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

COURSE NAME: Deep Learning For Perception

COURSE CODE: CS4045

Group member ID's	Group member name's
21K-3398	Laiba Nadeem
21K-4502	Ahmed Mustafa Khokar
21K-3413	Avinash Ghai

Table of Contents

1. Introduction

The Cycle Generative Adversarial Network (CycleGAN) is an advanced deep learning framework introduced by Jun-Yan Zhu et al. (2017) to address the challenge of unpaired image-to-image translation. Traditional image translation models often require paired datasets—where each image in one domain has a corresponding image in another domain. However, acquiring such paired datasets is often impractical, especially for diverse domains such as photographs and artworks. This limitation restricts the scope of many image translation tasks, such as photo enhancement, image colorization, and style transfer.

CycleGAN overcomes this challenge by enabling image translation between two domains, such as celebrity photos (Domain A) and cartoon images (Domain B), without the need for paired images. Instead of relying on paired datasets, CycleGAN employs cycle consistency loss to ensure that an image, after being translated from one domain to another and then back to its original domain, remains close to the original. This approach allows for flexible and scalable image translation tasks, opening up various applications, such as automatic art style transfer and unpaired image domain adaptation.

This project focuses on utilizing CycleGAN for translating celebrity photos into cartoon images and vice versa, demonstrating its capability to handle unpaired datasets and produce high-quality image translations.

---

2. Project Objective

The main objectives of this project are:

- **2.1** Learn the mapping between two domains: Train a CycleGAN model to learn the mapping between celebrity photos (Domain A) and cartoon images (Domain B) without requiring paired datasets.
- **2.2** Generate high-quality image translations: Develop a model capable of translating celebrity images into cartoon images and vice versa, ensuring high-quality outputs that retain important features of the original images.
- **2.3** Optimize the CycleGAN model using loss functions: Fine-tune the generator and discriminator models by utilizing CycleGAN's core loss functions—adversarial loss, cycle consistency loss, and identity loss—ensuring improved performance and accurate translations.
- **2.4** Evaluate model performance: Assess the quality of the generated images and analyze how well the model has learned to translate between domains after several epochs of training.

---

3. Dataset

The project uses two image datasets:

- **3.1 CelebA Dataset**: A collection of celebrity photos used as Domain A.
- **3.2 CartoonSet10K Dataset**: A set of cartoon-style images used as Domain B.

The images are resized to 64x64 pixels for uniformity and stored in compressed .npz format for efficient processing.

---

4. Methodology

4.1 Defining the CycleGAN Architecture

4.1.1 Generators and Discriminators

Our implementation of the CycleGAN model consists of two **generators** and two **discriminators**:

- **Generator A->B**: Transforms images from Domain A (celebrity photos) to Domain B (cartoon images).
- **Generator B->A**: Transforms images from Domain B (cartoon images) to Domain A (celebrity photos).
- **Discriminator A**: Distinguishes between real and fake celebrity images.
- **Discriminator B**: Distinguishes between real and fake cartoon images.

The **generators** employ an **encoder-decoder architecture**, utilizing **convolutional layers** for downsampling and **ResNet blocks** for efficient feature extraction. The **discriminators** use **Convolutional Neural Networks (CNNs)** with **InstanceNormalization** and **LeakyReLU** activation functions to stabilize training.

4.2 Loss Functions Implementation

4.2.1 Adversarial Loss

The **adversarial loss** ensures that the generated images are realistic enough to deceive the discriminator. It is calculated using **Binary Cross-Entropy**:

adversarial_loss = BinaryCrossentropy(from_logits=True)

- **Generator A->B**: The generator is updated to produce images that can fool **Discriminator B**.
- **Discriminator B**: The discriminator is updated to correctly classify real cartoon images as real and generated images as fake.
- Similarly for **Generator B->A** and **Discriminator A**.

4.2.2 Cycle Consistency Loss

The **cycle consistency loss** ensures that the image retains its original content after being translated from one domain to another and back:

- **Forward Cycle Consistency Loss**: Ensures that an image translated from Domain A to Domain B and back to Domain A remains unchanged:
- forward_cycle_loss = tf.reduce_mean(tf.abs(G_B_to_A(G_A_to_B(real_A)) - real_A))
- **Backward Cycle Consistency Loss**: Ensures that an image translated from Domain B to Domain A and back to Domain B remains unchanged:
- backward_cycle_loss = tf.reduce_mean(tf.abs(G_A_to_B(G_B_to_A(real_B)) - real_B))

4.2.3 Identity Loss (Optional)

While not explicitly implemented in the current version, **identity loss** ensures that if an image is already in the target domain, it remains unchanged when passed through the generator:

identity_loss = tf.reduce_mean(tf.abs(G_B_to_A(real_B) - real_B))

4.3 Preparing the Training Data

We used two datasets for training:

- **3.1 CelebA Dataset**: A collection of celebrity images used as Domain A.
- **3.2 CartoonSet10K Dataset**: A set of cartoon-style images used as Domain B.

The images were resized to 64x64 pixels for consistency, and they were stored in a compressed format for efficient processing. **Data augmentation**, though not implemented in this version, could be added to further enhance the dataset and prevent overfitting.

4.4 Training Process

4.4.1 Updating the Discriminators

Each **discriminator** is trained to distinguish between **real** and **fake** images. The discriminators are updated using the **adversarial loss**. For each discriminator, the loss is calculated as:

dA_loss = d_model_A.train_on_batch(real_A, real_labels) # For Discriminator A

dB_loss = d_model_B.train_on_batch(real_B, real_labels) # For Discriminator B

The **discriminators** are updated to minimize the **adversarial loss**, which encourages them to correctly classify real and fake images.

4.4.2 Updating the Generators

The **generators** are updated using both the **adversarial loss** and **cycle consistency loss**. The goal is for the generator to produce images that are realistic (according to the adversarial loss) and consistent with the original (according to the cycle consistency loss). The updates for the generators are as follows:

- **Generator A->B**: Updated by minimizing the **adversarial loss** and **forward cycle consistency loss**:

- gen_loss_AtoB = g_model_AtoB.train_on_batch(real_A, fake_B)
- **Generator B->A**: Updated by minimizing the **adversarial loss** and **backward cycle consistency loss**:
- gen_loss_BtoA = g_model_BtoA.train_on_batch(real_B, fake_A)

4.4.3 Total Loss Calculation

The **total loss** for the generators combines the **adversarial loss** and **cycle consistency loss**. For each generator, the total loss is calculated as:

total_loss = adversarial_loss + lambda_cycle * (forward_cycle_loss + backward_cycle_loss)

Where lambda_cycle is a hyperparameter that controls the weight of the cycle consistency loss in the total loss.

The **total loss** is used to update both the generators simultaneously.

4.4.4 Image Pooling

During training, an **image pool** is maintained to prevent the model from overfitting. The image pool helps ensure that the generated images do not become too similar, improving the diversity of the outputs.

4.5 Hyperparameter Selection

The performance of the CycleGAN model was highly sensitive to the choice of hyperparameters. The following hyperparameters were selected:

4.5.1 Learning Rate (0.0002)

A learning rate of 0.0002 was used to ensure stable convergence without overshooting during training:

learning_rate = 0.0002

4.5.2 Beta 1 in Adam Optimizer (0.5)

Beta 1 is set to 0.5 in the Adam optimizer to control the momentum term and stabilize training:

optimizer = Adam(learning_rate, beta_1=0.5)

4.5.3 Number of Residual Blocks (9)

9 residual blocks were used in the generator to balance model performance and training time:

n_resnet = 9

4.5.4 Number of Filters in Convolutional Layers

The number of filters used in each convolutional layer was set to **[64, 128, 256, 512]** to capture progressively more complex features:

filters = [64, 128, 256, 512]

4.5.5 Batch Size (1)

A batch size of **1** was used to promote diversity in the generated images and avoid overfitting:

batch_size = 1

4.5.6 Loss Weights

Lambda_cycle was set to **10** to emphasize the importance of cycle consistency loss during training:

lambda_cycle = 10

4.6 Model Saving and Visualization

To monitor progress, the generator models were saved periodically during training:

g_model_AtoB.save('generator_AtoB_model.h5')

g_model_BtoA.save('generator_BtoA_model.h5')

The generated images were visualized regularly to assess the quality of translations:

plt.imshow(generated_image)

plt.show()

4.7 Model Evaluation

The performance of the model was evaluated by periodically visualizing the generated images. This allowed us to monitor the reduction of artifacts and improvements in image quality over multiple epochs.

---

5. Feature Extraction Techniques

Feature extraction refers to transforming the input images into meaningful representations. The CycleGAN model relies on several techniques for effective feature extraction:

1. **Convolutional Layers (Conv2D)**:
    - Extract local features like edges and textures in the early layers.
    - Higher layers capture abstract, high-level features.
    - Filters (64, 128, 256) progressively capture more complex features.
2. **InstanceNormalization**:
    - Normalizes each feature map independently to stabilize training, especially useful for style transfer tasks.
3. **Residual Blocks (ResNet)**:

    - Residual blocks allow the model to learn complex transformations while preserving important image features through skip connections.

4. **Downsampling Layers (Strided Convolutions)**:
    - Use strided convolutions to reduce the image size and increase the number of filters, capturing higher-level features.
5. **Upsampling Layers (Conv2DTranspose)**:
    - These layers are responsible for upsampling the image to match the target image size while maintaining learned features.
6. **Tanh Activation**:
    - Used to scale the generated image's pixel values to the range [-1, 1], which matches the range used in CycleGAN's output layer.

---

6. Hyperparameter Details

In CycleGAN, several hyperparameters critically impact model performance:

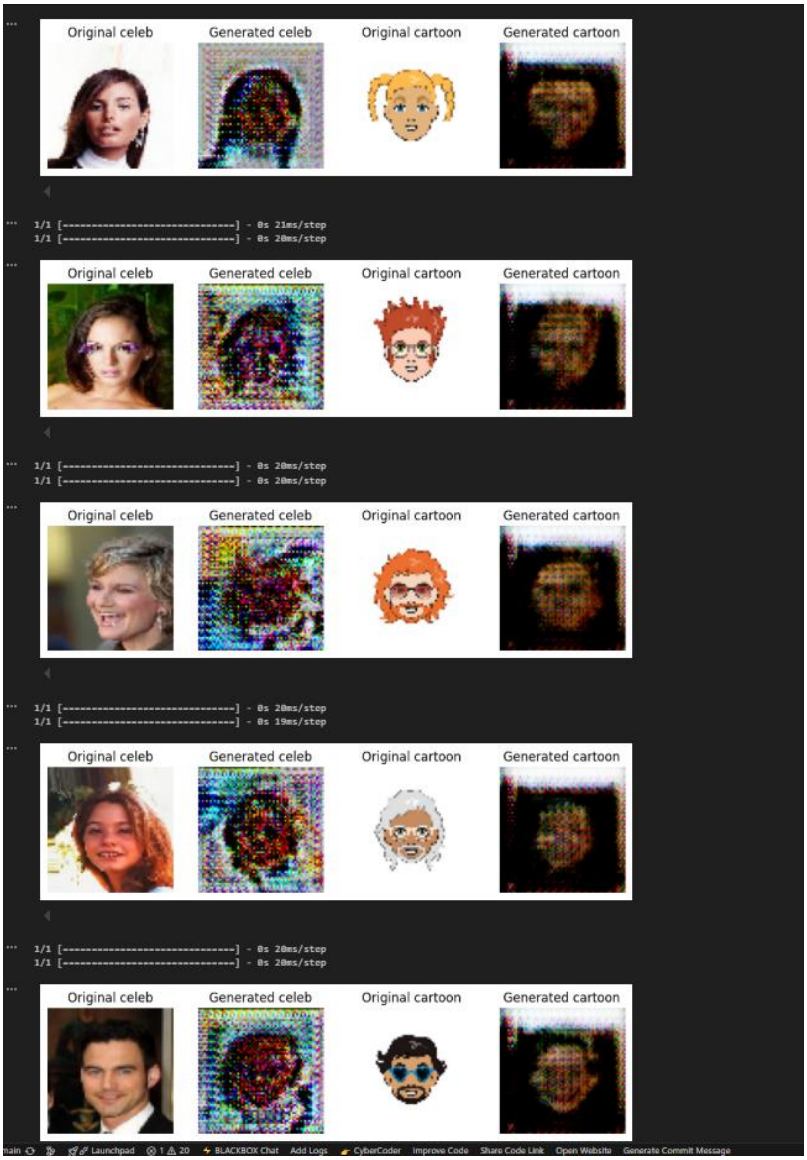
1. **Learning Rate (0.0002)**: A lower learning rate prevents the model from overshooting during optimization.
2. **Beta 1 (0.5) in Adam Optimizer**: Helps stabilize GAN training by controlling momentum.
3. **Number of Residual Blocks (9)**: Balances performance and training time.
4. **Number of Filters (64, 128, 256, 512)**: Determines the model's ability to extract features.
5. **Loss Weights (1, 5, 10, 10)**: Balances the importance of each loss during training.
6. **Batch Size (1)**: Promotes diversity in the generated outputs.

---

7. Results and Visualization

After training the model, generated images were visualized periodically to evaluate the quality of translations between domains. The experiment focused on unpaired image-to-image translation between celebrity photos (Domain A) and cartoon images (Domain B).

Results After 5 Epochs



After 5 epochs, the CycleGAN model showed progress, but the output still contained noticeable noise and artifacts.

- **Original Celeb and Generated Celeb**: The Generated Celeb images showed minor distortions and pixelation.
- **Original Cartoon and Generated Cartoon**: The Generated Cartoon images were noisy, with visible pixelation and fragmented details.

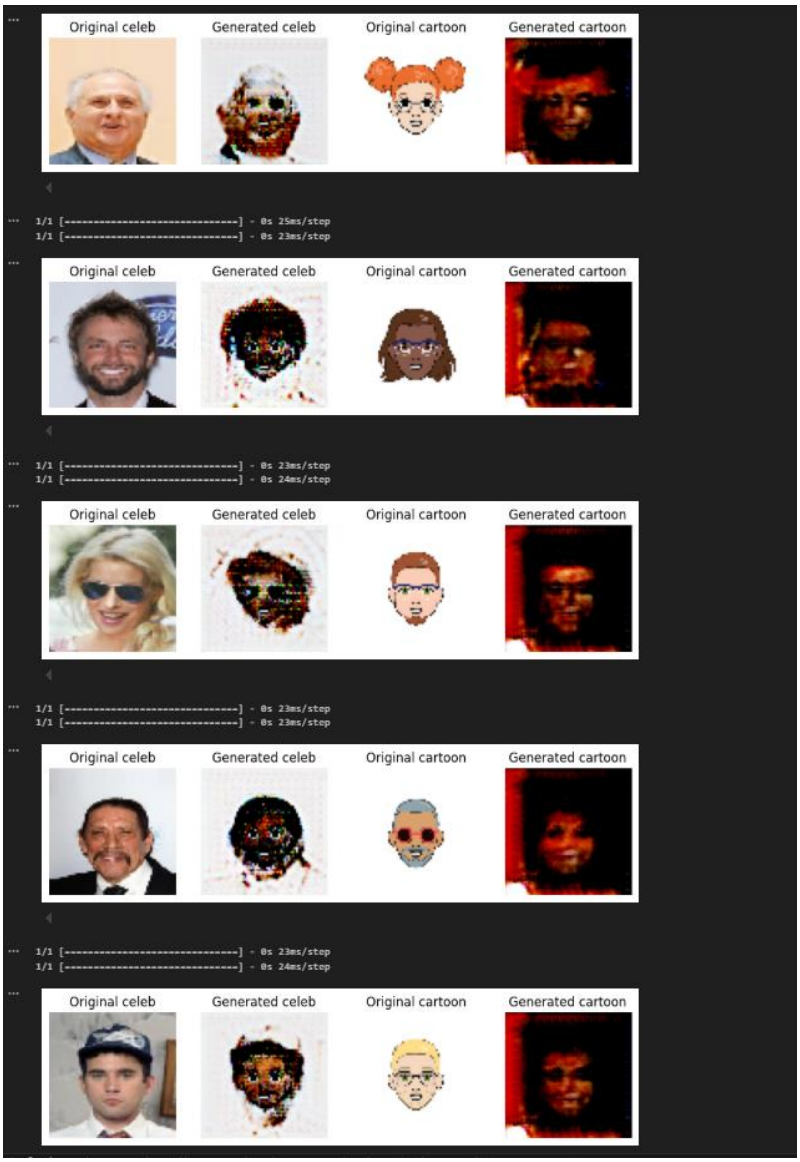
Progression:

- The images are still in an early phase of refinement, with significant noise, especially in the Generated Cartoon images. This is typical for early epochs.

Conclusion:

- 5 epochs are insufficient for generating fully refined images. More training is needed for clearer results.

Results After 10 Epochs



After 10 epochs, the Generated Celeb and Generated Cartoon images showed visible improvement.

- **Original Celeb and Generated Celeb**: The Generated Celeb images had reduced pixelation but were still blurry in some cases.

- **Original Cartoon and Generated Cartoon**: The Generated Cartoon images showed improvement but still contained pixelation and blurriness.

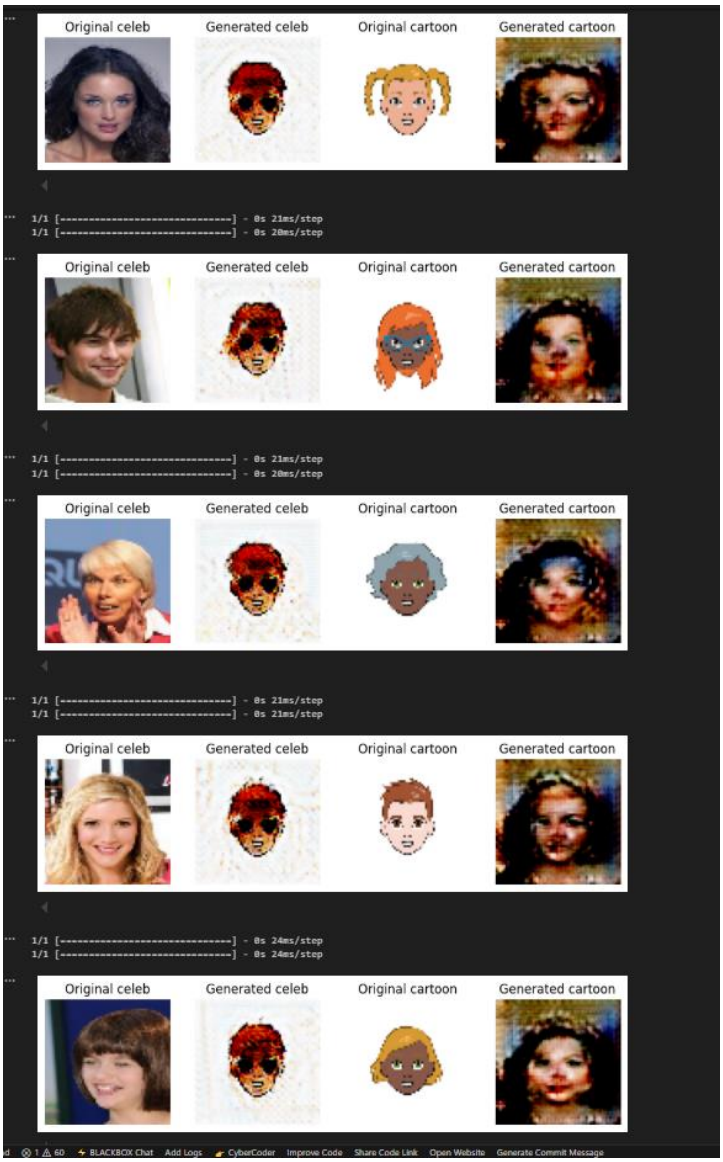
Progression:

- The images are less noisy than after 5 epochs, but the Generated Cartoon images still lack the clarity of the original cartoons.

Conclusion:

- After 10 epochs, further training is required to reduce pixelation and improve the quality of Generated Cartoon images.

Results After 15 Epochs



After 15 epochs, the generated images showed significant improvement in both clarity and structure.

- **Original Celeb and Generated Celeb**: The Generated Celeb images were clearer, with significantly less pixelation.
- **Original Cartoon and Generated Cartoon**: The Generated Cartoon images were more refined, though still somewhat blurry around facial features.

Progression:

- The Generated Celeb images are now nearly realistic, and Generated Cartoon images are more detailed, though still lacking full sharpness.

Conclusion:

- 15 epochs show clear progress, but further training is needed for sharper, more accurate outputs.

Comparison Table: Results After 5, 10, and 15 Epochs

Epochs	Generated Celeb	Generated Cartoon
5 Epochs	- Minor distortions and pixelation - Early stage of learning	- Noisy and pixelated - Fragmented details, not fully learned
10 Epochs	- Less pixelation, but still blurry in some areas	- Improved structure - Still some pixelation and blurriness
15 Epochs	- Clearer, less pixelation - More detailed and structured	- Refined translation - Some blurriness, still not as sharp as originals

This table provides a concise overview of the progression in the Generated Celeb and Generated Cartoon images over the course of the 3 different epoch milestones.

---

8. Future Work

Future work could involve:

- Improving model architecture.
- Implementing advanced data augmentation strategies.
- Scaling up the dataset for better generalization.

---

9. References

1. Zhu, J. Y., Park, T., Isola, P., & Efros, A. A. (2017). Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2242-2251. https://doi.org/10.1109/ICCV.2017.244

2. Abadi, M., et al. (2016). TensorFlow: A System for Large-Scale Machine Learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*.

3. Ulyanov, D., et al. (2016). Instance Normalization: The Missing Ingredient for Fast Stylization. arXiv:1607.08022.

4. Zhu, J. Y., et al. (2017). Cycle Consistency Loss in GANs. *IEEE ICCV 2017*.