[Team 22] Collection Style Transfer

Vaibhav Choudhary (vchoudh2), Mahmoud Abdelkhalek (maabdelk), Sharvil Vijagish Mainkar (smainka)

***Abstract*—Our goal in this project was to apply some of the concepts that we learned in ECE 542, including Convolutional Neural Networks (CNN) and Generative Adversarial Networks (GAN). We used a modified GAN called CycleGAN to perform collection style transfer, where an image is re-drawn in the style of a particular artist based on a collection of input images. We also used a CNN to quantitatively measure the style of images generated by CycleGAN and to compare them to a baseline image. We successfully show that the CNN is able to differentiate between images from a specific artist and random images, and hence it is also able to objectively measure the style of images generated by CycleGAN. We also show that our CycleGAN is able to perform better than a pre-trained CycleGAN made available by its original creators on certain datasets.**

I. Model Training and Selection

In [1], the authors present a network architecture called CycleGAN that is able to perform collection style transfer, where an image is transformed from a domain A to a different domain B and vice versa. Images are in the same *domain* if they share a general style. For example, several paintings from Vincent van Gogh exist in the same domain, while paintings from Vincent van Gogh and Claude Monet exist in different domains.

Unlike neural style transfer [2], where the content of one image and the style of another image are combined to synthesize a new image, CycleGAN learns the mapping between two image collections, such that an image synthesized by CycleGAN is able to capture the style of an entire collection of images. Using the PyTorch Python package [3], CycleGAN was re-created and trained on multiple datasets to perform collection style transfer. Next, the VGG CNN shown in [2] was used to measure the style losses of original paintings, images transformed by CycleGAN, and the original images before their transformation to determine if the images transformed by CycleGAN inherited the style of the target domain.

A. The Model

A GAN involves a battle between two players: a generator and a discriminator. The generator attempts to convert random noise into images that are similar to the images in the original dataset, and the discriminator attempts to determine whether an image comes from the original dataset or from the generator, as shown in figure 1.

In practice, both the generator and the discriminator are implemented using CNNs. The generator is similar to a decoder in an autoencoder, as it consists of de-convolutions to obtain an output image from a random input vector. Conversely, the discriminator consists of convolutions and max-pooling to perform classification. However, in order to perform collection style transfer, this GAN architecture needs to be modified.

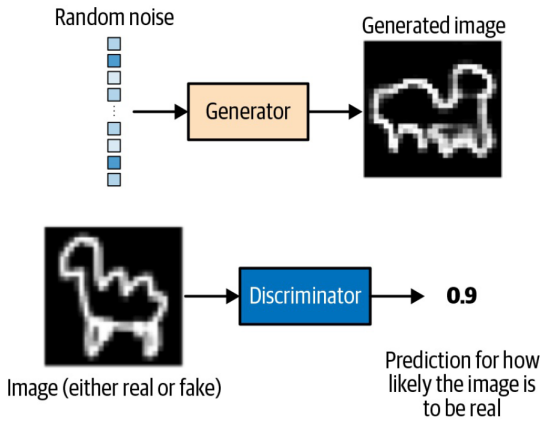

Fig. 1. The generator and discriminator in a GAN.

More precisely, an ”encoding” and ”transformation” stage are added to the generator, as shown in figure 2.

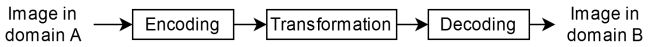

Fig. 2. Overview of the CycleGAN generator.

An input image from domain A is first *encoded* into a high-level representation with a vector in a latent space. Next, this vector is mapped to a different latent space corresponding to the images in domain B during the *transformation* stage. Finally, the transformed vector is decoded as in the original GAN architecture to obtain a generated image in domain B. The full modified generator architecture is shown in figure 3.

The corresponding DS_Block, T_Block, and US_Block parts of the generator are shown in figures 4, 5, and 6 respectively. The Instance Normalization layer replaces the typical Batch Normalization layer for both CycleGAN and the Fast Style Transfer (FST) network discussed in section I-B. As discussed in [4], this leads to qualitative improvements in the generated images.

In addition to modifying the generator, the discriminator was also modified. As discussed in [5], using a discriminator that is able to classify patches of the input image rather than individual pixels allows it to distinguish style rather than content. More specifically, given an $N \times M$ input image, the discriminator will output a $P \times Q$ image ($P < N$,$Q < M$), where every pixel classifies whether every 70×70 patch in the input image is real or fake. The full discriminator architecture is shown in figure 7. The CycleGAN discriminator uses a similar DS_Block as in the generator, with the exception that a Leaky ReLU activation function is used instead.

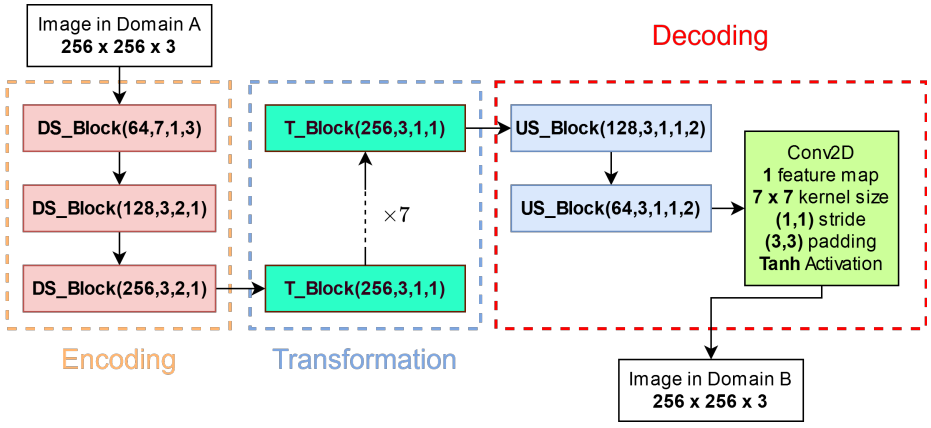

Fig. 3. Details of the CycleGAN generator.

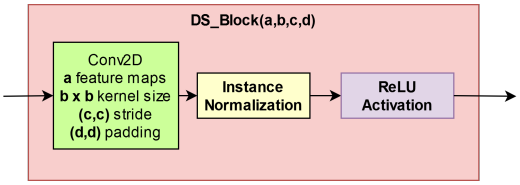

Fig. 4. The DS_Block used in the CycleGAN generator and discriminator. A Leaky ReLU activation function with $\alpha = 0.2$ was used for the discriminator.

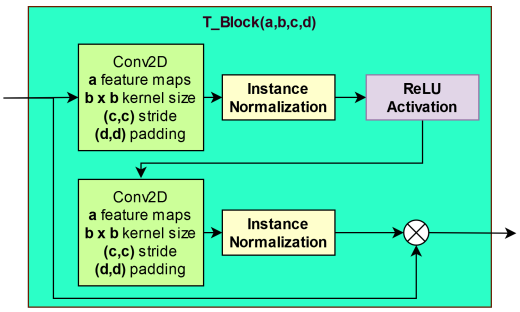

Fig. 5. The T_Block used in the CycleGAN generator.

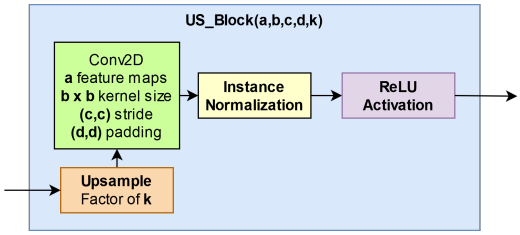

Fig. 6. The US_Block used in the CycleGAN generator.

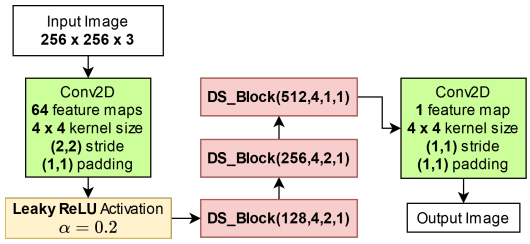

Fig. 7. The CycleGAN discriminator architecture.

$$\epsilon(l,k,j) = (Y_j - D_k(G_{lk}(X_l)))^2$$
$$MSE(l,k,j) = \frac{1}{N \cdot C \cdot H \cdot W} \cdot \sum_{N,C,H,W} \epsilon(l,k,j) \quad (1)$$

CycleGAN consists of 2 generators and 2 discriminators. Each generator-discriminator pair is used to transform an image from one domain to the other and to classify whether an image is real or fake. Let $G_{lk}(\cdot)$ be the output of the generator that transforms an image from domain l to domain k, $D_k(\cdot)$ be the output of the discriminator that classifies images in domain k, X_l be an image from domain l, and Y_j be an image of the same size as $D_k(\cdot)$ filled with j's. Since the generator should be able to fool the discriminator, then for a given batch of images, if N is the batch size, C is the number of channels, H is the image height, and W is the image width, then both generators and discriminators are trained using a mean squared error loss function, as shown in equation 1.

For example, the generator that transforms an image from domain A to domain B is trained with the loss function $MSE(A, B, 1)$, while the discriminator that classifies images in domain A is trained with the loss function $MSE(B, A, 0)$ since it has to be able to distinguish between real images and fake images created by the generator.

In addition to the mean squared error loss, both generators are trained using two different mean absolute error losses: a cycle loss (CL) and an identity loss (IL). The cycle loss ensures that image content is not lost when transforming from one domain to another. This is similar to translating an English sentence to French and back again. The resulting sentence should be the same as the original sentence. The identity loss

TABLE I
HYPER-PARAMETER TUNING RESULTS AFTER 50 EPOCHS OF TRAINING (BASELINE 1)

Painting	Number of T_Blocks	Optimizer	Learning Rate	Learning Rate Scheduler	Painting Style Loss	CycleGAN Style Loss	Original Image Style Loss
Monet	3	SGD	0.0008	Scheduler A	73.38	75.74	77.93
	9	Adam	0.0002	Scheduler B	73.38	75.17	77.93
van Gogh	3	SGD	0.0008	Scheduler A	18.74	228.07	174.27
	9	Adam	0.0002	Scheduler B	18.74	197.30	174.27
Ukiyo-e	3	SGD	0.0008	Scheduler A	3.38	4.25	5.10
	9	Adam	0.0002	Scheduler B	3.38	4.66	5.10
Cezanne	3	SGD	0.0008	Scheduler A	1.06	1.80	2.06
	9	Adam	0.0002	Scheduler B	1.06	1.91	2.06

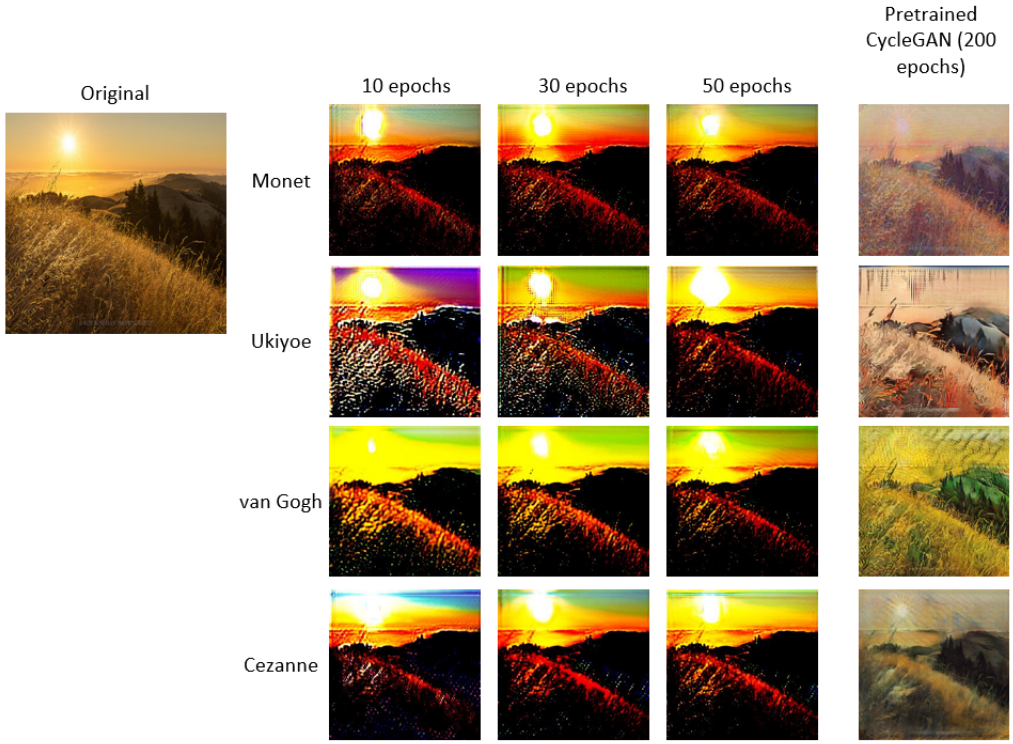

Fig. 8. CycleGAN results after 50 epochs.

preserves certain color properties in the original image during the transformation to ensure that important context is not lost. The cycle and identity losses are shown in equations 2 and 3 respectively.

$$\begin{aligned} \epsilon(l,k) &= \left\| X_l - G_{kl}(G_{lk}(X_l)) \right\|_1 \\ CL(l,k) &= \frac{1}{N \cdot C \cdot H \cdot W} \cdot \sum_{N,C,H,W} \epsilon(l,k) \end{aligned} \tag{2}$$

$$\begin{aligned} \epsilon(l,k) &= \left\| X_l - G_{kl}(X_l) \right\|_1 \\ IL(l,k) &= \frac{1}{N \cdot C \cdot H \cdot W} \cdot \sum_{N,C,H,W} \epsilon(l,k) \end{aligned} \tag{3}$$

For example, the generator that transforms an image from domain A to domain B was trained using the $CL(A,B)$ and $IL(A,B)$ losses.

B. Baseline

The traditional implementation of neural style transfer uses a CNN to distinguish an image's style from its content, which in turn allows us to generate new images that replicate the content and style of different images. In [2], style representations are extracted separately from content representations using traditional CNNs. The CNNs hierarchically stack to form style-content representations, it is possible to form style representations by capturing the correlation between the feature maps of the convolution layers.

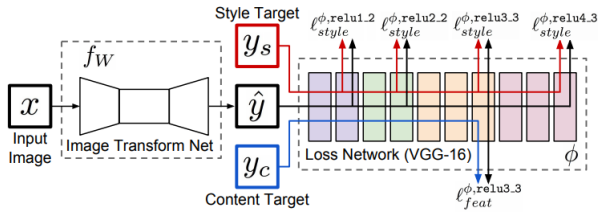

Fig. 9. Fast Style Transfer with VGG-16 as backbone.

The major disadvantage of traditional style transfer is that it is computationally expensive since each step of the optimization problem requires a forward and backward pass through the pre-trained network. To overcome this burden, we add another feed forward network to approximate the solution.

The Fast Style Transfer solution proposed in [6] serves as our baseline as the transformer network proposed is able to learn the style representation of the artist images hundreds of times faster than the architecture proposed in [2].

Our baseline model consists of a transformer net as shown in [4] with a few changes in the up-sampling block. Instead of using fractional-strided convolutions, we use nearest neighbour interpolation along with instance normalization. The baseline model is shown in figure 9. The pre-trained VGG-16 is used as our backbone network for extracting the content and style representation of the images. Instead of using a random noise image for calculating the style and content loss we use the image out from our transformer net as it leads to faster convergence. The transformer network is trained while the backbone is freezed. During the inference time the transformer net is used to generate the style image. We trained our baseline using COCO 2017 dataset [7]. The dataset contains 123,287 images of 91 different object categories. We use COCO as our content dataset we choose 4 different artists images to develop models.

The style loss is computed in the same way as in [2]. For example, to calculate instead of computing the style loss using a random noise image we use a transformer net generated content image to calculate the style loss. We take up a convolution layer of our choice and generate the feature maps. Then we take the feature maps for the style image and the same is generated for the content image and calculate the correlations for each individual images. The correlation matrix is called the gram matrix. We take the MSE between the gram matrices of the style image and the content images. This is our style loss.

II. Experimental Section

A. Metrics

Our primary goal while defining the metric was to quantitatively measure how well the style transferred from one domain to another. We decided to use the style loss from the Fast Style Transfer Network as the quantitative measure for the same. For images that had a better style transfer, the value of the style loss function would be lower. To measure this, we conducted two different baseline experiments to prove the efficacy of our CycleGAN algorithm that we had written.

The first baseline shown in figure 10 was used as a sanity check to evaluate the efficacy of the baseline itself. In this method, we measured the style loss output of an image generated by our CycleGAN, trained for 50 epochs and benchmarked this against a real painting of the artist whose style we had transferred. We also measured the loss given by a image with a random style for control. Intuitively, we expected the loss functions to have the highest and lowest values for the random image and an image of a real painting, with the loss of the generated by our CycleGAN to lie somewhere in the middle. We performed this experiment for the Ukiyoe and Monet Painting Data sets.

For the second baseline shown in figure 10, we measured the style loss values for the our CycleGAN algorithim as compared to a pre-Trained CycleGAN algorithm and the output of the Fast Style Network. The aim was to show that the CycleGAN network transferred the style better than that of the Fast Style Network. This meant that the CycleGAN was able to find a lower value of the Style Loss than the FST Network even when not explicitly trained to do so.

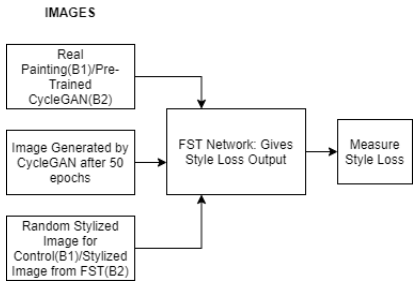

Fig. 10. The images used for Baseline 1 (B1) and Baseline 2 (B2) computations

B. Model Selection

The hyper-parameters to be optimized are the number of T_Blocks in the generator, the gradient descent optimizer to be used, the learning rate, and the learning rate scheduler. The performance metric is the absolute difference between the style loss (SL) of an original painting and the style loss of an image generated by CycleGAN. Results are shown in table I. Learning rate scheduler A decays the learning rate by a factor of 0.7 every 10 epochs, while learning rate scheduler B does not decay the learning rate at all. The CycleGAN model was able to learn some sense of style for the Monet, Ukiyo-e, and Cezanne paintings, but it clearly did not for the van Gogh paintings, as its style loss was higher than that of the original image before it was transformed. Additionally, the combination of the Adam optimizer, 9 T_Blocks, a learning rate of 0.0002, and scheduler B lead to the best results on average. Hence, this combination was chosen for further experimentation.

TABLE II
BASELINE 2 COMPARISON

Style Loss (10^{-5})	Pre-trained CycleGAN	CycleGAN (50 epochs)	FST
Ukiyo-e Dataset	3.7	3.9	4.5
Monet Dataset	73.2	74.9	73.7
Winter to Summer	70.9	50.7	-

C. Performance and Comparison to Baseline

The authors of the original CycleGAN paper were able to achieve satisfactory results after 200 epochs of training. However, because CycleGAN consists of 4 networks and due to a lack of computational resources, we were only able to train CycleGAN once for 200 epochs on a single dataset and for 50 epochs on other datasets. We trained CycleGAN on paintings from Claude Monet, Vincent van Gogh, Paul Cezanne, and the Japanese art genre Ukiyo-e [8] for 50 epochs, as shown in figure 8. Two different comparisons to baselines are given, one in table I and the other in table II. We observe that our CycleGAN model is able to produce better results than baseline 1 and is successfully able to produce a smaller style loss than the original image before it is transformed. We used the model in the previous section to benchmark in baseline 2 as shown in table II and achieved comparable results to the pre-trained CycleGAN and FST. We were not able to train to 200 epochs, due to limited computational resources. However, we were able to train our network for 200 epochs for one dataset and did achieve better loss results on the Winter to Summer Dataset, which is also visually intuitive in 11.

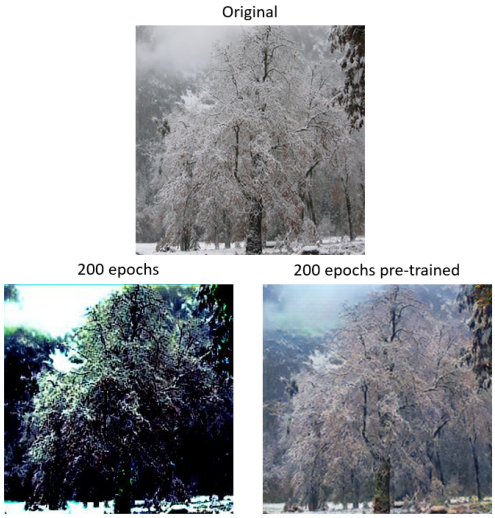

Fig. 11. CycleGAN results after 200 epochs.

REFERENCES

[1] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros, "Unpaired image-to-image translation using cycle-consistent adversarial networks," Nov. 15, 2018. arXiv: 1703.10593. [Online]. Available: http://arxiv.org/abs/1703.10593 (visited on 04/28/2020).

[2] L. A. Gatys, A. S. Ecker, and M. Bethge, "A neural algorithm of artistic style," *arXiv:1508.06576 [cs, q-bio]*, Sep. 2, 2015. arXiv: 1508.06576. [Online]. Available: http://arxiv.org/abs/1508.06576 (visited on 04/29/2020).

[3] (). PyTorch. Library Catalog: pytorch.org, [Online]. Available: https://www.pytorch.org (visited on 04/28/2020).

[4] D. Ulyanov, A. Vedaldi, and V. Lempitsky, "Instance normalization: The missing ingredient for fast stylization," *arXiv:1607.08022 [cs]*, Nov. 6, 2017. arXiv: 1607.08022. [Online]. Available: http://arxiv.org/abs/1607.08022 (visited on 04/30/2020).

[5] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros, "Image-to-image translation with conditional adversarial networks," *arXiv:1611.07004 [cs]*, Nov. 26, 2018. arXiv: 1611.07004. [Online]. Available: http://arxiv.org/abs/1611.07004 (visited on 04/30/2020).

[6] J. Johnson, A. Alahi, and F. Li, "Perceptual losses for real-time style transfer and super-resolution," *CoRR*, vol. abs/1603.08155, 2016. arXiv: 1603.08155. [Online]. Available: http://arxiv.org/abs/1603.08155.

[7] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," *Lecture Notes in Computer Science*, pp. 740–755, 2014, ISSN: 1611-3349. DOI: 10.1007/978-3-319-10602-1_48. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-10602-1_48.

[8] (). Index of /~taesung_park/CycleGAN/datasets, [Online]. Available: https://people.eecs.berkeley.edu/~taesung_park/CycleGAN/datasets/ (visited on 05/01/2020).