Unpaired Image-to-Image Translation Using Cycle-Consistent Adversarial Networks

Mellissa HAFIS[1], Manda ANDRIAMAROMANANA[1], and Ilyes SAIS[1]

[1]Université Paris-Saclay, Department of Computer Science , Deep Learning Module

20/04/2025

Abstract

In this Deep Learning project, we implemented and experimented with the CycleGAN architecture for unpaired image-to-image translation (e.g. horses ↔ zebras). We describe the method, its context, our PyTorch implementation, as well as the datasets and evaluation metrics used.

1 Introduction

This paper tackles the *image-to-image translation* problem—learning mappings between two visual domains (e.g., horses ↔ zebras, photos ↔ paintings).

Unlike traditional methods that require paired training data (each input image having a direct target counterpart), this work focuses on **unpaired translation**, where only independent sets of images from two domains (X and Y) are available.

To address this challenge, CycleGAN introduces two key mechanisms:

- **Adversarial loss**, encouraging each generator to produce outputs indistinguishable from real images in the target domain.
- **Cycle-consistency loss**, ensuring that translating from X to Y and back (or Y to X and back) recovers the original image.

1.1 Problematic

1.1.1 Lack of Paired Data

Paired datasets are hard or impossible to obtain in tasks like style transfer or object transformation, making supervised methods inapplicable.

1.1.2 Under-constrained Mapping

Without alignment, a model may generate random or repetitive outputs (*mode collapse*) unrelated to the input.

1.1.3 Lack of Inverse Consistency

Unpaired models often fail to guarantee that a translation and its inverse preserve the original content. CycleGAN addresses this by jointly training two mappings ($G : X \to Y$ and $F : Y \to X$) and enforcing cycle-consistency:

$$F(G(x)) \approx x, \quad G(F(y)) \approx y$$

1.1.4 Training challenges

Relying only on adversarial losses causes unstable training and often loses important semantic details from the source image.

1.2 Objectives

The goal is to learn unpaired image translation with outputs that are both statistically realistic and semantically faithful. Our project includes:

[leftmargin=*]Adapting the official PyTorch CycleGAN implementation. Rewriting data loaders and core modules (Generator, Discriminator, GANLoss). Training with GPU acceleration, early stopping, and evaluation via FCN-score and perceptual metrics. Documenting architecture and performing qualitative analysis.

Core aims include:

- Achieving translation without paired data.
- Preserving semantic content (e.g., pose, structure).
- Avoiding mode collapse via cycle consistency.
- Demonstrating generality across domains (style, objects, enhancements).

1.3 Why this model of CycleGAN (motivation)?

1.3.1 Addressing the lack of Paired Data

CycleGAN offers a practical solution when aligned datasets are unavailable.

1.3.2 Incorporation of Cycle Consistency

Inspired by back-and-forth translation (e.g., language), it constrains mappings to be reversible and meaningful.

1.3.3 Enhanced stability and meaningful mapping

The combination of adversarial and cycle losses stabilizes training and ensures outputs retain relevant features of the input.

1.3.4 Broad applicability and robustness

CycleGAN generalizes well across tasks—style transfer, object transfiguration, and photo enhancement—without requiring explicit supervision or domain-specific tuning.

2 Related work

2.1 Generative Adversarial Networks (GANs)

Key Insight: GANs generate realistic images using adversarial loss to make outputs indistinguishable from real data.

Applications: Image generation, editing, text-to-image synthesis, inpainting, and more.

CycleGAN's Use: Leverages adversarial loss to ensure realism in cross-domain image translation.

2.2 Paired Image-to-Image Translation

Early methods: Techniques like *Image Analogies* relied on specific input-output pairs for texture and style transfer.

Modern approaches: *Pix2Pix* introduced conditional GANs trained on aligned image pairs for tasks like sketch-to-photo translation.

Limitation: These methods require paired data, which CycleGAN overcomes.

2.3 Unpaired Image-to-Image Translation

Previous methods:

- Bayesian/MRF-based patch priors
- *CoGAN:* Used weight-sharing to learn joint representations
- Liu et al.: Combined VAEs and GANs for cross-domain translation
- Style-Content Separation: Relied on predefined similarity constraints

CycleGAN's Innovation: Bypasses task-specific constraints or shared embeddings, enabling general-purpose unpaired translation.

2.4 Cycle Consistency

Past applications:

- Visual tracking (forward-backward checks)
- Machine translation (back-translation validation)
- Structure-from-motion and semantic alignment

CycleGAN's innovation: Introduces cycle consistency ($F(G(x)) \approx x$ and $G(F(y)) \approx y$) to the unpaired image-to-image setting.

Concurrent work: Yi et al. independently applied similar ideas inspired by dual learning.

2.5 Neural style transfer

Approach: Transfers style from one image to another using deep feature statistics (e.g., Gram matrices).

Limitation: Operates on single images, not domain-level mappings.

CycleGAN's advantage: Learns transformations between entire image domains, enabling broader applications like object transfiguration and photo enhancement.

2.6 Conclusion

CycleGAN advances prior work by combining GANs and cycle consistency to eliminate the need for paired training data. It avoids fixed similarity metrics and single-image constraints, offering a flexible and general framework for domain-to-domain translation.

3 Description of the architecture

The architecture is designed to learn mappings between two image domains X and Y using unpaired training data. It is structured around two key mappings and two corresponding adversarial discriminators.

3.1 Mapping functions

Generator $G : X \to Y$: This generator translates images from domain X into images that resemble those in domain Y.

Generator $F : Y \to X$: This generator performs the inverse translation, converting images from domain Y into images that resemble those in domain X.

3.2 Adversarial discriminators

Discriminator D_Y: Discriminator D_Y distinguishes between real images $y \sim p_{data}(y)$ from domain Y and fake images $G(x)$ produced by the generator G. Its corresponding adversarial loss is given by:

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

The objective is to train G such that it minimizes this loss against an adversary D_Y that tries to maximize it.

Discriminator D_X: Similarly, discriminator D_X distinguishes between real images $x \sim p_{data}(x)$ from domain X and fake images $F(y)$ produced by the generator F. The loss for this mapping is:

$$\mathrm{L}_{GAN}(F, D_X, Y, X) = E_{x \sim p_{data}(x)}[\log D_X(x)] + E_{y \sim p_{data}(y)}[\log(1 - D_X(F(y)))]$$

3.3 Cycle consistency loss

Adversarial losses alone do not guarantee that the mapping will maintain correspondence between an individual input and its translated output. To enforce such consistency, a cycle consistency loss is introduced.

Forward Cycle Consistency: For an image $x \in X$, translating it to Y via G and then back to X via F should result in the

original image:

$$F(G(x)) \approx x$$

Backward Cycle Consistency: Similarly, for an image $y \in Y$:

$$G(F(y)) \approx y$$

The cycle consistency loss is expressed as:
$\mathrm{L}_{cyc}(G, F) = E_{x \sim p_{data}(x)}[\|F(G(x)) - x\|_1]$
$+ E_{y \sim p_{data}(y)}[\|G(F(y)) - y\|_1]$
The use of the L_1 norm here helps to quantitatively measure the deviation between the original and reconstructed images.

3.4 Full objective

The complete training objective combines the adversarial losses and the cycle consistency loss:
$\mathrm{L(G, F, D}_X, D_Y) = \mathcal{L}_{GAN}(G, D_Y, X, Y)$
$+ \mathcal{L}_{GAN}(F, D_X, Y, X) + \lambda \mathcal{L}_{cyc}(G, F)$
where λ is a hyperparameter that controls the relative importance of the cycle consistency term compared to the adversarial losses.

Autoencoder interpretation

This setup can be interpreted as training two "autoencoders":

- One autoencoder is given by $F \circ G : X \to X$, using the intermediate representation generated in domain Y.
- The other is $G \circ F : Y \to Y$.

These autoencoders ensure that translating back and forth should ideally reconstruct the original image.

4 Benefits of the architecture

- **Flexible data requirements:** Works with unpaired datasets from different domains, ideal when paired data is hard or impossible to obtain.
- **Improved mapping specificity via cycle consistency**
  - **Reduces ambiguity:** Cycle consistency limits the mapping space, preventing issues like mode collapse (e.g., mapping all inputs to the same output).
  - **Preserves content:** Ensures that translations retain the unique characteristics of the original image.
- **High Visual quality:** Adversarial loss drives the model to produce outputs indistinguishable from real images in the target domain.
- **Versatility and generality:** Task-agnostic design doesn't rely on handcrafted similarity metrics, making it suitable for diverse applications like style transfer, object transformation, and seasonal changes.
- **Balanced learning:** The combination of adversarial and cycle consistency losses ensures stable training and high-fidelity results.

5 Implementation

Below is an explanation of how to implement the CycleGAN architecture and a walkthrough of its training loop. We provide the PyTorch and Torch implementations. Check out more details in this GitHub repository.

5.1 Model architecture

5.1.1 Generators

We implement two symmetric ResNet-based generators $G : X \to Y$ and $F : Y \to X$, each composed of three stages:

1. **Initial convolution:**
   - Reflection padding of 3 pixels on all sides.
   - 7×7 convolution (input_nc→ngf channels), no bias.
   - Instance Normalization.
   - ReLU activation.
2. **Downsampling:** two successive blocks, each comprising
   - 3×3 convolution with stride 2 (doubling the number of channels),
   - Instance Normalization,
   - ReLU activation.

   This reduces spatial resolution by a factor of 4 and increases the channel dimension from ngf to $4\times$ngf.
3. **Residual blocks:** N ResNet blocks (we choose $N = 9$ for 256×256 images). Each block consists of
   - Reflection padding of 1 pixel,
   - 3×3 convolution (ngf·mult→ngf·mult),
   - Instance Normalization,
   - ReLU,
   - (optional) Dropout (0.5),
   - Reflection padding of 1 pixel,
   - 3×3 convolution,
   - Instance Normalization,
   - Skip connection adding input to output.
4. **Upsampling:** two blocks mirroring the downsampling stage, each comprising
   - 3×3 transposed convolution with stride 2 (halving the channel count),
   - Instance Normalization,
   - ReLU activation.
5. **Final convolution:**
   - Reflection padding of 3 pixels,
   - 7×7 convolution (ngf→output_nc),
   - Tanh activation to map outputs to $[-1, 1]$.

Key design choices:

[leftmargin=*]*Reflection padding* preserves edge information and avoids boundary artifacts. *Instance Normalization* encourages style-invariance. *Residual blocks* facilitate gradient flow and enable deep generators. *Tanh output* centers pixel values, matching our preprocessing normalization.

5.1.2 Discriminators

- DAD_A: Discriminates between real images of domain B and images generated by GAG_A.
- DBD_B: Discriminates between real images of domain A and images generated by GBG_B.

We implement a PatchGAN discriminator (70×70) for each domain (D_X and D_Y) using the `NLayerDiscriminator` class in `discriminators.py`. Each discriminator:

- Uses a series of convolutional layers with increasing filters, instance normalization, and LeakyReLU activations.
- Outputs a 1-channel feature map that classifies local patches as real or fake.
- Is initialized with normal-distributed weights (std = 0.02) for stable training.

The function `define_D` allows flexible configuration (e.g., number of layers, normalization type, GPU usage). Both discriminators are easily instantiated with:

$$D_X = \texttt{define\_D(input\_nc=3, ...)}$$

$$D_Y = \texttt{define\_D(input\_nc=3, ...)}$$

5.1.3 Loss functions

As described previously, the CycleGAN model combines three loss components: adversarial loss, cycle-consistency loss, and identity loss. The adversarial loss encourages realistic image generation via LSGAN, the cycle-consistency loss ensures invertibility between domains, and the identity loss preserves image structure when applicable.

The total generator loss is expressed as:

$$\mathcal{L}_G = \mathcal{L}_{GAN} + \lambda_{cyc}\mathcal{L}_{cyc} + \lambda_{idt}\mathcal{L}_{idt}$$

where λ_{cyc} and λ_{idt} balance the contributions of each term.

5.1.4 Training

The model is trained using two generators (G_A, G_B) and two discriminators (D_A, D_B). The training process follows these steps:

1. **Forward Pass:** Compute generated images $G_A(a)$ and $G_B(b)$, cycle reconstructions $G_B(G_A(a))$ and $G_A(G_B(b))$, and identity mappings $G_A(b)$ and $G_B(a)$.
2. **Generator Update:** Calculate the total generator loss and update G_A and G_B using the Adam optimizer.
3. **Discriminator Update:** Update D_A and D_B with real images and fake images sampled from a replay buffer (ImagePool) to stabilize training.

During training, the average loss values are stored in a CSV log file. At regular intervals, generated images are saved for visualization and models are checkpointed. The training can also resume from previously saved weights to continue from a specific epoch.

Due to time constraints, we performed only 100 epochs with a constant learning rate. In contrast, the paper's implementation used 100 epochs with a constant learning rate followed by another 100 epochs where the learning rate decayed linearly.

Also, we add a batch size of 1 during the training.

6 Dataset Description

6.1 Training Set

For all our experiments we use the standard `horse2zebra` dataset from the CycleGAN project. This dataset contains two unpaired image collections:

- **Domain A (horses)**: 1067 training images of horses, collected "in the wild" under varying poses, backgrounds and lighting conditions.
- **Domain B (zebras)**: 1334 training images of zebras, likewise diverse in viewpoint, scene composition and color.

Key properties:

- *Unpaired*: there is no one–to–one correspondence between horse and zebra images; during training we sample random pairs (x_i, y_j).
- *Resolution*: original images range roughly from 200×200 up to 600×500; we uniformly resize to 286×286 then apply a random 256×256 crop.
- *Augmentations*: in addition to the random crop, we apply horizontal flipping with probability 0.5.
- *Normalization*: pixel intensities are mapped from [0,255] to [–1,1] via $(I/127.5) - 1$.

During each training iteration, we draw a batch of size B from `trainA` and a batch of size B from `trainB`, pass them through the two generators G and F, compute adversarial and cycle-consistency losses, and update all networks jointly.

6.2 Test Set

The test split likewise contains two folders:

- **testA (horses)**: 120 held-out horse images.
- **testB (zebras)**: 120 held-out zebra images.

At test time, we:

1. Resize each image to 256×256 (no random crop or flip).

2. Normalize to [–1,1] with the same statistics as training.
3. Feed horses $\rightarrow G$ to produce *fake zebras*, and zebras $\rightarrow F$ to produce *fake horses*.
4. Save the outputs for qualitative inspection and quantitative evaluation (e.g. FCN–score).

Using this clear separation of train vs. test domains ensures that our reported FCN–scores and perceptual metrics reflect genuine generalization to unseen images. The diversity of backgrounds, lighting and animal poses in horse2zebra makes it a challenging benchmark for unpaired translation.

7 Evaluation and Results

7.0.1 Evaluation Metrics

To assess the semantic consistency between generated (`fake_A`) and original (`real_A`) images, we use a pretrained FCN-ResNet101 segmentation model. This network predicts pixel-wise semantic labels, allowing comparison of the structural content between the source and generated domains. The following metrics are computed:

- **Intersection over Union (IoU)**: Measures the overlap between predicted and ground truth segmentations:

$$\text{IoU} = \frac{P \cap G}{P \cup G},$$

  where P and G are the predicted and ground truth masks.

- **Pixel Accuracy**: Fraction of correctly classified pixels:

$$\text{Accuracy} = \frac{\#\{matchingpixels\}}{\#\{totalpixels\}}.$$

- **Mean Class-wise IoU**: Average IoU over all C semantic classes (excluding background):

$$\text{MeanClassIoU} = \frac{1}{C}\sum_{c=1}^{C}\frac{P_c \cap G_c}{P_c \cup G_c},$$

  with P_c, G_c the pixels of class c.

These metrics are computed per image-pair and averaged over the test set for an overall semantic-preservation score.

7.1 Results

Figure 1: The Real→Fake pairs with the highest FCN-score. Left: horses→zebras. Right: zebras→horses.

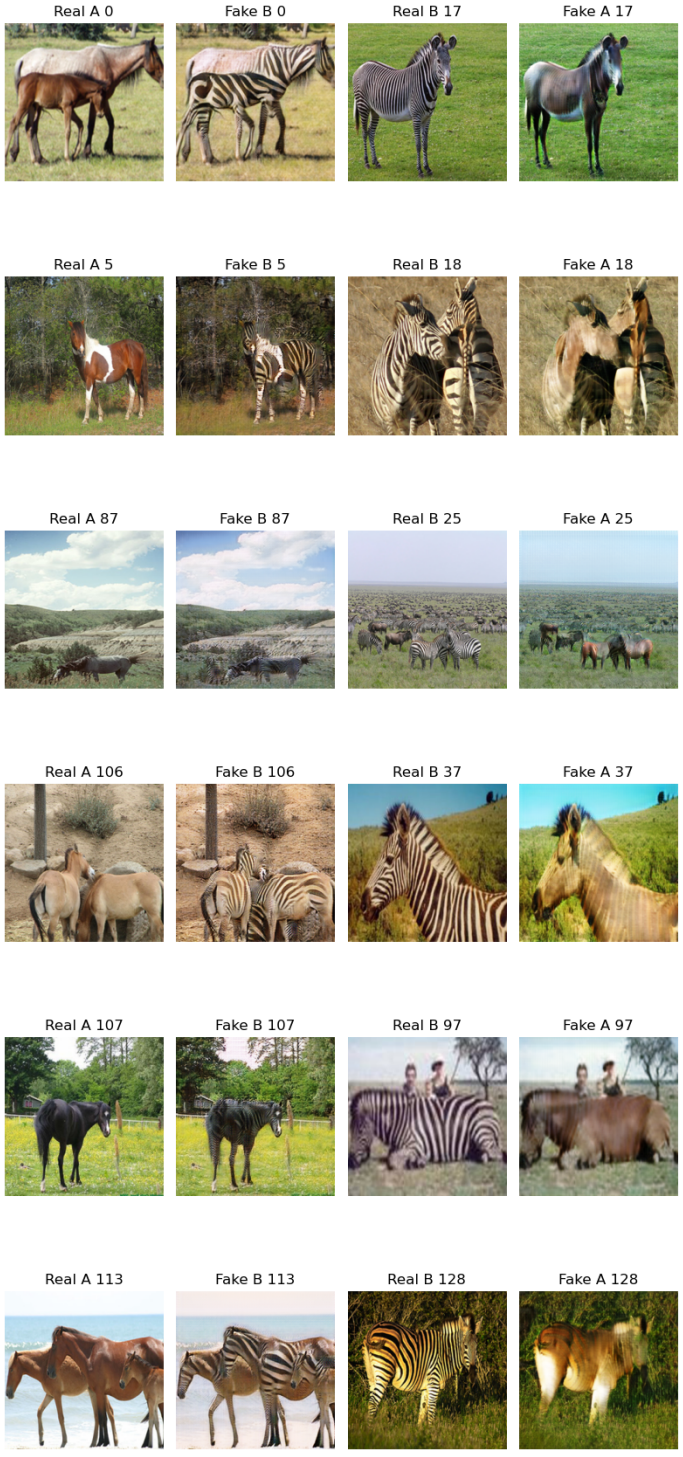

Figure 2: Additional top-scoring HorseZebra translations selected for qualitative inspection.

7.2 Evaluation

Comparison with CycleGAN paper metrics. The CycleGAN paper reports per-pixel and per-class accuracies, as well as class-wise IOU, for the task of labels → photo translation on the Cityscapes dataset. Our task, horse → zebra, is stylistic and unpaired, making a direct comparison of metric values less meaningful. Nevertheless, we follow a similar evaluation procedure by using an off-the-shelf semantic segmentation model (FCN-ResNet101) to approximate how well the generated image aligns semantically with the input. These scores provide a coarse proxy for semantic preservation in our translation task, though they are not directly comparable to those from the original CycleGAN setup.

Model	IOU	Pixel Accuracy	Mean Class IOU
CycleGAN (paper)	0.52	0.17	0.11
Cycle alone	0.22	0.07	0.02
GAN alone	0.51	0.11	0.08
GAN + forward cycle	0.55	0.18	0.12
GAN + backward cycle	0.39	0.14	0.06
Ours	**0.0983**	**0.6707**	**0.0654**

Table 1: Comparison of CycleGAN paper scores and our results. The metrics from the CycleGAN paper are based on the Cityscapes dataset for semantic segmentation tasks, while our scores were computed for horse → zebra style transfer using FCN-ResNet101.

8 Conclusion

In this project, we successfully implemented the CycleGAN architecture for unpaired image-to-image translation. By combining adversarial training with cycle consistency loss, we achieved meaningful domain translation without requiring paired data.

Throughout the project, we encountered some challenges. The main difficulty was related to training time and hardware constraints—training CycleGAN locally required a GPU, which limited experimentation. Additionally, coordinating development in parallel introduced challenges, particularly when integrating modules developed separately. For instance, Manda highlighted the difficulty of writing the training script without initially knowing the exact interfaces of the generator and discriminator components.

Despite these hurdles, our teamwork and regular meetings (1 to 2 times per week) allowed us to make steady progress and adjust as needed.

Each team member contributed significantly: - Mellissa implemented the discriminators and the evaluations models. - Ilyes developed the generators and the training loop and the testing loop. - Manda created the loss functions and managed the full training and testing runs using his GPU-equipped machine. - All members collaborated on the report and took part in the planning, coding reviews, and experimentation.

This project allowed us to gain hands-on experience with GAN architectures and strengthen our skills in collaborative machine learning development.