Style Transfer with CycleGAN

Sélim Ben Abdallah

Abstract

This study aims to explain what CycleGAN [5] is and how it can be used for style transfer. In the first section, I introduce the architecture of CycleGAN. In the second section, I present my implementation of CycleGAN, training it to transfer style from real landscape photographs to Monet-style paintings, and vice versa.

First Note: In a previous article, I introduced Generative Adversarial Networks (GANs) and trained both a GAN [1] and a DCGAN [4] on the MNIST dataset [3] to generate handwritten digits.

Second Note: I wrote this article myself and used ChatGPT-4o to help correct sentences and improve the clarity of certain sections.

1 A brief introduction to CycleGAN

CycleGAN [5] is presented as an approach for learning to translate an image from a source domain X to a target domain Y in the absence of paired examples. The goal is to learn a mapping $G : X \rightarrow Y$ such that the distribution of images from $G(x)$ is indistinguishable from the distribution Y using an adversarial loss. Because this mapping is highly under-constrained, it is coupled with an inverse mapping $F : Y \rightarrow X$, and a cycle consitency loss is used to enforce $F(G(X)) \simeq X$.

1.1 Main components

We have two generators : G translates images from domain X to Y ; F translates images from domain Y to X.

We have two discriminators : D_Y discriminates between real images in domain Y and fake images $G(X)$. D_X discriminates between real images in domain X and fake images $F(Y)$.

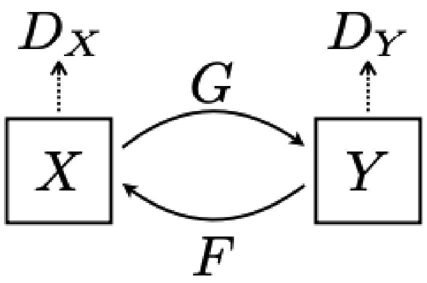


Figure 1: Mapping functions and adversarial discriminators (CycleGAN paper [5])

1.2 Loss functions

The model combines two types of losses. For both directions, an Adversarial Loss $\mathcal{L}_{GAN}$ encourages generated images to be indistinguishable from real images in the target domain.

For the mapping function $G : X \rightarrow Y$ and its discriminator D_Y, $\mathcal{L}_{GAN}$ is expressed as :

$$\mathcal{L}_{GAN}(G, D_Y, X, Y) = E_{y \sim p_{\text{data}}(y)}[\log D_Y(y)] + E_{x \sim p_{\text{data}}(x)}[\log(1 - D_Y(G(x)))] \tag{1}$$

where G learns to generate images $G(X)$ that look similar to images from domain Y, while D_Y aims to distinguish between translated samples $G(X)$ and real samples y. As said in [5], G aims to minimize this objective against an adversary D that tries to maximize it, i.e. $min_G \; max_{D_Y} \; \mathcal{L}_{GAN}(G, D_Y, X, Y)$.

Similarly, for the mapping function $F : Y \rightarrow X$, and its discriminator D_X, such that $min_F \; max_{D_X}$ $\mathcal{L}_{GAN}(F, D_X, Y, X)$, i.e. :

$$\mathcal{L}_{GAN}(F, D_X, Y, X) = E_{x\sim p_{\text{data}}(x)}\left[\log D_X(x)\right] + E_{y\sim p_{\text{data}}(y)}\left[\log\left(1 - D_X(F(y))\right)\right]. \tag{2}$$

In addition, a Cycle Consistency Loss $\mathcal{L}_{cyc}$ ensures that translating from one domain to another and back again reconstructs the original image. It is expressed as :

$$\mathcal{L}_{cyc}(G, F) = E_{x\sim p_{\text{data}}(x)}\left[\|F(G(x)) - x\|_1\right] + E_{y\sim p_{\text{data}}(y)}\left[\|G(F(y)) - y\|_1\right]. \tag{3}$$

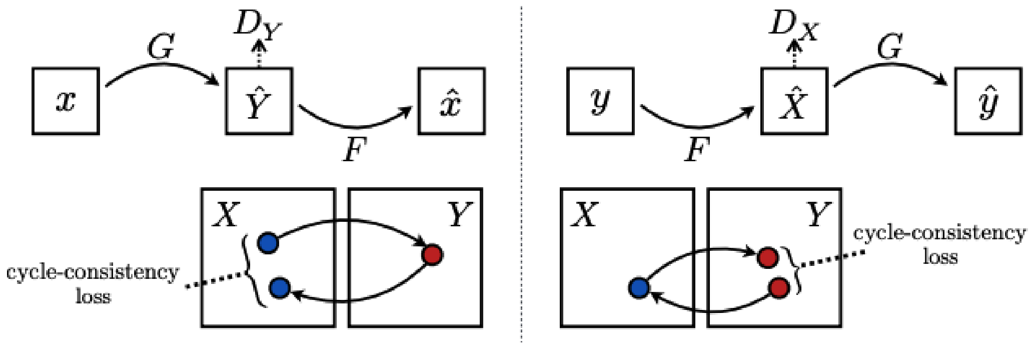


Figure 2: Forward and Backward consistency losses [5])

Finally, we express the full objective loss as :

$$\mathcal{L}(G, F, D_X, D_Y) = \mathcal{L}_{GAN}(G, D_Y, X, Y) + \mathcal{L}_{GAN}(F, D_X, Y, X) + \lambda\,\mathcal{L}_{cyc}(G, F), \tag{4}$$

Where λ controls the relative importance of the two objectives. With these losses now being presented, the problem that CycleGAN solves is expressed as:

$$\min_{G,F}\;\max_{D_X,D_Y}\;\mathcal{L}(G, F, D_X, D_Y) \tag{5}$$

.

2 Experiments

2.1 Architectural design

Both generators are based on ResNet's architecture. Each generator uses two stride-2 convolutions for downsampling, followed by six residual blocks, then two ConvTranspose2d layers (fractionally-strided convolutions) for upsampling. Residual blocks implement skip connections (x + block(x)). Instance normalization (InstanceNorm2d) is used throughout. ReLU activations appear after each convolution in the encoder/downsampling and upsampling stages; the final output layer uses Tanh. The generator's architecture is as follows (only one of the six ResBlock is represented here) :

```
Generator(
  (Gen): Sequential(
    (0): ReflectionPad2d((3, 3, 3, 3))
    (1): Conv2d(3, 64, kernel_size=(7, 7), stride=(1, 1), bias=False)
    (2): InstanceNorm2d(64, eps=1e-05, momentum=0.1, affine=False, track_running_stats=False)
    (3): ReLU(inplace=True)
    (4): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
    (5): InstanceNorm2d(128, eps=1e-05, momentum=0.1, affine=False, track_running_stats=False)
    (6): ReLU(inplace=True)
    (7): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
    (8): InstanceNorm2d(256, eps=1e-05, momentum=0.1, affine=False, track_running_stats=False)
    (9): ReLU(inplace=True)
    (10): ResBlock(
      (block): Sequential(
        (0): ReflectionPad2d((1, 1, 1, 1))
        (1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), bias=False)
        (2): InstanceNorm2d(256, eps=1e-05, momentum=0.1, affine=False, track_running_stats=False)
        (3): ReLU(inplace=True)
        (4): ReflectionPad2d((1, 1, 1, 1))
        (5): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), bias=False)
        (6): InstanceNorm2d(256, eps=1e-05, momentum=0.1, affine=False, track_running_stats=False)
      )
    )
    (11): ConvTranspose2d(256, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), output_padding=(1, 1), bi
    (12): InstanceNorm2d(128, eps=1e-05, momentum=0.1, affine=False, track_running_stats=False)
    (13): ReLU(inplace=True)
    (14): ConvTranspose2d(128, 64, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), output_padding=(1, 1), bia
    (15): InstanceNorm2d(64, eps=1e-05, momentum=0.1, affine=False, track_running_stats=False)
    (16): ReLU(inplace=True)
    (17): ReflectionPad2d((3, 3, 3, 3))
    (18): Conv2d(64, 3, kernel_size=(7, 7), stride=(1, 1))
    (19): Tanh()
  )
)
```

The discriminator follows the typical 70×70 PatchGAN [2] design: multiple 4×4 convolutions with strides (2, 2, 2, 1, 1) and padding that produce a dense output map, classifying overlapping patches rather than the entire image. The discriminator's architecture is as follows :

```
Discriminator(
  (Disc): Sequential(
    (0): Conv2d(3, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
    (1): LeakyReLU(negative_slope=0.2, inplace=True)
    (2): Conv2d(64, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (3): InstanceNorm2d(128, eps=1e-05, momentum=0.1, affine=False, track_running_stats=False)
    (4): LeakyReLU(negative_slope=0.2, inplace=True)
    (5): Conv2d(128, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (6): InstanceNorm2d(256, eps=1e-05, momentum=0.1, affine=False, track_running_stats=False)
    (7): LeakyReLU(negative_slope=0.2, inplace=True)
    (8): Conv2d(256, 512, kernel_size=(4, 4), stride=(1, 1), padding=(1, 1), bias=False)
    (9): InstanceNorm2d(512, eps=1e-05, momentum=0.1, affine=False, track_running_stats=False)
    (10): LeakyReLU(negative_slope=0.2, inplace=True)
    (11): Conv2d(512, 1, kernel_size=(4, 4), stride=(1, 1), padding=(1, 1))
  )
)
```

The identity loss term $\mathcal{L}_{\text{identity}}$, mentioned in some experiments of [5], was not used in this work.

2.2 Hyperparameters

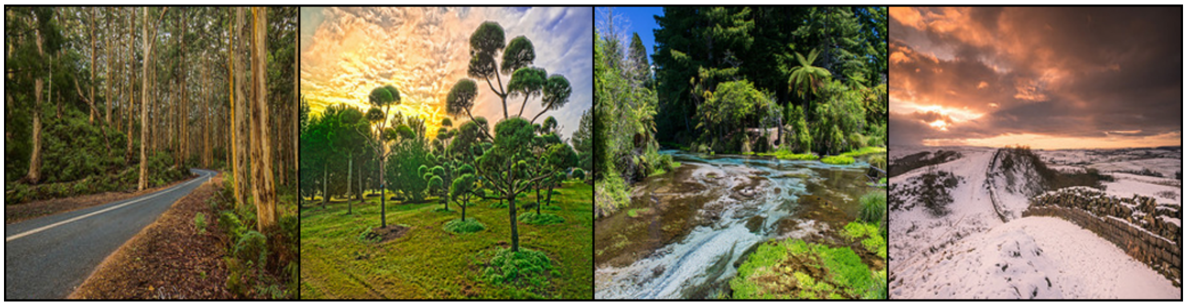
I trained a CycleGAN for 100 epochs using a Google Colab A100 GPU to learn a mapping between landscape photographs and Monet paintings. The goal is to obtain a model capable of transferring Monet's artistic style to real landscape images and, conversely, transforming Monet paintings into realistic landscape images. The parameters for this experiment, were set as follows :

Parameters	**Value**
GPU (Google Colab)	NVIDIA A100
Image size	256*256
Batch size	8
Total Epochs	100
Initial LR	2e-4
λ	10
Model capacity base widths (ngf / ndf)	64
Depth of residual paths (n_{blocks})	6
ADAM $beta_1$	0.5
ADAM $beta_2$	0.999
GAN loss type	MSELoss
Cycle loss type	L_1

Table 1: Hyperparameters - CycleGAN training

2.3 Results

I trained my model, on 100 epochs, to transfer style from real landscape photographs to Monet-style paintings, and vice versa.


(a) Original Landscapes

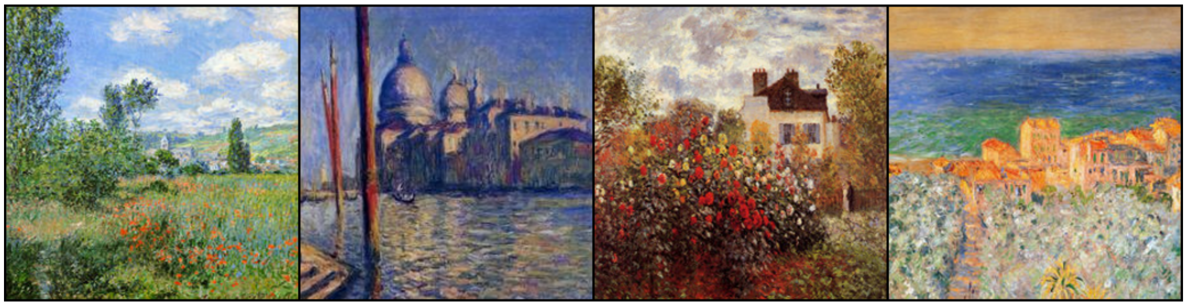
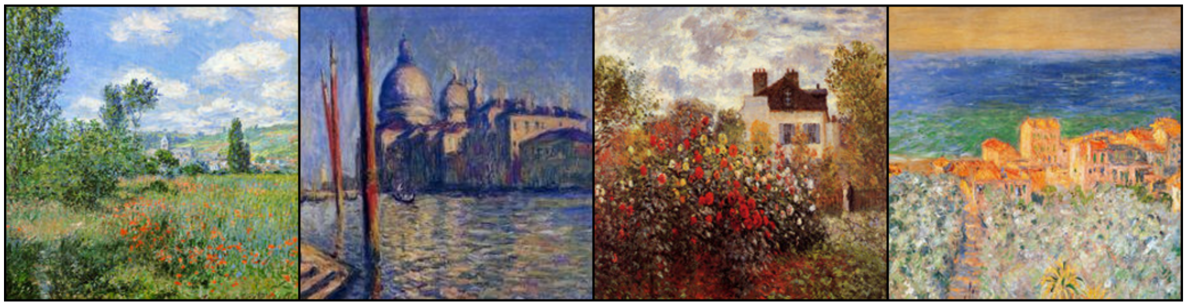
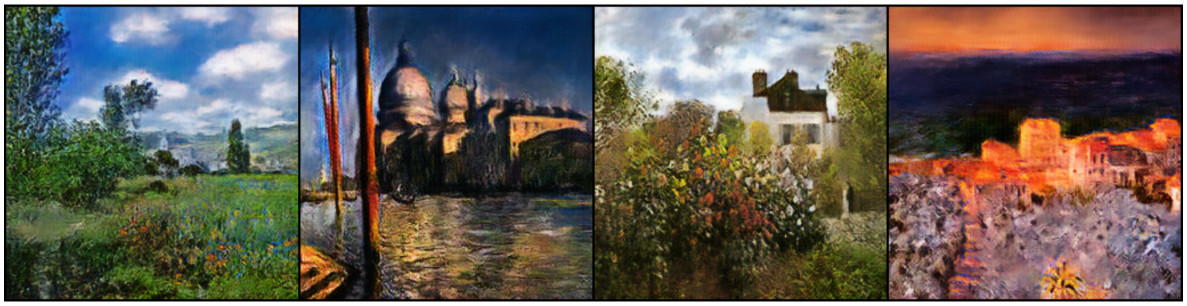
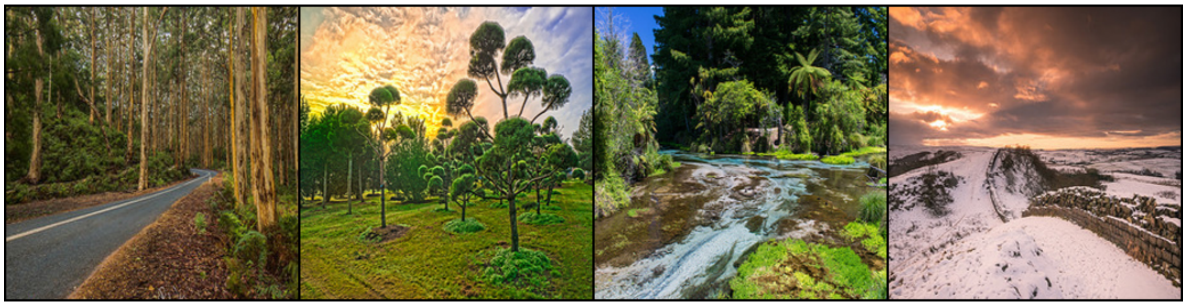

(b) Original Monet Paintings

Figure 3: Original Images

After 100 epochs, the model successfully translated styles. Qualitatively, the "Landscape to Monet" translation appears to perform better than the "Monet to Landscape" direction.


(a) Original Monet


(b) Monet to Landscape


(c) Original Landscape

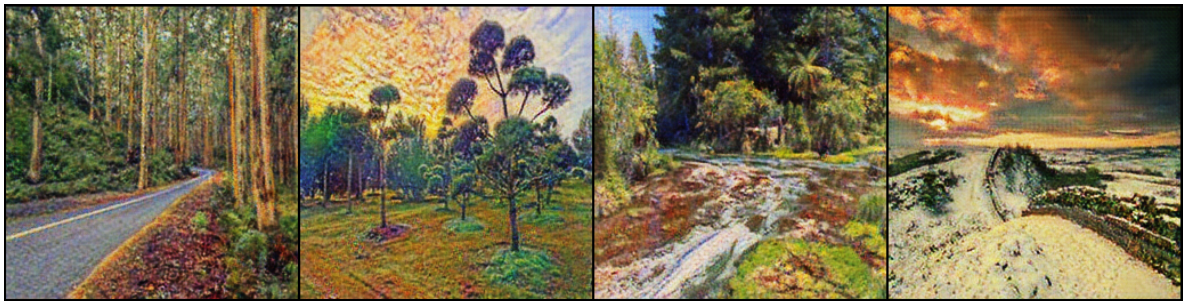
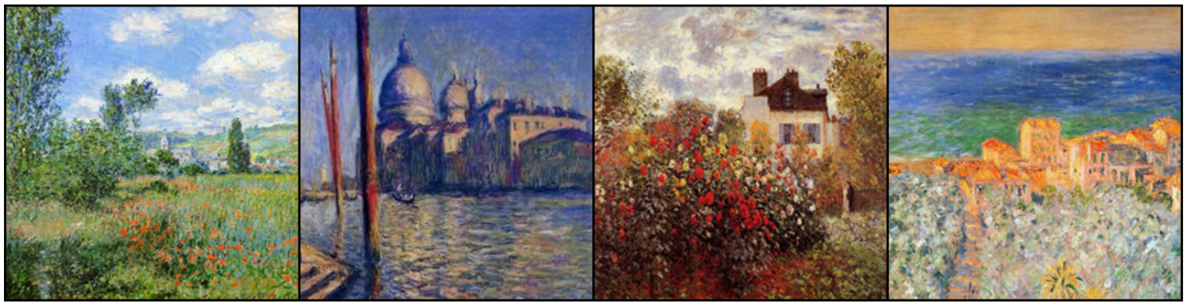
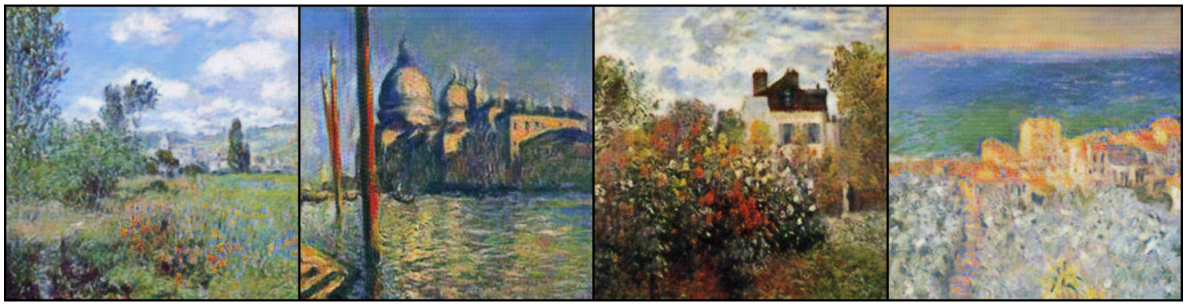
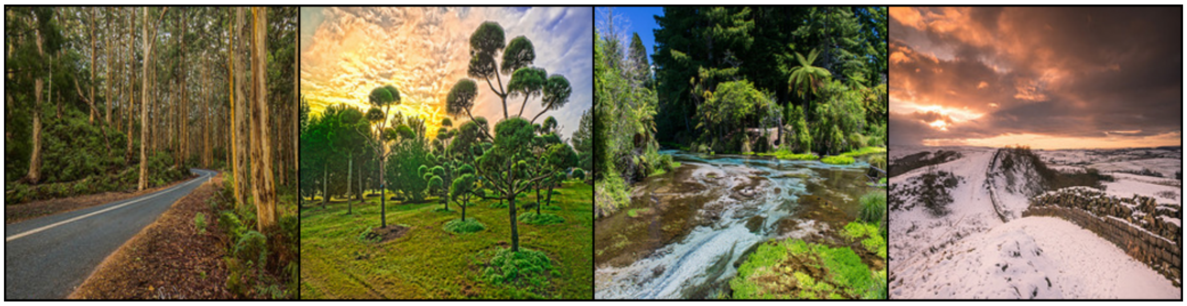

(d) Landscape to Monet

Figure 4: Style Transfer at 100 epochs

We can also observe that the model was effective in maintaining cycle consistency.


(a) Original Monet


(b) Monet Cycle Consistency


(c) Original Landscape

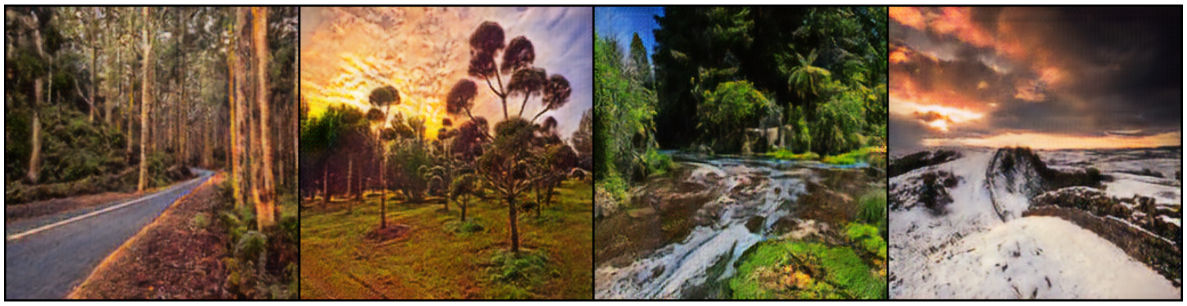

(d) Landscape Cycle Consistency

Figure 5: Cycle Consistency at 100 epochs

2.4 Conclusion

In this project, I implemented and trained a CycleGAN model for unpaired image-to-image translation between real landscape photographs and Monet-style paintings. The model demonstrated strong performance, especially in translating realistic landscapes into "Monet's" representations. This work confirmed the capacity of CycleGANs to perform style transfer without paired data. For future work, I plan to explore the Pix2Pix [2] model, which relies on paired datasets, to better understand the trade-offs between supervised and unsupervised image translation techniques.

3 Code

The code for the CycleGAN model is available on github : **https://github.com/selim-ba/**

References

[1] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014.

[2] Philip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A. Efros. Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1125–1134, 2017.

[3] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[4] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

[5] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 2223–2232, 2017.