ARTEMIS

James Baker
Computer Science Department
Rutgers University
jlbaker361@gmail.com

Abstract

We propose a novel method for generating textures. First an autoencoder is trained to encode and decode the style representations of images, which are extracted from source images with a pretrained VGG network. Then, the decoder component of the autoencoder is extracted and used as a generator in a GAN. The generator works with an ensemble of discriminators. Each discriminator is trained to classify a different style representation, and the generator is trained to create images that create convincing style representations in order to deceive all of the generators. The generator is also trained to maximize a diversity term. The resulting images had a surreal, geometric quality. We call our approach ARTEMIS (**ART**istic **E**ncoder **M**ethod **I**ncluding **S**elf-Attention), as it uses the self-attention layers and an encoder-decoder architecture.

Introduction

Related Work

Style Transfer

Approaches that didn't use neural networks, like (Hertzmann 1998), which could redraw the edges in images to look painted, or (Gooch, Reinhard, and Gooch 2004), which could make human faces appear more cartoon-like, were usually limited in the range of styles they could apply to images. The first style transfer algorithm using deep learning was first implemented by (Gatys, Ecker, and Bethge 2015). In this paper, style and content representations were extracted from images using the output of intermediate layers of a pretrained VGG-19 network. An image was optimized to minimize the distance between the gramm matrices of the generated image and the gramm matrices of the style and content representations. By adding a photorealistic regularization term to the objective function when optimizing the image, style could be transferred to a content image without losing the realism of the (Luan et al. 2017). While a pretrained VGG-19 model is standard for extracting feature representations of images, work has been made to improve the utility of the lighter ResNet for extracting feature representations as well (Wang, Li, and Vasconcelos 2021).

Instead of optimizing an image, GAN approaches like (Zhu et al. 2017) used a CycleGAN to generate stylized images, using content images as the source to the generator, expanded upon by (Liu, Michelini, and Zhu 2018). Style transfer with GANs have also implemented attentional (Park and Lee 2019) and residual (Xu et al. 2021) components.

GANs for Image Generation

Originally introduced by (Goodfellow et al. 2014), Generative Adversarial Networks (GANS), and an expansion on them, the Deep Convolutional GAN (DCGAN) (Radford, Metz, and Chintala 2015) have been used for writing text (Saeed, Ilic, and Zangerle 2019), composing melodies (Kolokolova et al. 2020), making 3-D models (Wu et al. 2018) and super-resolution (Wang et al. 2018). (Elgammal et al. 2017) optimized the generator to not only create convincing samples, but also to make it hard to classify generated samples to belong to a particular artistic category like baroque or impressionism. Works like (Ulyanov et al. 2016; **?**) conditioned their GANs on sketches to generate realistic images.

Texture Generation

A texture is the " the perceived surface quality of a work of art" (Wikipedia 2022). Alternatively called style, it refers to things like the shape of lines, thickness of strokes and visual features other than the raw content of an image. Texture Generation is the process of generating new images that have unique textures, distinct from texture synthesis, which is "from an input sample, new texture images of arbitrary size and which are perceptually equivalent to the sample" (Raad et al. 2017). In addition to many traditional ways to generate unique textures (Dong et al. 2020), machine learning can be used to generate new textures. Works like (Li and Wand 2016), (Johnson, Alahi, and Fei-Fei 2016) and (Ulyanov et al. 2016) used convolutional networks trained to minimize higher-level stylistic loss to generate novel textures. This method was advanced by (Wilmot, Risser, and Barnes 2017) via training the network to minimize histogram loss. While most work has been for flat images, (Gutierrez et al. 2020) expanded texture synthesis to 3-dimensional objects, mapping each point in 3-dimensional space to a color.

Transfer Learning

Network-based transfer learning uses a model, or components of a model trained for one task to perform another task,

for a new task (Caruana 1994; Tan et al. 2018). (Wen et al. 2019) used a pretrained VGG-19 (Simonyan and Zisserman 2015) network to extract features and use them to classify mechanical data. Using a pretrained InceptionV3 network, (Jignesh Chowdary et al. 2020) determined whether a human was wearing a mask or not. (Espejo-Garcia et al. 2020) compared VGG and Inception for classifying weeds. The pretrained model does not need to be trained on a different dataset than the final model. For example, GANs can be improved by using pretrained layers of an autoencoder (Nagpal et al. 2020) or variational autoencoder (Ham, Jun, and Kim 2020) as part of the generator.

Visual Attention

When humans process input from a sequence or an image, we contextualize each part of the input using other parts of the input. However, we do not pay equal attention to every other part of the input (Mnih et al. 2014). For a textual example, in the sentence "Alan said he was hungry", the word "alan" is more useful in determining the meaning of "he" than the word "hungry". For a visual example, in order to guess the location of the right eye on someones face, the most important information would be the location of the left eye, not the length of their beard. Attention, also known as Non-Locality was originally proposed for text (Bahdanau, Cho, and Bengio 2014; Luong, Pham, and Manning 2015), it was then applied to vision (Wang et al. 2017), finding applications in medicine (Guan et al. 2018), super-resolution (Dai et al. 2019; **?**), and image generation (Gregor et al. 2015; Zhang et al. 2019).

Data

Preprocessing

The images used were the 80,000+ images in the WikiArt Images dataset (Saleh and Elgammal 2015). Each image was cropped to be $(256 \times 256 \times 3)$. Images that were smaller than $(256 \times 256 \times 3)$ were concatenated with themselves and then cropped. Images were preprocessed by using the output of intermediate layers of a VGG-19 network pretrained on the ImageNet dataset. Only the 16 convolutional layers were used. These image style representations were usually different shapes than the original images. For example, a $(256 \times 256 \times 3)$ image would create a $(32 \times 32 \times 512)$ style representation output from the "block4_conv1" layer.

Network

VGG Layers

Given the pretrained VGG-19 model, VGG, we define a set of its intermediate blocks B, and a set of corresponding dimensions $DIMS$ corresponding to the output of the corresponding block when the VGG network processes an image $\in \mathbb{R}^{256\times256\times3}$ image. For example $B = (block1\_conv1, block3\_conv1, block5\_conv1)$, and $DIMS_B = (\mathbb{R}^{256\times256\times64}, \mathbb{R}^{64\times64\times256}, \mathbb{R}^{16\times16\times12})$. We then use the notation $VGG(B)$ to mean the style representation outputs of each layer in B when the VGG processes an image, and therefore

$$VGG_B : \mathbb{R}^{256\times256\times3} \longrightarrow DIMS_B$$

For example:

$$VGG_{(block1\_conv1, block5\_conv1)} : \mathbb{R}^{256\times256\times3} \longrightarrow (\mathbb{R}^{256\times256\times64}, \mathbb{R}^{16\times16\times12})$$

When B is a singleton set, consisting of a single block, we may also write $VGG_{blockname}$ or $DIMS_{blockname}$, for example:

$$VGG_{block3\_conv1} : \mathbb{R}^{256\times256\times3} \longrightarrow DIMS_{block3\_conv1}$$

VGG Block	DIM_{block}
no block	$\mathbb{R}^{256\times256\times3}$
block1_conv1	$\mathbb{R}^{256\times256\times256}$
block2_conv1	$\mathbb{R}^{128\times128\times128}$
block3_conv1	$\mathbb{R}^{64\times64\times256}$
block4_conv1	$\mathbb{R}^{32\times32\times512}$
block5_conv1	$\mathbb{R}^{16\times16\times512}$

Table 1: VGG Layers and Dimensions

Autoencoder

The Autoencoder consisted of three parts. First, the encoder E mapped an image or an image style representation to a latent space representation $z \in \mathbb{R}^{2\times2\times64}$. Given the shape of an image or image representation DIM_{block}, where block is the block of the VGG-19 network that produced the style representation, (for example, the shape of the image style representation from $block1\_conv1$ of the VGG-19 $\in \mathbb{R}^{256\times256\times64}$), we can write:

$$Encoder_{block} : DIM_{block} \longrightarrow \mathbb{R}^{2\times2\times64}$$

The second part was a decoder that mapped the latent noise to an image.

$$Decoder : \mathbb{R}^{2\times2\times64} \longrightarrow \mathbb{R}^{256\times256\times3}$$

The final part was the pretrained VGG_{block}.
The Autoencoder was defined as:

$$Autoencoder_{block} = VGG_{block}(Decoder(Encoder_{block}))$$

$$Autoencoder_{block} : DIM_{block} \longrightarrow DIM_{block}$$

The Encoder used convolutional layers to downsample, and the decoder used Convolutional Transpose blocks to upsample. After each Convolutional and Convolutional Transpose Block, we used Batch Normalization, to add noise (Ioffe and Szegedy 2015) and stabilize the gradients (Santurkar et al. 2019), followed by LeakyReLu. While we concede that the swish (Ramachandran, Zoph, and Le 2017) and related mish (Misra 2019) activation functions have performed better in some tasks, LeakyReLu was used for the most part due to it being less computationally expensive than the alternatives.

Adversarial Components

To build the generator, we extracted the decoder after pre-training the autoencoder, and attached a VGG_B instance, where B was the set of blocks we wanted to study.

$$Generator_B = VGG_B(Decoder)$$
$$Generator_B : \mathbb{R}^{2\times2\times64} \longrightarrow DIMS_B$$

For each $block \in B$, we had a separate discriminator, that would take an image or style representation and output a probability between 0 and 1

$$Discriminator_{block} : DIM_{block} \longrightarrow [0, 1]$$

Experiments

Loss Function

The autoencoder was trained to minimize reconstruction loss between an image (or style representation) x and the reconstructed image (or style representations) x'

$$\mathcal{L}_{AE} = \sum_{i}^{N} ||x_i - x_i'||_F$$

The diversity loss penalized the generator for outputting samples that were too similar (Li et al. 2017). Given generated samples $G(z_i), G(z_j), , , G(z_k)$, diversity loss was defined

$$\mathcal{L}_{DV} = -\sum_{i}^{k} \frac{||G(z_i) - G(z_j)||_F}{||z_i - z_j||_F}, i \neq j$$

In experiments, for each training step, we used k=3 generated samples.

In traditional GAN fashion (Goodfellow et al. 2014), for each sample, real, x_i or generated $G(z_i)$, the Discriminator output a value from 0 to 1, D_A, corresponding to its confidence that the sample was real. The Discriminator Adversarial loss was defined, for each $Discriminator_{block}$ for $block \in B$

$$\mathcal{L}_{D_{block}} = \sum_{i}^{N} log(D_{block}(x_i)) + (1 - log(D_{block}(G(z_i)))$$

The generator was trained to fool each Discriminator, so its loss function was defined

$$\mathcal{L}_{GA} = -\sum_{block}^{B}\sum_{i}^{N} log(D_{block}(G(z_i))$$

The total Generator loss to be minimized was thus

$$\mathcal{L}_G = \mathcal{L}_{CE} + \mathcal{L}_{DV} + \mathcal{L}_{GA}$$

Training

The autoencoders were trained for 250 epochs. The Adversarial loop was trained for 300 epochs. The batch size was 8. We added gaussian noise $\sim \mathcal{N}(0, 0.01)$ to labels for binary classification in order to implement label smoothing (Salimans et al. 2016; per 2017), to reduce the vulnerability of the network to adversarial samples.

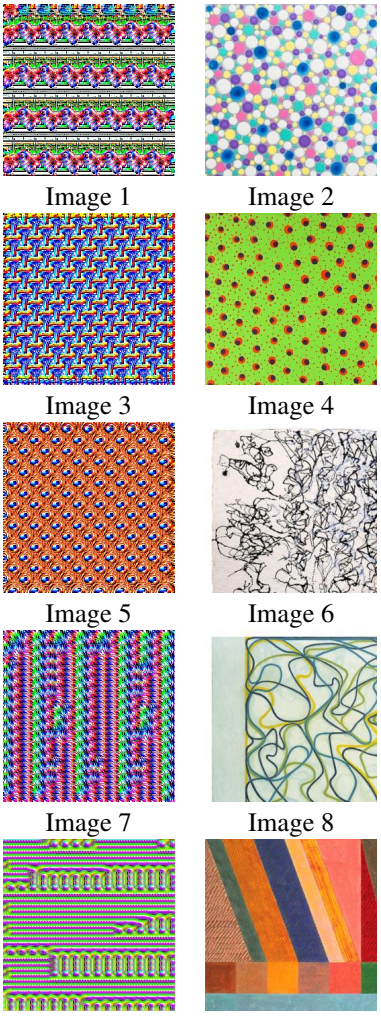

Figure 1: The Images used for Comparison

Subjective Results

Beauty is in the eye of the beholder, so we decided to evaluate the generated art by comparing it to actual abstract art in the WikiArt dataset. Figure 1 shows the 5 artificial images, all indexed with odd numbers and in the left column, and the 5 manmade images, all indexed with even numbers, in the right column.

An online survey was given out, where 54 participants were asked to rate each image on a scale of 1-10, 10 being the most positive and 1 being the most negative. For each image the average rating and standard deviation are given in table 2. Notably, while the artificial images had much lower ratings, there was less variance in their ratings, suggesting the value of the ARTEMIS model is to produce images that humans will have a predictable reaction to.

Image	μ	σ
Image 1	3.37	2.18
Image 2	5.03	2.63
Image 3	3.64	2.36
Image 4	3.94	2.32
Image 5	3.37	2.19
Image 6	6.26	2.16
Image 7	3.45	2.05
Image 8	6.59	2.53
Image 9	3.72	2.31
Image 10	6.93	2.25

Table 2: Caption

Conclusion

Contributions

This paper proposed a novel method of using generative adversarial networks with an ensemble of discriminators, each trained to classify images extracted from a different layer of the pretrained VGG-19 network. Trained on the WikiArt dataset, the network generated images that were akin to abstract art. Compared to manmade art, the artificially generated art was not as well received but there was less variability in how people reacted to the art.

References

Bahdanau, D.; Cho, K.; and Bengio, Y. 2014. Neural machine translation by jointly learning to align and translate. cite arxiv:1409.0473Comment: Accepted at ICLR 2015 as oral presentation.

Caruana, R. 1994. Learning many related tasks at the same time with backpropagation. In *Proceedings of the 7th International Conference on Neural Information Processing Systems*, NIPS'94, 657–664. Cambridge, MA, USA: MIT Press.

Dai, T.; Cai, J.; Zhang, Y.; Xia, S.-T.; and Zhang, L. 2019. Second-order attention network for single image super-resolution. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 11057–11066.

Dong, J.; Liu, J.; Yao, K.; Chantler, M.; Qi, L.; Yu, H.; and Jian, M. 2020. Survey of procedural methods for two-dimensional texture generation. *Sensors* 20(4).

Elgammal, A. M.; Liu, B.; Elhoseiny, M.; and Mazzone, M. 2017. CAN: creative adversarial networks, generating "art" by learning about styles and deviating from style norms. *CoRR* abs/1706.07068.

Espejo-Garcia, B.; Mylonas, N.; Athanasakos, L.; Fountas, S.; and Vasilakoglou, I. 2020. Towards weeds identification assistance through transfer learning. *Computers and Electronics in Agriculture* 171:105306.

Gatys, L. A.; Ecker, A. S.; and Bethge, M. 2015. A neural algorithm of artistic style. *CoRR* abs/1508.06576.

Gooch, B.; Reinhard, E.; and Gooch, A. 2004. Human facial illustrations: Creation and psychophysical evaluation. *ACM Trans. Graph.* 23(1):27–44.

Goodfellow, I. J.; Pouget-Abadie, J.; Mirza, M.; Xu, B.; Warde-Farley, D.; Ozair, S.; Courville, A.; and Bengio, Y. 2014. Generative adversarial networks.

Gregor, K.; Danihelka, I.; Graves, A.; and Wierstra, D. 2015. DRAW: A recurrent neural network for image generation. *CoRR* abs/1502.04623.

Guan, Q.; Huang, Y.; Zhong, Z.; Zheng, Z.; Zheng, L.; and Yang, Y. 2018. Diagnose like a radiologist: Attention guided convolutional neural network for thorax disease classification. *CoRR* abs/1801.09927.

Gutierrez, J.; Rabin, J.; Galerne, B.; and Hurtut, T. 2020. On demand solid texture synthesis using deep 3d networks. *CoRR* abs/2001.04528.

Ham, H.; Jun, T. J.; and Kim, D. 2020. Unbalanced gans: Pre-training the generator of generative adversarial network using variational autoencoder. *CoRR* abs/2002.02112.

Hertzmann, A. 1998. Painterly rendering with curved brush strokes of multiple sizes. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '98, 453–460. New York, NY, USA: Association for Computing Machinery.

Ioffe, S., and Szegedy, C. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR* abs/1502.03167.

Jignesh Chowdary, G.; Punn, N. S.; Sonbhadra, S. K.; and Agarwal, S. 2020. Face mask detection using transfer learning of inceptionv3. In Bellatreche, L.; Goyal, V.; Fujita, H.; Mondal, A.; and Reddy, P. K., eds., *Big Data Analytics*. Springer International Publishing.

Johnson, J.; Alahi, A.; and Fei-Fei, L. 2016. Perceptual losses for real-time style transfer and super-resolution. *CoRR* abs/1603.08155.

Kolokolova, A.; Billard, M.; Bishop, R.; Elsisy, M.; Northcott, Z.; Graves, L.; Nagisetty, V.; and Patey, H. 2020. Gans & reels: Creating irish music using a generative adversarial network. *CoRR* abs/2010.15772.

Li, C., and Wand, M. 2016. Combining markov random fields and convolutional neural networks for image synthesis. *CoRR* abs/1601.04589.

Li, Y.; Fang, C.; Yang, J.; Wang, Z.; Lu, X.; and Yang, M.-H. 2017. Diversified texture synthesis with feed-forward networks. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* 266–274.

Liu, H.; Michelini, P. N.; and Zhu, D. 2018. Artsy-gan: A style transfer system with improved quality, diversity and performance. In *2018 24th International Conference on Pattern Recognition (ICPR)*, 79–84.

Luan, F.; Paris, S.; Shechtman, E.; and Bala, K. 2017. Deep photo style transfer. *CoRR* abs/1703.07511.

Luong, M.; Pham, H.; and Manning, C. D. 2015. Effective approaches to attention-based neural machine translation. *CoRR* abs/1508.04025.

Misra, D. 2019. Mish: A self regularized non-monotonic neural activation function. *CoRR* abs/1908.08681.

Mnih, V.; Heess, N.; Graves, A.; and Kavukcuoglu, K. 2014. Recurrent models of visual attention. *CoRR* abs/1406.6247.

Nagpal, S.; Verma, S.; Gupta, S.; and Aggarwal, S. 2020. A guided learning approach for generative adversarial networks. *2020 International Joint Conference on Neural Networks (IJCNN)* 1–8.

Park, D. Y., and Lee, K. H. 2019. Arbitrary style transfer with style-attentional networks. *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* 5873–5881.

2017. *Adversarial Perturbations of Deep Neural Networks*. 311–342.

Raad, L.; Davy, A.; Desolneux, A.; and Morel, J. 2017. A survey of exemplar-based texture synthesis. *CoRR* abs/1707.07184.

Radford, A.; Metz, L.; and Chintala, S. 2015. Unsupervised representation learning with deep convolutional generative adversarial networks. cite arxiv:1511.06434Comment: Under review as a conference paper at ICLR 2016.

Ramachandran, P.; Zoph, B.; and Le, Q. V. 2017. Searching for activation functions. *CoRR* abs/1710.05941.

Saeed, A.; Ilic, S.; and Zangerle, E. 2019. Creative gans for generating poems, lyrics, and metaphors. *CoRR* abs/1909.09534.

Saleh, B., and Elgammal, A. M. 2015. Large-scale classification of fine-art paintings: Learning the right metric on the right feature. *CoRR* abs/1505.00855.

Salimans, T.; Goodfellow, I. J.; Zaremba, W.; Cheung, V.; Radford, A.; and Chen, X. 2016. Improved techniques for training gans. *CoRR* abs/1606.03498.

Santurkar, S.; Tsipras, D.; Ilyas, A.; and Madry, A. 2019. How does batch normalization help optimization?

Simonyan, K., and Zisserman, A. 2015. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*.

Tan, C.; Sun, F.; Kong, T.; Zhang, W.; Yang, C.; and Liu, C. 2018. A survey on deep transfer learning. *CoRR* abs/1808.01974.

Ulyanov, D.; Lebedev, V.; Vedaldi, A.; and Lempitsky, V. S. 2016. Texture networks: Feed-forward synthesis of textures and stylized images. *CoRR* abs/1603.03417.

Wang, X.; Girshick, R. B.; Gupta, A.; and He, K. 2017. Non-local neural networks. *CoRR* abs/1711.07971.

Wang, X.; Yu, K.; Wu, S.; Gu, J.; Liu, Y.; Dong, C.; Loy, C. C.; Qiao, Y.; and Tang, X. 2018. ESRGAN: enhanced super-resolution generative adversarial networks. *CoRR* abs/1809.00219.

Wang, P.; Li, Y.; and Vasconcelos, N. 2021. Rethinking and improving the robustness of image style transfer. *CoRR* abs/2104.05623.

Wen, L.; Li, X. Y.; Li, X.; and Gao, L. 2019. A new transfer learning based on vgg-19 network for fault diagnosis. *2019 IEEE 23rd International Conference on Computer Supported Cooperative Work in Design (CSCWD)* 205–209.

Wikipedia. 2022. Texture (visual arts) –Wikipedia, the free encyclopedia. https://en.wikipedia.org/wiki/Texture_(visual_arts). [Online; accessed 16-February-2022].

Wilmot, P.; Risser, E.; and Barnes, C. 2017. Stable and controllable neural texture synthesis and style transfer using histogram losses. *CoRR* abs/1701.08893.

Wu, J.; Zhang, C.; Zhang, X.; Zhang, Z.; Freeman, W. T.; and Tenenbaum, J. B. 2018. Learning shape priors for single-view 3d completion and reconstruction. *CoRR* abs/1809.05068.

Xie, S.; Girshick, R. B.; Dollár, P.; Tu, Z.; and He, K. 2016. Aggregated residual transformations for deep neural networks. *CoRR* abs/1611.05431.

Xu, W.; Long, C.; Wang, R.; and Wang, G. 2021. DRB-GAN: A dynamic resblock generative adversarial network for artistic style transfer. *CoRR* abs/2108.07379.

Zhang, H.; Goodfellow, I.; Metaxas, D.; and Odena, A. 2019. Self-attention generative adversarial networks. In Chaudhuri, K., and Salakhutdinov, R., eds., *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, 7354–7363. PMLR.

Zhu, J.; Park, T.; Isola, P.; and Efros, A. A. 2017. Unpaired image-to-image translation using cycle-consistent adversarial networks. *CoRR* abs/1703.10593.

Architecture

Encoder

Though there were a few different input shapes, we made the encoder map them all to a standard noise size $\mathbb{R}^{2\times2\times64}$. The encoder was made of encoder blocks, parameterized by c, an integer representing the number of output channels in the encoder block, and s, the size of the stride:

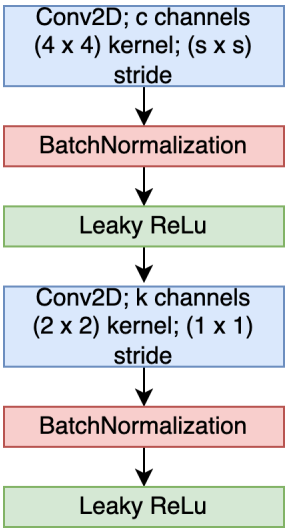

Figure 2: Encoder Block

First we wanted to make sure every layer had 64 channels. For input dim $\mathbb{R}^{256\times256\times3}$, we started with a convolutional block with 8 channels and kernel size $(8 \times 8$ with stride $(1 \times 1$, followed by BatchNormalization and Leaky ReLu. Then we had 3 encoder blocks, each with $s = 1$, with $c = 16, 32, 64$. For all the other input dimensions (see table 1), we held $s = 1$ constant, and added layers where $c = \frac{1}{2}$ of the output channels of the last layer, until the output had 64 channels.

Then we wanted to shrink the spatial dimensions to be (2×2). Holding c constant, we used $s = 2$, (this would halve the spatial dimensions each time), and applied encoder blocks until the output was $(2 \times 2 \times 64)$. We then added normal noise $\sim \mathcal{N}(0, 1)$. Adding this noise will make the decoder more robust and lessen the chance of overfitting.

Decoder

The decoder mapped the noise to an image $\mathbb{R}^{2\times2\times64} \rightarrow \mathbb{R}^{256x256x3}$. It was mainly composed of batch decoder blocks shown in figure 3, parameterized by the boolean $attention \in$ True||False, and c, the number of output channels.

The decoder also contained group decoder blocks, show in figure 4, parameterized by c, the number of output channels, and k, the size of the kernel.

The decoder consisted of 7 batch decoder blocks, where for each block, $c = \max(32, c')$, where c' was the amount of output channels in the prior batch decoder block, or the amount of channels in the input in the case of the first batch

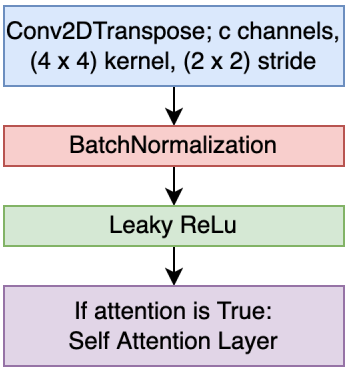

Figure 3: Batch Decoder Block

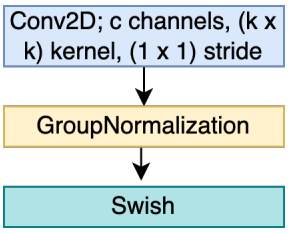

Figure 4: Group Decoder Block

decoder block, and $attention$ was True for the first 4 blocks, and False for the last 3 batch decoder blocks. Following that, there were 3 group decoder blocks, with $c = (16, 8, 4$ and $k = (8, 8, 4)$, respectively. Finally, there was a Convolutional Layer with 3 output channels, a sigmoid activation function, and a rescaling layer to bound the values between 0 and 255, thus producing an image.

Discriminator

The discriminator(s) were composed of discriminator blocks shown in figure 5, parameterized by c, the number of output channels and $resnext$, a boolean of whether to use a ResNext layer (Xie et al. 2016).

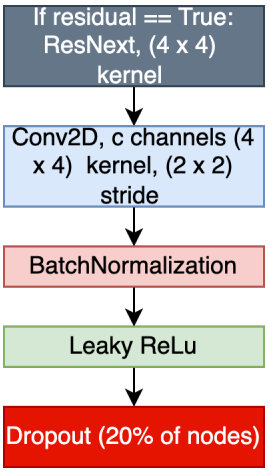

Figure 5: Discriminator Block

First, the discriminator had 4 discriminator blocks, each with $c = \frac{1}{2}c'$, where c' was the amount of channels in the prior block, or the amount of channels in the input in the case of the first discriminator block, and for all but the first discriminator block, $resnext$ was True. After that, we flatten the input. Then we apply a fully-connected layer with 8 nodes, batch normalization, leaky ReLu, then the final dense layer with 1 node and sigmoid activation; this final node serves to output the (0,1) label.