Image-to-Image Translation with CycleGAN
Mason Wang and Qizheng Zhang

Introduction

In this project, we work on image-to-image translation with CycleGAN. In particular, we train CycleGAN to map real human faces to faces of anime characters, and vice versa. We use the selfie2anime dataset, which is available here: https://www.kaggle.com/arnaud58/selfie2anime. The training set is composed of 3,400 images of real human faces and 3,400 images of faces of anime characters that are unpaired. The test set is composed of 100 images of real human faces and 100 images of faces of anime characters that are unpaired.

We have implemented three versions of CycleGAN. Besides that, as a baseline, we train the official PyTorch code released by the authors of the paper. We do this to evaluate the quality of our own implementation.

The first version of our own implementation could be found in the notebook "Final Cycle GANs Pytorch v1.ipynb". The generator and discriminator architecture of this implementation largely follow those of the original paper "Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks" [1]. We use least-squares loss for both the generator's adversarial loss and the discriminator's loss, use the same learning rate and optimizer, and opt to include the identity loss suggested in section 5.2. We keep an image cache of 25 instead of 50 images, and also have a different sampling procedure. The first version is based on references [3] and [4].

The second version improves upon the sampling procedure from the image cache from the first version. It adopts a different ratio of generator losses, suggested by [11]. The generator architecture changes slightly. Lastly, we implement a gradient penalty, following "Improved Training of Wasserstein GANs" [7], and "On the Effectiveness of Least Squares Generative Adversarial Networks". We reach the memory limits in Google Colab before we are able to finish an epoch of this model. The implementation can be found in the notebook "Final Cycle GANs Pytorch v2.ipynb".

The third version is identical to the second version, but without gradient penalty. This modification allows us to run the code and train the model without being bothered by memory issues. The implementation can be found in the notebook "Final Cycle GANs Pytorch v3.ipynb".

In this report, we will describe our model architecture, design choices, implementation strategies, and results & observations.

Baseline implementation

In order to obtain good baseline results for our implementation of Cycle-GANs, we train a Cycle-GANs network from https://github.com/junyanz/pytorch-CycleGAN-and-pix2pix on the dataset of unpaired anime and real human faces. In this codebase, the implementation of the Cycle-GAN network is done for us by Jun-Yan Zhu*, Taesung Park*, Phillip Isola, Alexei A. Efros. (*equal contributions). This codebase corresponds to the paper "Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks".

Version 1 Architecture

There are numerous differences between our implementation and the baseline model we trained using the code from "Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks."

Loss function

While the original paper mentions the LSGAN loss, we did not train the baseline model using this flag. We did adopt least-squares loss in our model. Using least-squares loss alters our generator and discriminator objectives and is supposedly more "stable and resistant to mode collapse" (Maire, 2022). Below is the discriminator objective and the adversarial objective of our generators.

$$D^* = \arg\min_D \left[\mathbb{E}_{x \sim p_{\text{data}}} (D(x) - 1)^2 + \mathbb{E}_{z \sim p} (D(G(z)))^2 \right]$$

Push discrim. response on real data close to 1 — Push response on generated data close to 0

$$G^* = \arg\min_G \mathbb{E}_{z \sim p} (D(G(z)) - 1)^2$$

Push response on generated data close to 1

In addition to the two adversarial losses, the two generators' loss also includes L1 cycle losses for each generator, weighted 5 times more than the adversarial loss, and identity losses for each generator, weighted 10 times more.

Discriminator

The architecture for our discriminator is based off of the architecture implemented in this tensorflow tutorial: https://machinelearningmastery.com/cyclegan-tutorial-with-keras/. This tensorflow implementation is based off of the architecture in the original paper.

The discriminator is composed of a stack of convolutional blocks that gradually increase the channel dimension while halving the spatial dimension at each step. In each convolutional block, we

1) Convolve the image with a kernel size of 2, stride of 2, and increase the number of channels
2) Apply instance normalization
3) Apply a LeakyReLU activation function (alpha = 0.2)

We increase the number of channels from 3 to 64 to 128 to 256 to 512. We convolve once more with a 512-channel to 512-channel layer, then perform instance normalization and apply a LeakyReLU activation. Then, we convolve down to a single channel. At this point, the spatial size of our layer is 16x16. The reason we convolve to an output layer of spatial dimension 16x16 instead of a single classification number is because a 16x16 spatial dimension may be more expressive, and could contain more information about where particular features occur on the image.

Instead of using Batch Normalization as the DCGAN paper does, we use instance normalization. This is recommended for tasks like style transfer because it erases style information [5].