Gated-GAN: Adversarial Gated Networks for Multi-Collection Style Transfer

Brandon Benitt
A20416733

Kajol Tanesh Shah
A20496724

Department of Computer Science
Illinois Institute of Technology
Chicago, IL - 60608

Responsibilities/Contributions

Tasks	Name
Data Loading and Preprocessing	Brandon
Building the Generator	Kajol
Building the Discriminator	Brandon
Cycle-GAN Model and Training Loop	Brandon, Kajol
Gated-GAN Model and Training Loop	Brandon, Kajol
Writing the code to train the model and print intermediate results	Brandon
FID Score/Evaluation	Kajol
Writing the paper	Brandon, Kajol
Create Report and Presentation	Brandon, Kajol

Abstract

Style Transfer is the term used to describe the generation of an image that contains the content from one image but the style from another or a collection of others. This problem has been solved in many ways throughout the previous years of research, with generative adversarial networks (GANs) emerging as an effective approach to solve this problem. Traditionally, GANs are able to provide reasonably good results by transferring a style learned from a collection of images to a single content image. However, GANs suffer from mode collapse, which results in unstable training and making style transfer quality hard to guarantee. To add to that, traditional GANs are able to learn one specific style from a collection of images. If one wishes to be able to convert from one content image to multiple different style collections, multiple models must be trained with all different weights, resulting in large training times and a large amount of parameters which may be unattractive when memory is an issue. Therefore, a model is proposed to be able to learn multiple different styles and increase the reliability of training results. The proposed model is named 'Gated-GAN' which is an adversarial gated network. The generative network has three different modules: an encoder, a gated transformer, and a decoder. To learn and output multiple different styles, input images can be passed through different branches of the gated transformer, with each branch designated for learning a different style. To stabilize training, the encoder and decoder are combined as an auto-encoder to reconstruct the input images. The discriminative network is used to distinguish between whether the output image was a generated image containing the new style or a genuine painted image. An auxiliary classifier is used to recognize the different styles of the transferred images, which in turn helps the generative network generate images in multiple styles. The results obtained from our project are not fully complete as there were issues implementing the training loop with the proper loss functions. The results obtained were trained as if it was a Cycle-GAN model, which is a popular model for style transfer using GANs. The results we obtained were evaluated using FID score. With further research and improved implementation, our group concluded the results in the original paper are well developed and the theory behind the implementation is solid.

Problem Statement

Style transfer is the process of redrawing or generating a new image that combines the content or scene from one image and the artistic style from a different image or a collection of images. Initially, the problem first stemmed from wanting to make an input image look as if it was drawn with the same style as in a painting from a famous artist. Manually transferring the image to a different style by interpreting another artist's style/technique is something that would take a professional artist considerable time and effort, if they are even able to achieve adequate results. To automate the process, AI and computer vision techniques have been used to obtain promising results. As there are many practical applications for style transfer, such as quickly creating animations from real life scenes, there has been a lot of research and development into creating

more accurate models that produce clear images and are pleasing to the eye. Unfortunately, many implementations of GANs that are able to obtain collective style transfer are only capable of generating images of one style. Therefore, multiple networks and models must be trained in order to achieve multiple models.

Since style transfer is often suitable for users to have fun with on a mobile app, there are two options that the app developers have. The first option is to have the user send an image to the cloud where the style transfer will be completed and the user will receive the transformed image back over a communication channel. Since in certain applications this may be undesirable since it requires a network connection, certain developers may wish to create an app where the model is saved locally and the generated image is computed on the mobile device. Since storage is often an issue on mobile devices, it would be counterproductive to store multiple models for style transfer that perform similar tasks. Therefore, there is a need to create a single model that is able to produce images from a selection of multiple styles. This will greatly reduce the number of parameters that need to be stored on the mobile device as there would be many shared parameters between the different styles. We can therefore say that a model needs to be created that shares the majority of parameters but effectively achieves to translate images into multiple different styles. Our proposed solution outlined below is able to achieve the two goals mentioned.

Proposed Solution

Recently, convolutional neural networks have been used to create a neural style transfer algorithm to separate image content from style to generate new images by combining the content of an arbitrary image with the style of a specific image. These approaches have been successfully developed and used, but they rely on the style from a singular image instead of a collection of images from a specific artist. For example, it may be attractive to learn the overall style of an artist instead of just learning the style of one painting from the artist. Therefore, generative adversarial networks (GANs) have been applied to solve the task.

GANs utilize a deep learning approach in which two neural networks compete with each other. The first neural network is called a generative network. The generative network, after being trained, will be able to generate an image with the content from an input image and the style learned from a collection of artwork during training. The discriminator is the second neural network in the framework. The generative and discriminative networks are constantly in an optimized two player game during training, where the discriminative network is responsible for determining/classifying if the input is painted by the artist or has the same style as the artist. The generative network learns to generate images to fool the discriminator into believing that the generated image is truly painted by the artist and not a fake image. However, it is well known that the traditional GAN training procedure is not particularly fit for this problem. Typically

GANs require paired training samples to help the model learn what an output from the generator should look like. In our case, since we don't know exactly what the output of the generator should look like since we don't have paintings of every real life scene from every artist, one option is to adapt to a cycle-consistent adversarial network. In particular, Cycle-GAN [1] addresses the unpaired image-to-image translation problem that we encounter in collective style transfer. The authors of Cycle-GAN simultaneously trained two pairs of generative networks and discriminative networks, one to produce imitative paintings and the other to transform the imitation back to the original photograph and pursue cycle consistency.

While Cycle-GAN was able to effectively solve the image-to-image translation problem and was able to collectively learn a style from a particular artist, each model could only transfer an image to one particular style. Therefore, in mobile applications where space is limited, Cycle-GAN is not practical to implement multiple different collections of styles. Therefore, there is a need for a singular model that can output multiple images with the same content, but in multiple different styles while reusing parameters and weights to save space. In addition, previous methods such as Cycle-GAN that adopt cycle-consistent loss require additional networks that convert the stylized image back into the original one. Since we wish to have a model that outputs multiple different styles of the same content image, we can see that the training algorithm will become more complicated if cycle-consistent loss is adopted. To move past this issue, an encoder-decoder subnetwork and an auto-encoder reconstruction loss is introduced to guarantee that the outputs will have the consistent semantic information with the content images. To achieve the one sided mapping that we wish to obtain, the auto-encoder reconstruction loss is used instead of cycle loss, therefore further reducing the number of parameters needed in the model.

The name of the proposed model is called Gated-GAN. The network architecture of the Gated-GAN model has the following components. One generative network that consists of an encoder, gated transformer, and decoder. The gated transformer has multiple gates that we select from based on the style that we are training/transferring to. If the gated transformer is skipped, the encoder and the decoder are trained as an auto-encoder to preserve semantic consistency between the input images and the images that the model generates. The other network that is built in the architecture is a discriminative network that is based on the well known Patch-GAN architecture. The purpose of the discriminative network is to distinguish between generated and real images, and identify the specific styles of the images that are in question.

As discussed previously, this model is different from Cycle-GAN [1] even though it contains almost the same architecture as Cycle-GAN. The largest differences between Cycle-GAN and the Gated-GAN models are the way in which they are trained. While both are adversarial models that use adversarial loss to train the generator to learn the style, Gated-GAN does not use cycle loss or identity loss. Instead, the generative network uses the combination of adversarial loss, total variation loss to smooth the generated image, and auxiliary classifier loss as the overall loss

function. In the Gated-GAN model, it is necessary to implement an auxiliary classifier with a categorical cross entropy loss function to help the model train differently for each style. Without the auxiliary classifier, the model confuses the styles together and the training process tends to merge the styles together.

Seen in Figure 1 is the overall network layout of the Gated-GAN model.

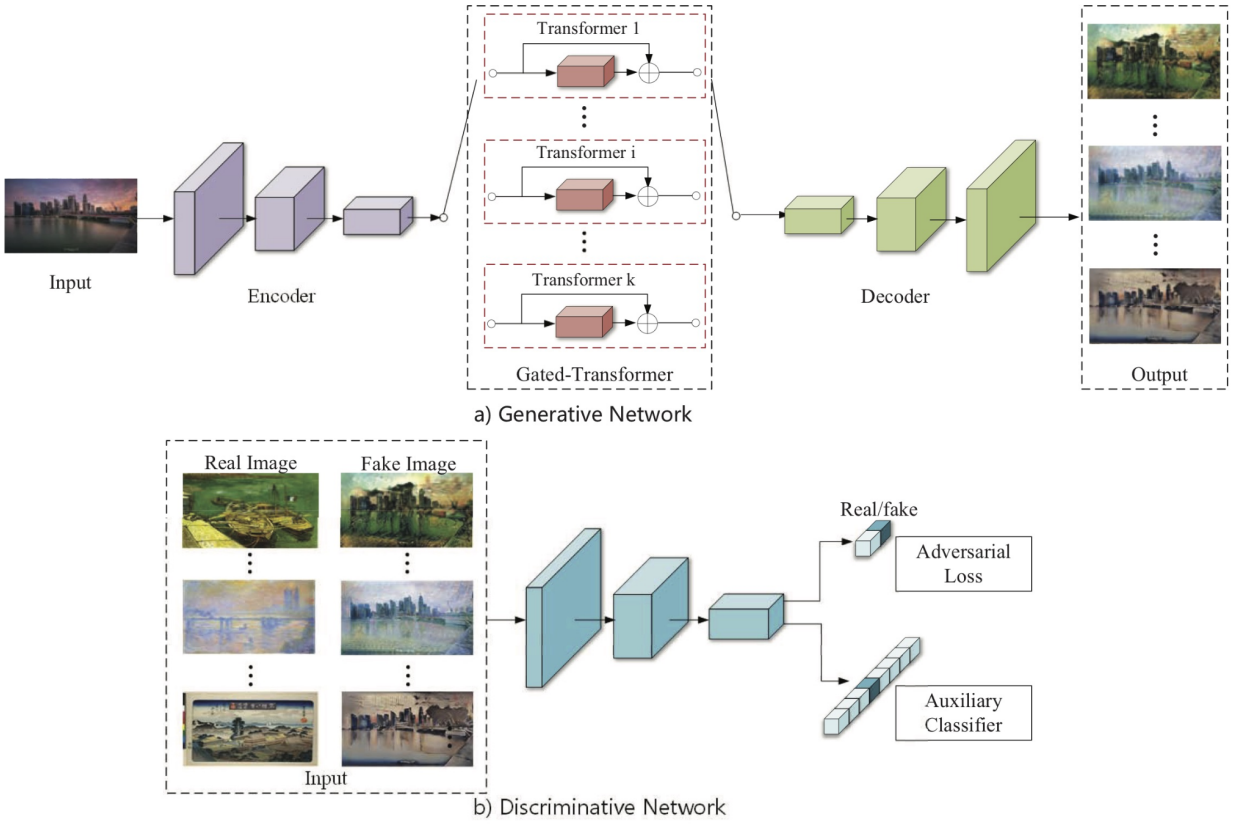

Figure 1: Architecture of the Gated-GAN Model consisting of a generative network and a discriminative network that uses adversarial loss and an auxiliary classifier to aid in training multiple styles [2]

Implementation Details

To begin, it's important to dive deeper into the network architecture previously discussed. The proposed architecture consists of a generative and a discriminative network. The generative network learns to generate style transfer images with the desired style and the discriminative network learns to distinguish between the generated images and the original painter's image. The generative network consists of three modules as previously mentioned: an encoder, a gated

transformer and a decoder. The encoder consists of three convolution layers where two layers are stride-2 convolutions. The gated transformer by default consists of four residual blocks - one for each of the four styles that will be trained on it. The decoder consists of five residual blocks and three convolution layers out of which two of them have stride=0.5. The discriminator consists of four convolution layers with stride=2 where the LeakyReLu activation function is used. Instance normalization is used after the convolution layers both in the generator and the discriminator.

After we were able to successfully implement the network architecture, we moved onto focusing on implementing the training loop with multiple different loss functions. Although the model architecture is similar to that described in Cycle-GAN, this Gated-GAN network is different from the Cycle-GAN as here we try to achieve multi-collection style transfer in a single network whereas the other models could only transfer one style per network. In Cycle-GAN, we convert the content image to the stylized image and then back to the original image and calculate the cycle-consistent loss. But, as style transfer is a one-sided process, it is not expected to transfer it back to the original with great accuracy. So, instead, in the proposed method, we use the auto-encoder reconstruction loss to make sure that the output images have consistent semantic information with the content images. Using this, the proposed algorithm can be generalized for multiple layers with less parameters. The proposed method makes use of 3 loss functions: adversarial loss, auxiliary classifier loss and total variation loss. To smooth the generated image, we make use of the total variation loss[2]. Below is the total generator loss function used for this model.

$$\mathcal{L}(G) = \mathcal{L}_{GAN} + \lambda_{CLS}\mathcal{L}_{CLS} + \lambda_{TV}\mathcal{L}_{TV}$$

Figure 2: Total generative loss function consisting of the following 3 loss functions from left to right: adversarial loss, auxiliary classifier loss, total variation loss [2].

It's important to note the specifics of the different components of the generative loss function. Below we can see the three sub loss functions written out.

$$\mathcal{L}_{GAN}(G, D) = \mathbb{E}_{y \in Y}\left[(D(y) - 1)^2\right] + \mathbb{E}_{x \in X}\left[D(G(x))^2\right].$$

Figure 3: Adversarial loss implemented as mean squared error loss [2].

$$\min_G \mathcal{L}_{CLS}(G) = -\mathbb{E}_{x \in X}\left\{\log C(Style = c|G(x, c))\right\}$$

Figure 4: Auxiliary classifier loss implemented as categorical cross entropy loss[2].

$$\mathcal{L}_{TV} = \sum_{i,j} \left[(G(x)_{i,j+1} - G(x)_{i,j})^2 + (G(x)_{i+1,j} - G(x)_{i,j})^2 \right]^{\frac{1}{2}}$$

Figure 5: Total variation loss used to smooth the generated image[2].

To calculate the loss between the reconstructed output and the input, we use the auto-encoder loss function as seen in Figure 6. Auto-encoder loss is introduced to stabilize the GAN training.

$$\mathcal{L}_R = \mathbb{E}_{x \in X} \left[||Dec(Enc(x)) - x||_1 \right].$$

Figure 6: Auto-encoder reconstruction loss implemented as L1 loss[2].

Once we had the model architecture laid out and the training loop with appropriate loss functions was good to go, we spent more time worrying about the data we were going to use to train the model. In the original paper that introduced the Gated-GAN, as well as other tutorials and papers that have focused on collective style transfer, a dataset is created with 4 different artists' work, paired with real life images to consist of style and content images respectively. Specifically, the dataset [7] includes a few thousand images of each artist's work as well as real life images to go along with them. The artists included were Vangogh, Cezanne, Monet, and Ukiyoe. Their artwork is similar in some ways, but collectively are different enough to produce different results for each style if the model is trained properly and thoroughly. When we loaded the dataset [7], we used the built in functions offered by tensorflow add ons to import the images into our program since the image data was very large to download locally and then upload into the program. In addition, Tensorflow provides users with great functions to create batch data for deep learning models. As seen in other tutorials, we decided to create Dataset objects of each of the different styles and content images. We were able to preprocess the data while doing so, which included data augmentation to improve the quality of our training results. We started with a (143, 143, 3) image and then randomly cropped, rotated, and flipped the image to create (128, 128, 3) images that would be input into the model. Since GANs are already known for taking very long to train, small dimensions were used to decrease the training time as much as possible while still getting accurate results.

After we had the data loaded and preprocessed, the model created, and the training loop ready to go, we were ready to train our model. At this point, we hit a lot of errors in our code in multiple different places. After debugging for as long as possible, we were able to get the data loaded and formatted correctly along with the model created and compiled. Unfortunately, we were not able to train the model on the training loop that we implemented with the proper loss functions and training technique described earlier in the paper and used in the original implementation of Gated-GAN. We tried implementing the Gated-GAN model in Python using Keras. The original implementation [4] of this model was created using the Lua programming language with Torch

as the deep learning framework. There was also code that we found in [5] that tried to implement the model in PyTorch, so to try and have our own success we decided it would be a good idea to try and implement the model in Keras. During our development, we debugged the training loop for as long as we could, but we decided to use the Cycle-GAN method to train our model to see what results we would be able to achieve by attempting to complete our goal using methods that were outlined as problematic in the Gated-GAN paper. We kept our architecture the same as if we were going to train using the methods and loss functions described earlier in the paper, but we simply trained our model on the Cycle-GAN training loop and referenced a tutorial on the Keras website that successfully implements style transfer using Cycle-GAN [3]. While we realize that training this way with multiple styles likely would result in poor results, we believed that it would be interesting to see if the gated transformer would be able to overwhelmingly control the multiple styles introduced in the model, or if it was an absolute necessity to not train with cycle-consistent loss, use auto-encoder reconstruction loss, and use an auxiliary classifier to supervise the training of different styles on the same model. Seen below is a diagram showing how Cycle-GAN trains the overall model using cycle consistent loss.

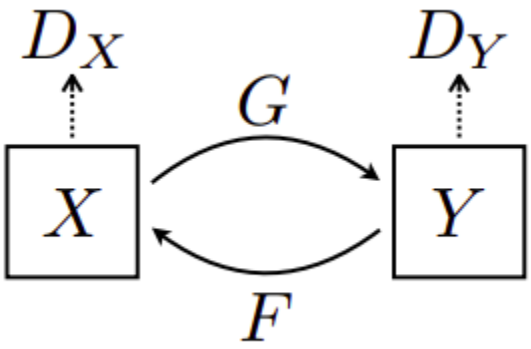

Figure 7: Cycle-GAN Training Process [1]

Here, the generator G learns to transform the real image to a stylized image and the generator F learns to transform the stylized image back to the original image. The discriminators Dx and Dy will differentiate between generated and original images to help the generator learn the proper style of the images being fed as input. Below are the loss functions implemented for CycleGAN:-

$$\mathcal{L}_{\text{cyc}}(G, F) = \mathbb{E}_{x \sim p_{\text{data}}(x)}[\|F(G(x)) - x\|_1] + \mathbb{E}_{y \sim p_{\text{data}}(y)}[\|G(F(y)) - y\|_1].$$

Figure 8: Cycle consistency loss for CycleGAN

Overall, our final implementation includes two different Python notebooks. One was developed to be used strictly with the Gated-GAN training methodology that failed to train. If the errors

were fixed in the training loop, the evaluation process could easily be adapted to that notebook to achieve similar evaluation to that used in the second notebook. In the second notebook, we trained the model using the Cycle-GAN training loop to achieve some generated output. We were able to train the model for 10 epochs, as each epoch took about 12 minutes to complete on Google Colab's GPUs. Since each user only has limited GPU time, this is the most we were able to train our model. The results and evaluation of the output from this second notebook is mentioned in the following section, as well as a discussion on the overall process of implementing this project.

How to run/use the program

Our code was written in Python notebook files, so users could read and understand the code as easily as possible. In addition, it is highly recommended to use a GPU to train the model, as the training time decreases significantly. To take advantage of free GPU resources, we ran our code on google colab for the duration of development. To run code on google colab, one has to upload the notebook, turn on GPU acceleration in the notebook settings, and then run all the cells in the notebook from start to finish. The only thing that must be done before running the training loop is to create folders for the stylized images to be saved to. For example, since we used 4 different styles, we created folders named 'style1', 'style2', 'style3', and 'style4'. After each epoch our code generated and saved 50 images from each artist into these folders. Users would be able to download those images from colab if they wish. We also provide a cell that saves the weights in the model. The weights get saved to a folder named 'final_weights'. To use these saved weights, one can follow the tutorial provided in this reference [10] using the tensorflow built in functions. The saved weights are provided in the google drive folder linked here:
https://drive.google.com/drive/folders/1qxkvli4itBBboiATBk7ENvOwxFA_E_Gs?usp=sharing

In the project submission folder, two different Python notebooks were provided as submissions. The one titled 'Multi_style_transfer_with_Cycle_GAN' is the notebook that trains the model with the Cycle-GAN training loop. This is also the notebook that has all of our results of our experiments. The other notebook provided is titled 'GatedGAN' and contains the code for our attempt at implementing the GatedGAN training loop that contains the auxiliary classifier, auto-encoder reconstruction loss, and total variation loss.

Results and Discussion

To evaluate the results gathered from the experiment, it's important to have a quantitative evaluation metric. Since we are generating images with a neural network, we want to ensure that they are smooth images that humans would believe are real images, or at least close enough to

real images that they care to look at them. Since we are not sure what the actual output of the image should be given that we don't have painted pictures of every scene that we wish to use our model on for style transfer, we must evaluate the generated images based on the quality of output and how similar the generated distribution is to the real distribution of paintings of the same style. Therefore, we compute the Frechet Inception Distance (FID) score between our generated images and the testing sets for each style that we trained our model on. The FID score estimates the quality of a collection of synthetic images based on how well the top-performing image classification model Inception V3 classifies them as one of 1,000 known objects. The formula for calculating FID score can be seen below.

$$FID = \|\mu_r - \mu_g\|^2 + T_r(\Sigma_r + \Sigma_g - 2(\Sigma_r \Sigma_g)^{1/2})$$

Figure 8: FID Formula

In general, the steps used to calculate the FID score are outlined below:

1. Use the Inception V3 pre-trained model to extract the feature vectors of real images and fake images generated by the generator
2. Calculate the feature-wise mean of the feature vectors generated in step 1
3. Generate the covariance matrices of the feature vectors - Σr and Σg
4. Calculate trace (The sum of the elements along the main diagonal of the square matrix) of the covariance of the real image feature vectors plus the covariance of the generated feature vectors minus two times the square root of the covariances multiplied together.
5. Calculate the squared difference of the mean vectors calculated in step 2 - ||m-m_w||^2.
6. Finally, add the output of step 4 and step 5

To generate the FID scores, a few simple utility functions were created to get the image data into the proper format to calculate the scores.

Since we were not able to successfully implement the training loop that was introduced in the original Gated-GAN paper, we expected that our results would not be great since we have previously talked about the different flaws of training multiple styles without an auxiliary classifier. In addition, the training time for our model was relatively large so we were not able to train the model for as many epochs as mentioned in the original implementation. Instead, we trained the model for 10 epochs using the Cycle-GAN training loop. Seen below are a few different screenshots of generated images produced from the model after different epochs.

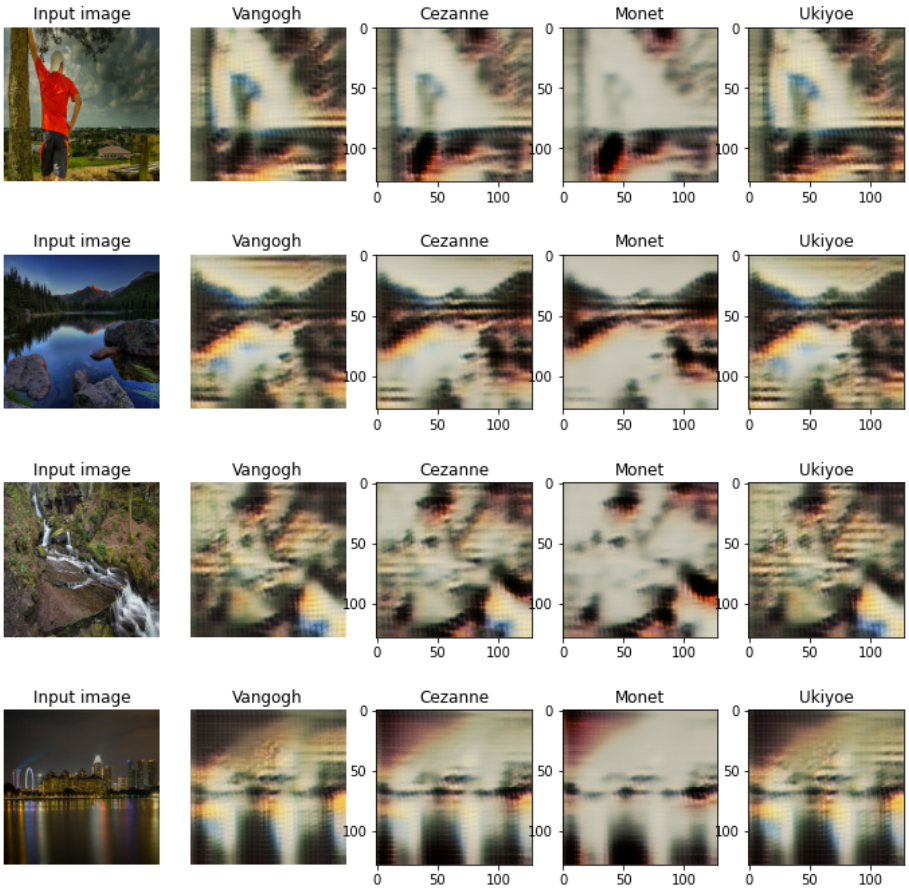

Figure 9: Generated images after training for one epoch

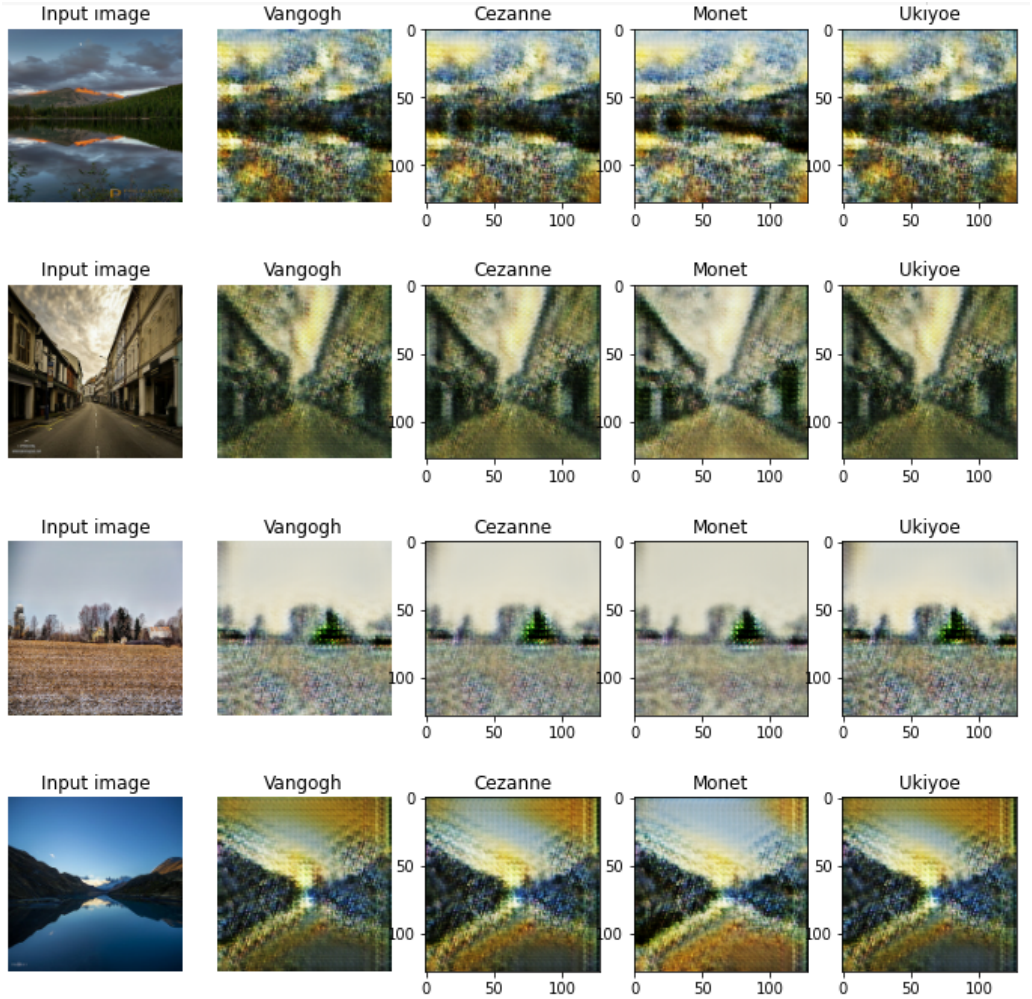

Figure 10: Generated images after training for three epochs

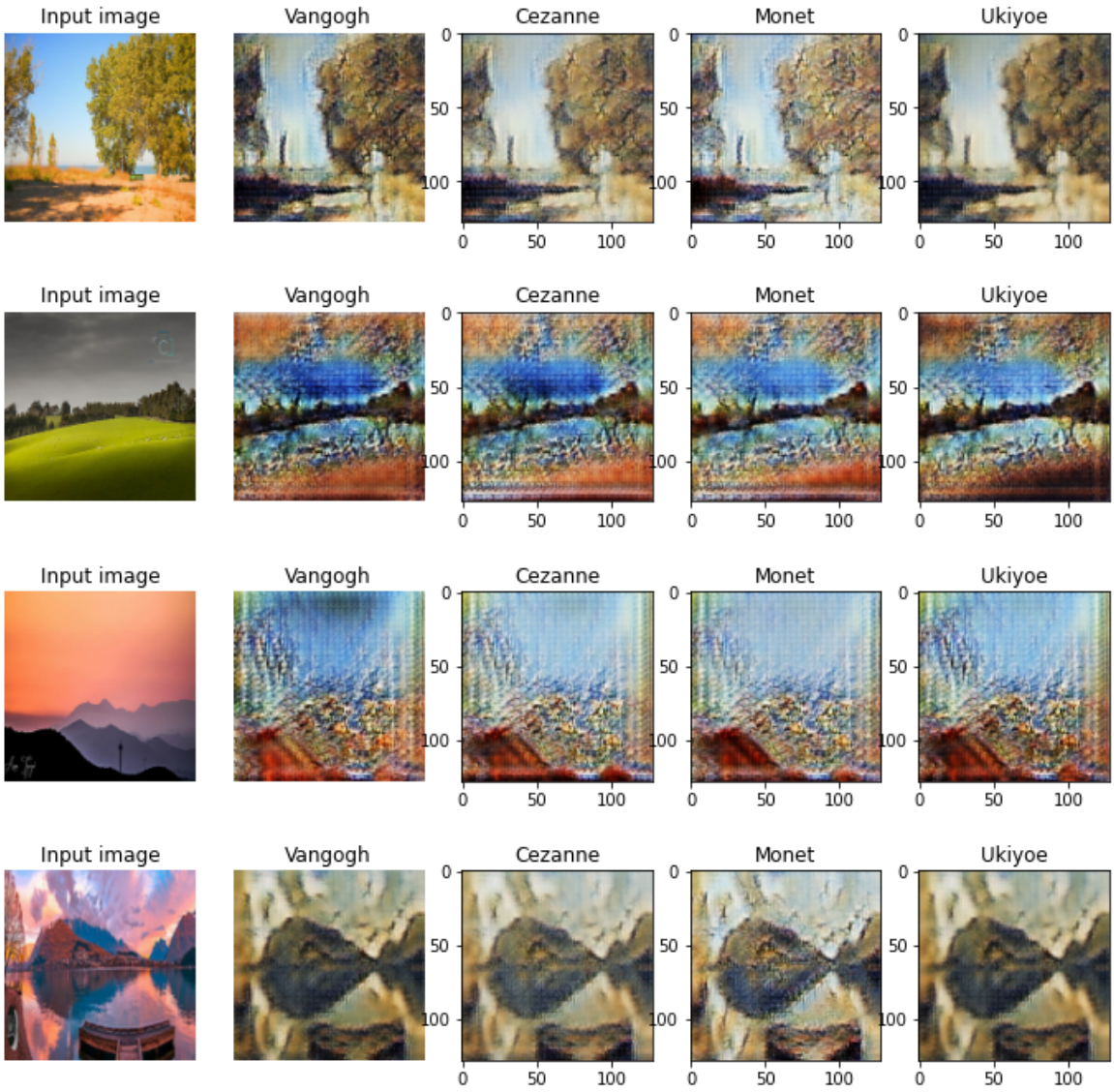

Figure 11: Generated images after training for eight epochs

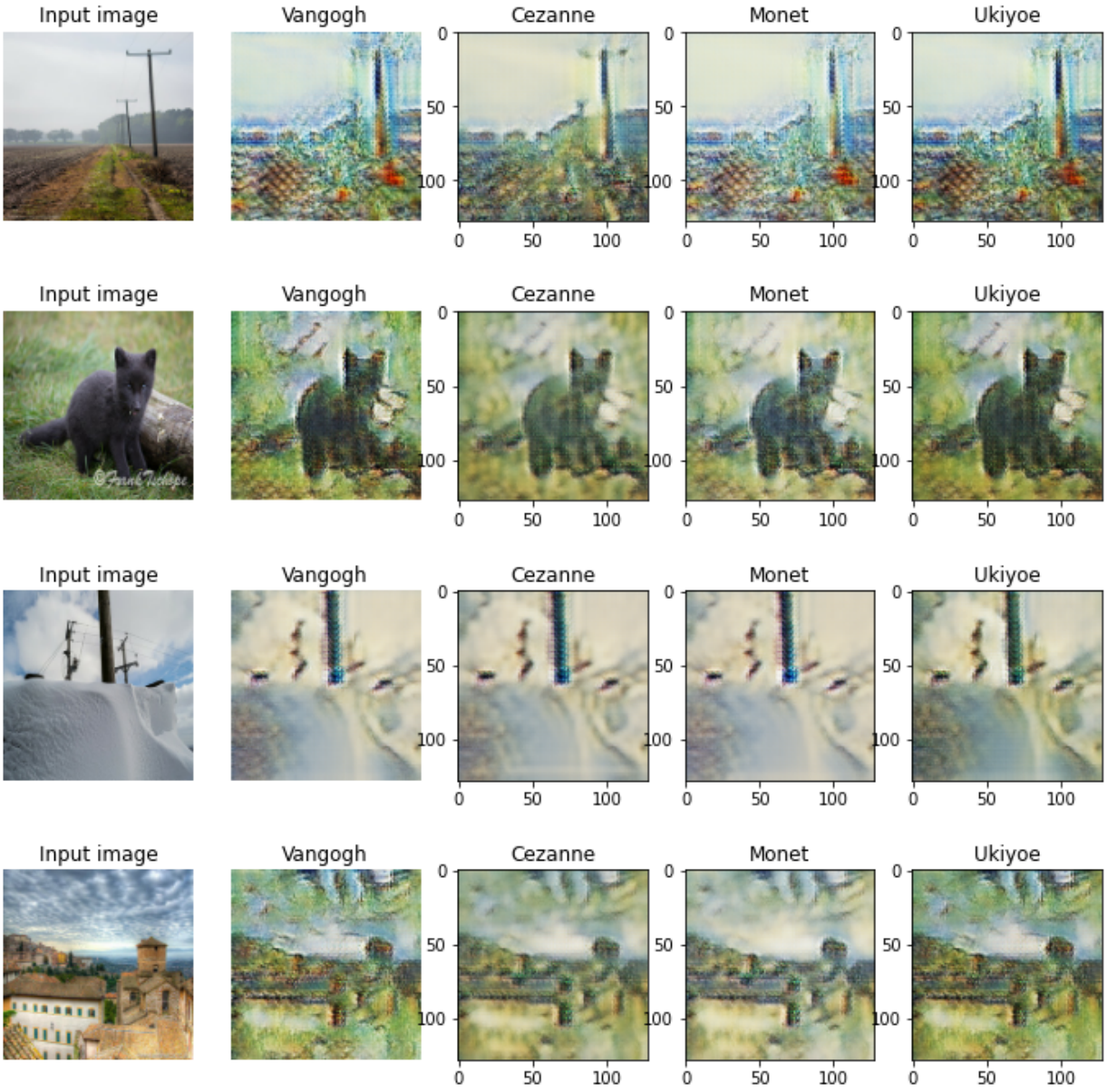

Figure 12: Generated images after training for ten epochs

As we can see in the generated images, we are not able to obtain extremely realistic generated images. There are a few images that were generated that look better than others, such as the first row of images generated after the eighth epoch of training. Specifically, we can see that the Monet painting has more texture than the other images as well as the Ukiyoe painting looking very smooth. On the other hand, the first row of output of generated images after the 10th epoch of training is not very good at all. It's very clear to see what the content in the image is and it seems like a lot of pixels on a screen randomly thrown together. To quantify the results, we can see the output of our FID scores below.

	Vangogh	Cezanne	Monet	Ukiyoe
FID:	2370.253	3258.643	2272.889	2664.582

Figure 13: FID scores of each of the outputted styles

In the original implementation of the Gated-GAN architecture, the authors compared their implementation with Cycle-GAN. We can see the table below.

TABLE II
QUANTITATIVE EVALUATION ON COLLECTION STYLE TRANSFER IN TERMS OF FID TO MEASURE THE PERFORMANCE. LOWER SCORE INDICATES BETTER QUALITY.

Style	Content Images	CycleGAN [14]	Ours
Monet	86.50	64.14	55.13
Cezanne	186.73	106.96	107.27
Van Gogh	173.01	107.03	109.59
Ukiyoe	195.25	103.36	115.96
MEAN	160.37	95.37	96.99

Figure 14: Cycle-GAN and Gated-GAN FID scores

In comparison to the results obtained from the original papers, we can see that our results are very poor. Below we can see how clear the generated images are in the Gated-GAN paper.

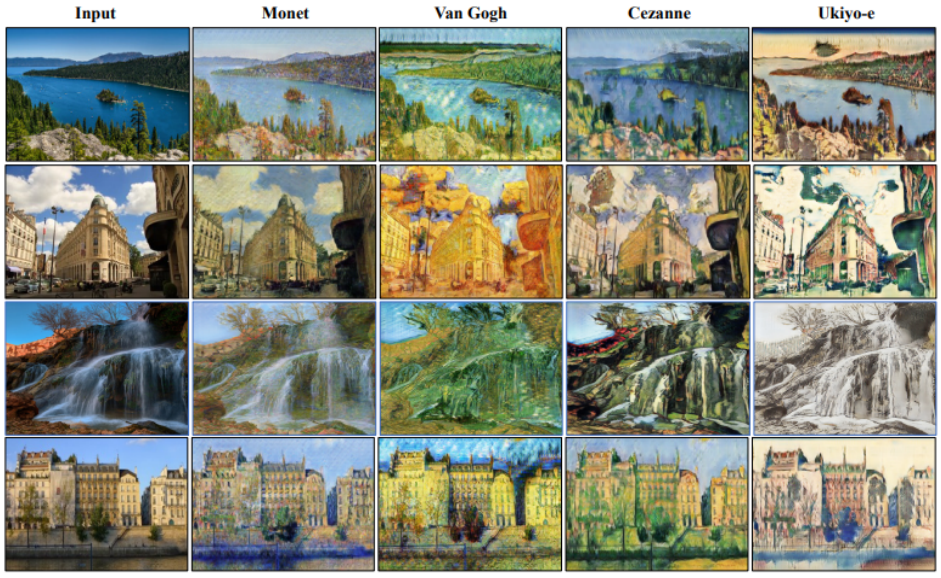

Figure 15: Generated images from the initial implementation

There are many possible reasons that our results are not as good as those achieved by the initial implementation of Cycle-GAN and Gated-GAN. To begin, we only trained our model for 10 epochs. We can see that after the first epoch the generated images were very poor and by the 10th epoch, viewers were able to make out the content in each image. During training, we also did not see any sign of the model loss leveling off. This indicates that with more training, we would decrease our loss significantly and improve our results.

Another possible reason why our FID scores and images are not on par with the initial implementations is that we tried to generate images of different styles using the same network. Since we did not successfully implement an auxiliary classifier, our model probably had a hard time distinguishing between the different styles. This is evident in our images as we can see that the generated images are all very similar, with only minor differences. If we trained the model with only one style and more epochs, it is very likely that we would obtain similar results as in the initial implementation of Cycle-GAN.

Conclusion

As discussed above, using the proposed method, we can implement multi-collection style transfer in a single network by using a gated-transformer that will use different branches to train images on different styles. We tried to implement Gated-GAN for multi-style transfer but as we were not successful in implementing it, we created a Cycle-GAN model for multi-style transfer. Based on our results, we can say that Cycle-GANs are better suited for only one style transfer per network and won't perform well with multi-style transfer and Gated-GANs are much suitable for multi-style transfer purpose, which can also be seen when we compare the FID scores of CycleGAN and Gated-GAN. On the basis of this, we can say that the Auxiliary classifier loss and Total Variation loss play a very important role in training the model to generate different stylized images simultaneously using a single network model. Also, using the Auto-encoder loss, the GAN network can be stabilized and so, the gated-GAN is stable and an effective approach as demonstrated in [2]. All in all, our group was able to successfully implement the Gated-GAN network architecture in Keras while training it using the Cycle-GAN training loop. We were able to understand the inner workings of GANs, as well as different variants of them along the way.

Future Scope

In future, we will try to complete the implementation of our Gated-GAN model for multi-style transfer and resolve the issues. Also, the Gated-GAN model can also be used to achieve style interpolation, i.e. to create a new style from styles of different artists or genres[2]. This model can also be used for image generation tasks like season transfer, photo enhancement, etc. and to generate diversified style transfer results.

Bibliography

[1] J. Zhu, T. Park, P. Isola A. A. Efros, "Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks" in *The IEEE International Conference on Computer Vision (ICCV)*, Oct 2017

[2] X. Chen, C. Xu, X. Yang, L. Song, D. Tao, "Gated-GAN: Adversarial Gated Networks for Multi-Collection Style Transfer" in the *IEEE Transactions on Image Processing,* Feb 2019.

[3] Keras (2020, Aug 12). *CycleGAN* [Online]. Available: https://keras.io/examples/generative/cyclegan/

[4] Github (2019, Apr 7). *Gated-GAN: Adversarial Gated Networks for Multi-Collection Style Transfer* [Online]. Available: https://github.com/xinyuanc91/Gated-GAN

[5] Github (2019, Jun 3). *Gated-Gan for Multi-Style Transfer* [Online]. Available: https://github.com/colemiller94/gatedgan

[6] Jason BrownLee (2019, Aug 30). *How to Implement the Frechet Inception Distance (FID) for Evaluating GANs* [Online]. Available: https://machinelearningmastery.com/how-to-implement-the-frechet-inception-distance-fid-from-scratch/

[7] TensorFlow. Cycle-GAN [Online]. Available: https://www.tensorflow.org/datasets/catalog/cycle_gan (Dataset)

[8] Tensorflow [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/all_symbols (Library Documentation)

[9] Keras [Online]. Available: https://keras.io/api/ (API documentation)

[10] Save and load Keras models [Online]: https://www.tensorflow.org/guide/keras/save_and_serialize (explanation with code)