CS424 – Group Project Report

Akeela Darryl Fattha
akeelaf.2022@scis.smu.edu.sg

Fadhel Erlangga Wibawanto
fadhelew.2022@scis.smu.edu.sg

Tan Zhi Rong
zhirong.tan.2022@scis.smu.edu.sg

Grace Angel Bisawan
gbisawan.2022@scis.smu.edu.sg

Lee Jia Heng
jiaheng.lee.2023@scis.smu.edu.sg

Abstract

This report provides an overview of Cycle-Consistent Adversarial Networks (CycleGAN), a technique for unpaired image-to-image translation. We explore the architecture, key innovations, applications, and limitations of CycleGAN models in computer vision tasks.

Contents

1. Task 1

1.1 Introduction

We attempt to create a CyleGAN using novel and modern techniques to improve the performance of the model to convert cartoon faces to realistic faces, and vice-versa. We will be using improved blocks, various attention mechanisms and normalisation techniques in our generators, as well as additional loss functions to better guide the models to learn the mapping between the two domains. We opted not to use any augmentations to our images, and we use an image size of 256x256 as our input/output size.

With our enhancements, we achieved the following results:

	FID	**IS**	**GMS**	**Avg GMS**
Raw to Cartoon	46.70193	2.41466 ± 0.24392	4.39783	3.85666
Cartoon to Raw	39.83461	3.62382 ± 0.23992	3.31548	

1.1.1 Configuration & Hyperparameters

All optimisers used Adam with the following hyperparameters and loss weights:

- Random seed: 42
- Torch seed: 16109772023028774379
- n epochs: 100
- learning rate: 2e-4
- betas: (0.5, 0.999)
- λ_{cyc}: 10.0
- λ_{id}: 5.0
- λ_{gan}: 1.0
- λ_{gp}: 10.0
- λ_{fm}: 5.0

1.2 Architecture

1.2.1 Generator

Our generator incorporates several modern attention mechanisms and architectural techniques including Convolutional Block Attention Module (CBAM), Squeeze-and-Excitation (SE) blocks, and Self-Attention mechanisms for improved feature representation, generating higher-quality images with better detail preservation, style consistency, and structural coherence.

Some of our key design choices:

- **Instance Normalization:** Used throughout the network instead of batch normalization, as it has been shown to produce better results for style transfer and image-to-image translation by normalizing each instance independently.
- **Reflection Padding:** Applied before convolutions to reduce boundary artifacts that can appear in generated images, particularly important for maintaining realistic edges.
- **Mixed + Improved Residual Blocks:** The strategic placement of different residual block types allows the network to benefit from complementary approaches to feature transformation. They also exhibit the following characteristics and sub-blocks:
  - **Channel Attention:** Models interdependencies between channels using both max and average pooling operations.
  - **Spatial Attention:** Focuses on important spatial regions by creating attention maps from channel-wise statistics.v
  - All blocks benefit from SE attention for channel recalibration

- **Squeeze-and-Excitation Attention:** Used in Mixed Residual Blocks, it recalibrates channel-wise feature responses by explicitly modeling interdependencies between channels, allowing the network to selectively emphasize informative features.
- **Enhanced Self-Attention:** Positioned after the residual blocks, it helps ensure global coherence in the generated images by modeling long-range dependencies that convolutional operations cannot capture efficiently. This is particularly valuable for maintaining structural integrity in facial regions.

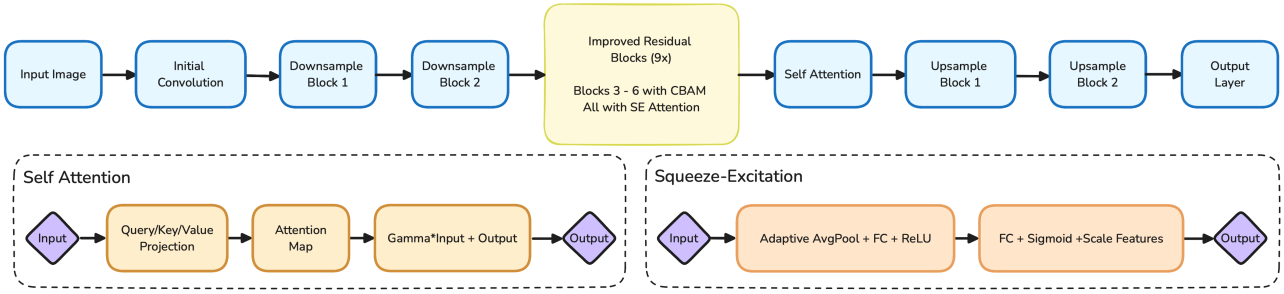
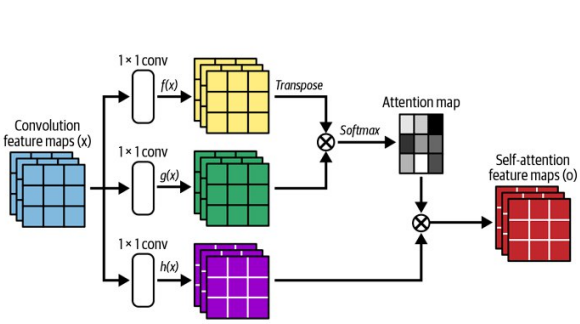
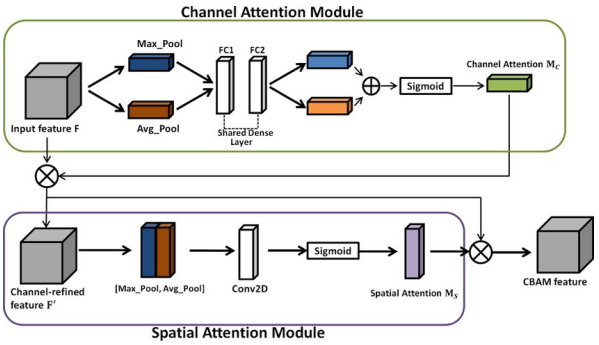


Figure 1.1: Generator Architecture



Self Attention Block



1.2.2 Discriminator

Our discriminator uses a basic sequential architecture with convolutional layers that doubles in number of channels 3 times, from 64 to 512. Each convolution output is passed to a spectral norm layer, instance normalisation and a leaky relu activation function. We then branched into a Avg Pool into a FC layer for global classification and a PatchGAN block for local classification.

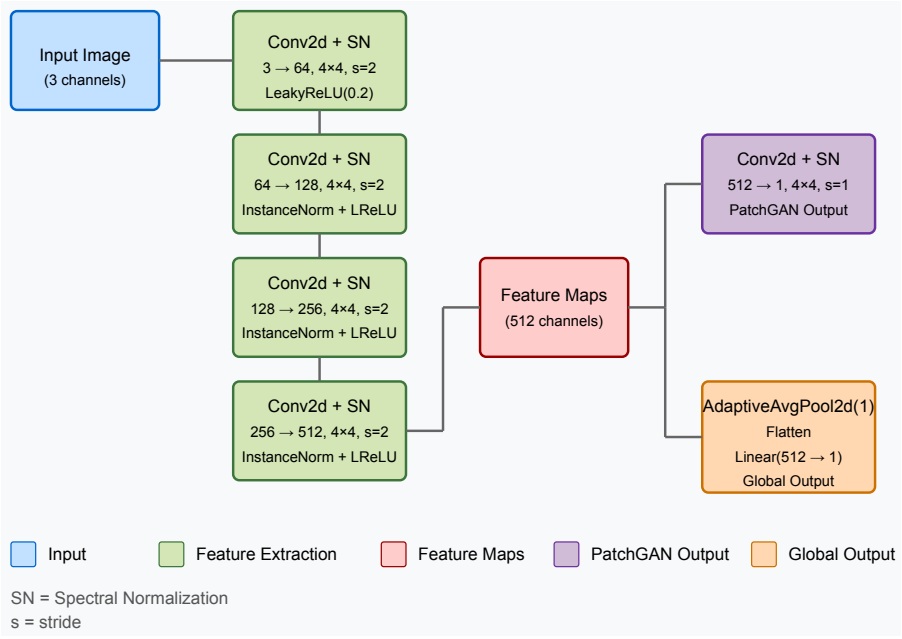


Figure 1.2: Discriminator Architecture

1.3 Data Preparation

1.3.1 Augmentations

We started with resizing images to 128x128, paired with a batch size of 16 before we hit a memory limit. We also applied no augmentations to the images, as a baseline to compare against later on when we add augmentations.

After adding various minor augmentations like affines, color jitters, flips, light gaussian blurs and posterising, we found that any combination of them resulted in worse metrics overall. From visual inspection, we found that the generated raw images were more noisy than without augmentations, which could contribute to the worse metrics. Furthermore, the generated cartoon images were a lot more flat than the target images, losing a lot more of the finer details and textures. Lastly, affines and any crops were clearly visible in the generated images when they otherwise should not be.

Thus, in both cases, we believe that adding the augmentations were encouraging the model to over-process the style conversions, which likely caused the reduction in metrics. Finally our best result was achieved with no augmentations, and upscaling the images to 256x256 with a batch size of 4.

256x256 images Avg GMS: 3.85666

128x128 images Avg GMS: 4.47829

1.3.2 Training/Validation

We worked with simple 99/1 split on training/validation data since we found that we needed as much data as possible for training the model to achieve our best results.

We also made use of an "Image Pool" to help in training. This pool functions as a buffer to store previously generated images, which helps to reduce the risk of mode collapse where the Discriminator becomes too powerful, and prevents the Generator from learning anything new. The pool is updated with new generated fake images during training, and once the threshold is reached, we randomly sample from the available pool instead of using the latest generated image.

1.4 Loss Functions

1.4.1 Discriminator Losses

We used a combination of loss functions beyond the standard adversarial loss to improve the performance of our discriminator. Our overall loss function for the discriminator is as follows:

$$L_D = L_{GAN} + \lambda_{gp} L_{gp} \tag{1.1}$$

Gradient Penalty – L_{gp}

We also use a gradient penalty, inspired from Wasserstein GAN (WGAN)[1] in order to prevent the discriminator from learning too fast. Doing so avoids problems like mode collapse with vanishing gradients, and ensures that the generator is always learning, even if it is not performing well enough to "beat" the discriminator yet, thus improving stability of training.

Relativistic – L_{GAN}

We extend the typical adversarial loss by using a relativistic loss, which compares the logits of discriminating the real and fake images in a relative manner on a global scope with Binary Cross Entropy Loss. Furthermore, we took inspiration from PatchGAN[2] which whether the images is real or fake based on smaller patches, a local scope. Thus, we have another source of information for determining real or fake, which is computed with MSE loss. The equation for our relativistic loss for our "real" side is as follows, and the opposite will be applied for the "fake" side i.e. $D_{real} <=> D_{fake}, True <=> False$:

$$L_{GAN} = 0.7 \cdot MSE(D_{real} - \bar{D}_{fake}, True)_{patch} + 0.3 \cdot BCE(D_{real} - \bar{D}_{fake}, True)_{global} \tag{1.2}$$

1.4.2 Generator Losses

The generator also makes use of the **Relativistic** loss, which is an extension of the standard adversarial GAN loss. Our overall loss function for the generator is as follows:

$$L_G = L_{GAN} + \lambda_{cyc} L_{cyc} + \lambda_{id} L_{id} + \lambda_{feat} L_{feat} \tag{1.3}$$

Cycle Consistency – L_{cyc}

As we would like the fake images to be able revertible to its original image i.e. the pixels are identical, we maintain using L1 Loss for our Cycle Consistency.

Identity – L_{id}

Similarly, we also use L1 Loss for our Identity loss since images in the same domain should not be altered.

Feature – L_{feat}

Another extension that we incorporated is the Feature Loss, which serves as a perceptual loss that compares the features of the generated image with the target image. Although traditionally used with pre-trained networks, we used the features extracted from the training discriminator instead. These features are compared using L1 loss, which helps to ensure that the generated images are perceptually similar to the target images.

[1] Arjovsky, M., Chintala, S., & Bottou, L. (2017, July 17). Wasserstein Generative Adversarial networks. PMLR. https://proceedings.mlr.press/v70/arjovsky17a.html

[2] Isola, P., Zhu, J., Zhou, T., & Efros, A. A. (2016). Image-to-Image Translation with Conditional Adversarial Networks. arXiv (Cornell University). https://doi.org/10.48550/arxiv.1611.07004

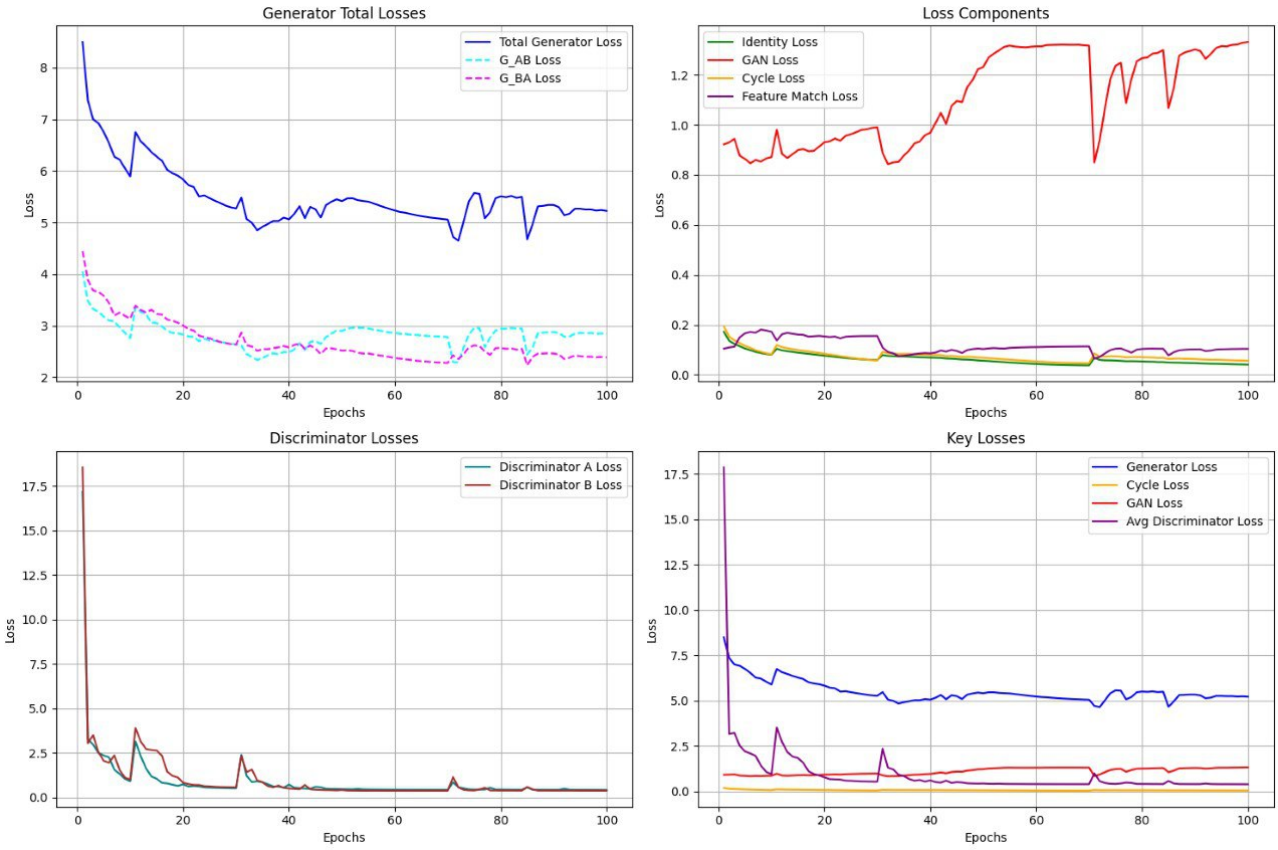


Figure 1.3: Overall loss curves

2. Task 2

2.1 Introduction

2.1.1 Configuration

2.2 Architecture

2.2.1 Discriminator

GANs consist of two networks: a generator that creates images and a discriminator that evaluates them. The two networks are trained adversarially, with the generator trying to fool the discriminator.

2.2.2 Generator

CycleGAN extends the GAN framework by using two generator-discriminator pairs, allowing translation between domains X and Y. The key innovation is the cycle-consistency loss, which ensures that translating an image to the target domain and back produces the original image.

2.3 Data Preparation

2.3.1 Pre-Processing

Used ChatGPT to generate a list of mapping of animal to pokemon variations
used a pretrained vgg16 to remove background for animals

2.3.2 Training/Validation

A simple 80/20 split on training data with validation

2.4 Loss Functions

Our overall loss functions for CycleGAN are as follows:

$$L_{total} = L_{GAN} + \lambda_{cyc} L_{cyc} + \lambda_{id} L_{id} + \lambda_{edge} L_{edge} + \lambda_{color} L_{color} \tag{2.1}$$

2.4.1 Discriminator

Patch

Least Squares

2.4.2 Generator

Adversarial

Use basic

Cycle Consistency

Identity

Edge Consistency

Color Consistency

2.4.3 Adaptive Loss Weighting