Figure 1 showing, encoder is composed of an in-layer I, two dense block D_1&D_2 and two transition layer T_1&T_2, and can be presented as follow :

$$\mathcal{E} = IN \to D_1 \to T_1 \to D_2 \to T_2$$

$$\begin{cases} IN & = & C_3(3, 32) \to C_3(32, 64) \to MP \\ D_1 & = & D(64, 6, 32) \\ T_1 & = & C_1(256, 128) \to MP \\ D_2 & = & D(128, 6, 32) \\ T_2 & = & C_1(320, 256) \to MP \end{cases}$$

Where $C_3(in, out)$ is kernel size-3, stride-1, padding-1 convolution layer of input channel in and output channel out, $C_1(in, out)$ is kernel size-1, stride-1, padding-0 convolution layer of input channel in and output channel out, MP is kernel size-2, stride-1, padding-0 maxpooling layer and $D(in, n, g)$ is dense block with n layers and $k_0 = in$, growth rate *i.e.* $k = g$

4.2.2. Decoder

We use transpose convolution for upsampling, because we hope every sub-module is trainable except pooling layers. We don't use densenet in decoder because densenet will increase channels most of the time; instead, we simply apply multiple convolution layers. Our decoder $\mathcal{D}$ is composed of two up layers U_1&U_2 and an output layer Out, and can be presented as follow :

$$\mathcal{D} = U_1 \to U_2 \to OUT$$

$$\begin{cases} U_1 & = TC(256, 128) \\ & \quad \to \underbrace{C_3(128, 128) \to ... \to C_3(128, 128)}_{6} \\ U_2 & = TC(128, 64) \to \underbrace{C_3(64, 64) \to ... \to C_3(64, 64)}_{6} \\ Out & = TC(64, 32) \to C_3(32, 32) \to C_3(32, 32) \\ & \quad \to C_1(32, 3) \to Sigmoid \end{cases}$$

Where $TC(in, out)$ is kernel size-4, stride-2, padding-1 transpose convolution layer of input channel in and output channel out, C_1&C_3 are the same as above.

All convolutional layers are followed by instance normalization and a ReLU nolinearity except the last C_1 layer.

4.3. StyleBank Layer

In *StyleBank* layer $\mathcal{K}$, we train n convolution filters banks $\{K_i | i = 1, 2, ..., n\}$ where $K_i = C_3(256, 128) \to C_3(128, 128) \to C_3(128, 256)$. By decoding transformed features $\tilde{F}_i = K_i \otimes F$, output O_i is expected to have the same style as S_i. *StyleBank* layer is a interactive layer that allow us to either choose an existed filter or add a new one. While training a new style, we can uses original encoder $\mathcal{E}$ and decoder $\mathcal{D}$ and only train the convolution filters, which can save a lot training time.

4.4. Loss Functions

Two training branches have different loss function respectively, and in this part we are totally following the original StyleBank[1].
In auto-encoder branch(*i.e.* $\mathcal{E} \to \mathcal{D}$), we use MSE (Mean Square Error) between input image $\mathcal{I}$ and output image $\mathcal{O}$. The *identity loss* $\mathcal{L}_\mathcal{I}$ is given as follow :

$$\mathcal{L}_\mathcal{I}(\mathcal{I}, \mathcal{O}) = ||\mathcal{O} - \mathcal{I}||^2$$

In stylizing branch (*i.e.* $\mathcal{E} \to \mathcal{K} \to \mathcal{D}$), we use *perceptual loss* $\mathcal{L}_\mathcal{K}$ proposed in [6]. $\mathcal{L}_\mathcal{K}$ is composed of 3 different part: content loss $\mathcal{L}_c$, style loss $\mathcal{L}_s$ and variation regularization loss $\mathcal{L}_{tv}$

$$\mathcal{L}_\mathcal{K}(\mathcal{I}, S_i, \mathcal{O}_i) = \alpha \mathcal{L}_c(\mathcal{O}_i, \mathcal{I}) + \beta \mathcal{L}_s(\mathcal{O}_i, S_i) + \gamma \mathcal{L}_{tv}(\mathcal{O}_i)$$

$$\begin{cases} \mathcal{L}_c(\mathcal{O}_i, \mathcal{I}) & = \sum_{l \in \{l_c\}} ||F^l(\mathcal{O}_i) - F^l(\mathcal{I})||^2 \\ \mathcal{L}_s(\mathcal{O}_i, S) & = \sum_{l \in \{l_s\}} ||G(F^l(\mathcal{O}_i)) - G(F^l(S_i))||^2 \\ \mathcal{L}_{tv}(\mathcal{O}_i) & = \sum_{h,w} |\mathcal{O}_{i(h+1,w)} - \mathcal{O}_{i(h,w)}| \\ & \quad + |\mathcal{O}_{i(h,w+1)} - \mathcal{O}_{i(h,w)}| \end{cases}$$

Where $\mathcal{I}$, S_i, $\mathcal{O}_i$ are the input content image, style image and stylization result; F^l and G are feature map

and Gram matrix computed from layer l of pretrained VGG-16 network[7]. In our implementation $\{l_c\} = \{relu4\_2\}, \{l_s\} = \{relu1\_2,\ relu2\_2,\ relu3\_2,\ relu4\_2\}$.

4.5. Training Strategy

We use Algorithm 1 from the original StyleBank[1] to train our model. For every $T+1$ iterations, we train T times around the banks, and train 1 time on the auto-encoder. In this way, we can strike the balance between both auto-encoder branch and stylizing branch.

Algorithm 1 Jointly training Strategy.

for every $T+1$ iterations **do**
  // Training at branch $\mathcal{E} \rightarrow \mathcal{K} \rightarrow \mathcal{D}$
  for $t = 1$ *to* T **do**
    // Update $\mathcal{E},\ \mathcal{D},\ \mathcal{K}_j$(each style)
    // train style
  end for
  // Training at branch $\mathcal{E} \rightarrow \mathcal{D}$
  // Update $\mathcal{E}, \mathcal{D}$ only
  // train auto-encoder
end for

5. CAPABILITIES OF OUR NETWORK

Based on the network introduced in Section 4, in this part, we will introduce the capabilities of our network.

5.1. Incremental Training

Previous style transfer networks have to re-train the whole model for a new style, which is time-consuming. In contrast, an iterative optimization mechanism can have an ability of online-learning for a style. The structure StyleBank propose has both virtues of feed-forward neural network(FNN) and iterative optimization method. StyleBank enable an incremental training for new styles, which has comparable learning time to the online-learning method along with the efficiency of FNN. We first jointly train the auto-encoder and filter banks with Algorithm 1. After that, we can train new styles on the StyleBank layer with our auto-encoder fixed. The process converge fast since the StyleBank layer is updated in every iterations.

5.2. Result demo

5.2.1. Auto-encoder

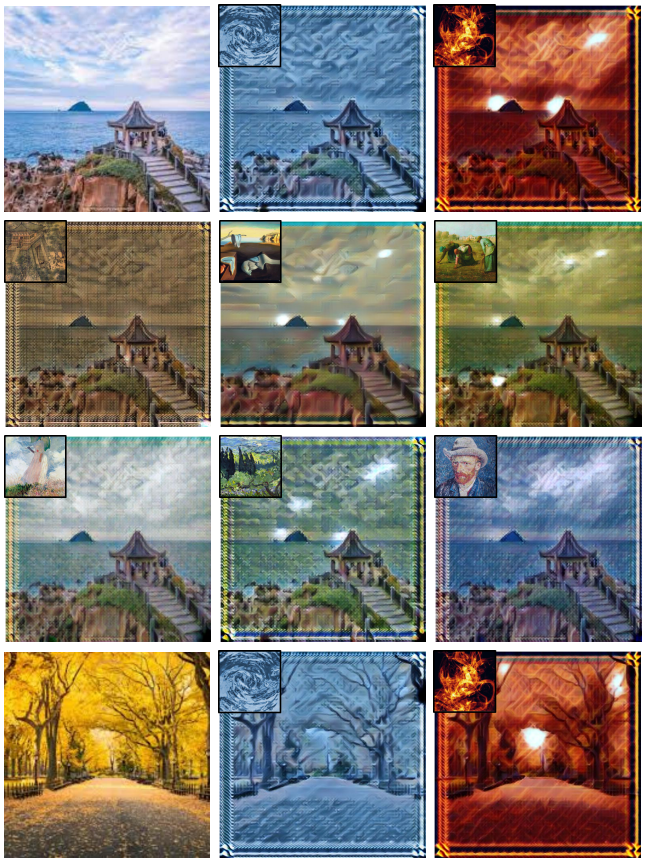
We show that in the Incremental Training strategy, our auto-encoder still performs well.

Origin Auto-encoder

Figure 2. Auto-encoder Demo

5.2.2. Style Transfer

We also show that in the same strategy, our style transfer performs well.



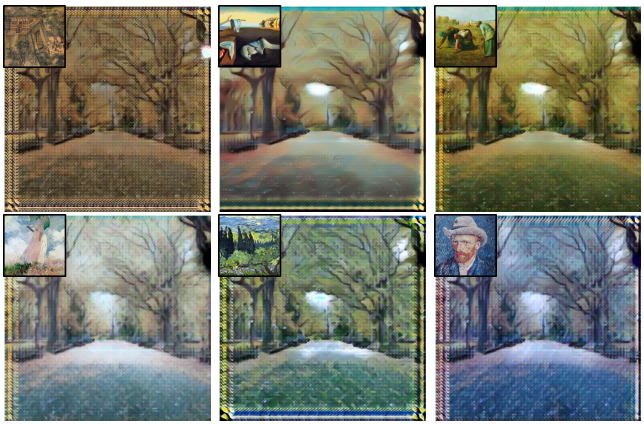


Figure 3. Style Transfer Demo

5.3. Shortcoming

In this part, we will introduce the shortcoming of out network.

5.3.1. Edge loss

Since out auto-encoder is based on convolution layer, it's inevitably that there are serious edge losses on our styled images. We also discover that the deeper we train, the more the edge losses.

Input Output

Figure 4. Edge loss

5.3.2. Checkerboard artifacts [8]

We also discover that there are duplicate color blocks due to deconvolution. That is, deconvolution has uneven overlap, causing the Checkerboard artifacts. It can be shown on Figure 5.

Input Output

Figure 5. Checkerboard artifacts

5.3.3. Too distinctive style

Since we fixed the ratio of α, β, γ in our loss function, when there comes a distinctive style e.g. extraordinary bright. We may have a strange result. See Figure 6.

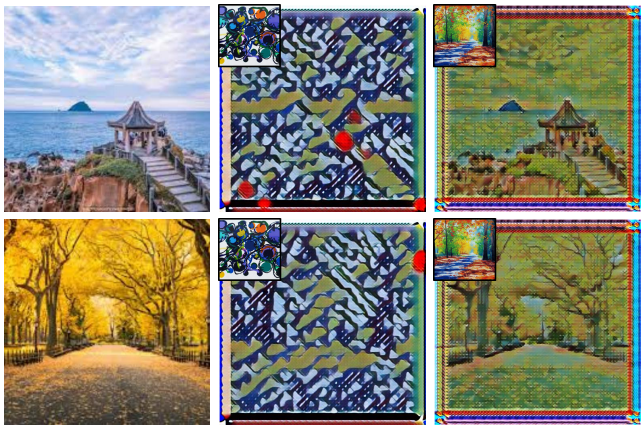


Figure 6. Extraordinary Style

6. EXPERIMENTS

6.1. Ratio of α, β, γ in loss function

6.1.1. Big α

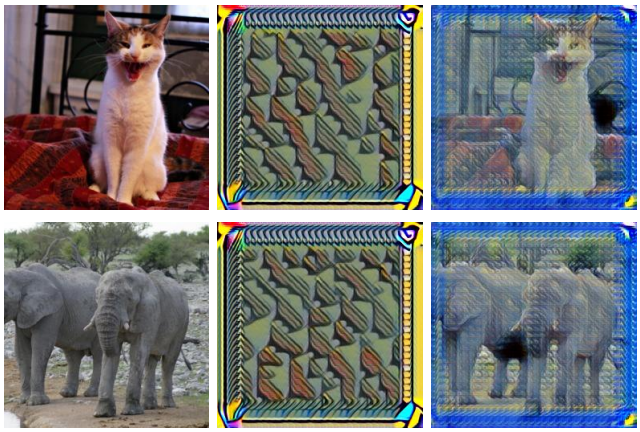
Once we use a too big *alpha* e.g. $\alpha : \beta = 1 : 1$, we can have a image with only content and no style on it. See Figure 7.

Input Style 1 Style 2

Figure 7. big alpha

6.1.2. Big β

We also try on big β value e.g. $\alpha : \beta = 1 : 10^6$, the result will be a image with a lot of strange hole on it. See Figure 8.

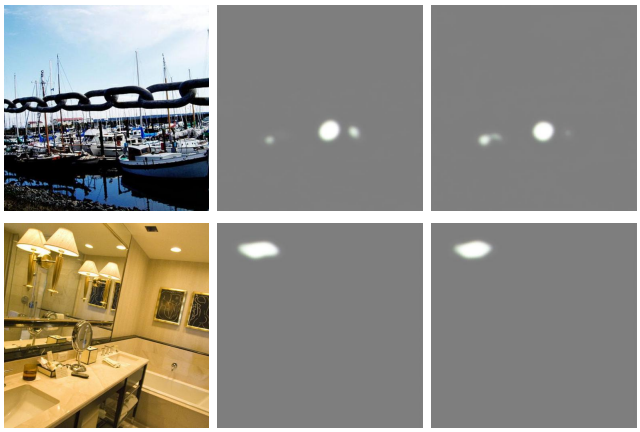


Input Style 1 Style 2

Figure 8. big beta

6.1.3. Big γ

We also experiment on a big γ value e.g. $\gamma = 1$, it them comes with a black image with several light dots. Which is apparently not we want. See Figure 9.



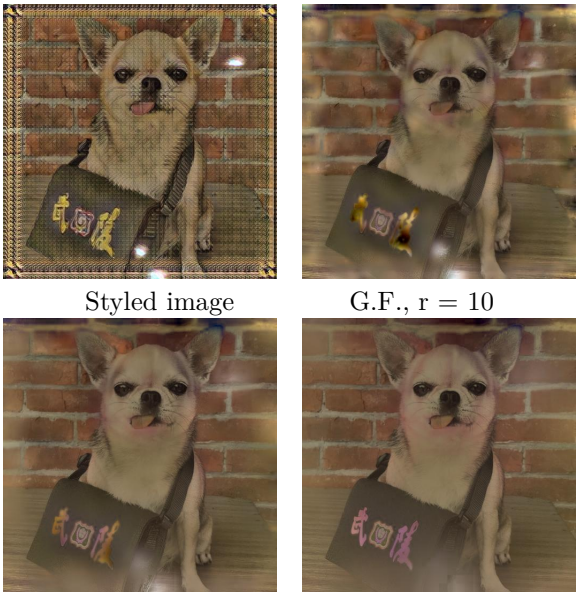
Input Style 1 Style 2

Figure 9. big gamma

6.2. Guided Filter

6.2.1. Guided Filter

In order to fix broken margin, we implement Guided Filter [4], a filter that can preserve the edge of guided image and denoise the input image. In our case, we take content image $\mathcal{I}$ as guided image and styled image O_i as input image.



Styled image G.F., r = 10

G.F.,r = 20 G.F., r = 40

Figure 10. Guided Filter

Guided Filter can fix checkerboard artifacts even with little r, and the bigger r comes the more smooth edge as well. However, big r may cause color blending together due to the large window for solving the equation. For example, the words on the bag becomes purple when $r = 40$.

6.2.2. Fast Guided Filter

The original guided filter can be implemented in $O(N)$ time. However, fast guided filter [3] resizing the input and guided image by s can be accelerated to $O(N/s^2)$ time.

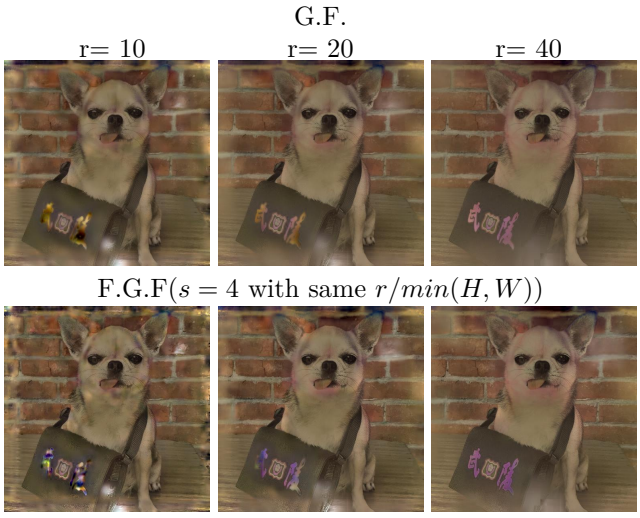


Figure 11. Compare G.F. & F.G.F.
Because of downsampling, fast guided filter may loss more detail than original filter while reducing time.

However, we still recommend to use fast guided filter for the concern of time.

7. DISCUSSION AND CONCLUSION

In this paper, We create a style transfer model, which can produce a style in a short time. Besides, once the model was trained, we can train new style in a even shorter time. We show that we can combine one images content with the other's style. Our work also contains revising part of the origin model from convolutional layer to Dense Block[2]. We also try to solve the edge loss and checkboard artifacts with guided filter[4][3].

Therefore, there are still some issues for further investigation. For instance, we do not directly solve the checkboard artifacts and edge loss. Instead, we solve it with a guided filter, which is not the best solution to it. And also, a special image with bright client or distinguished style can have a bad result image.

And our result is push to `https://github.com/Alx-Lai/2022_ICG_Final`

8. REFERENCES

[1] Jing Liao Nenghai Yu Dongdong Chen, Lu Yuan and Gang Hua, "Stylebank: An explicit representation for neural image style transfer," in *Proceedings Title*. University of Science and Technology of China, Microsoft Research Asia, 2017.

[2] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q Weinberger, "Densely connected convolutional networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[3] Kaiming He and Jian Sun, "Fast guided filter," *CoRR*, vol. abs/1505.00996, 2015.

[4] Kaiming He, Jian Sun, and Xiaoou Tang, "Guided image filtering," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 6, pp. 1397–1409, 2013.

[5] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio, "Generative Adversarial Networks," June 2014.

[6] Justin Johnson, Alexandre Alahi, and Li Fei-Fei, "Perceptual losses for real-time style transfer and super-resolution," *CoRR*, vol. abs/1603.08155, 2016.

[7] Karen Simonyan and Andrew Zisserman, "Very deep convolutional networks for large-scale image recognition," 2015.

[8] Augustus Odena, Vincent Dumoulin, and Chris Olah, "Deconvolution and checkerboard artifacts," *Distill*, 2016.