Practices in visual computing II (Spring 2025)
Assignment 2
Image to image translation with CycleGAN

Total points: 100

Due: Friday, 21 February, 11:59 PM

Introduction

In 2015 generative adversarial networks (GANs) [1] took the computer vision world by storm. Their ability to generate high-quality and realistic images was unseen previously. Naturally soon after, many follow-up works appeared that tried to solve previously unsolvable problems using the new framework. One such problem was image-to-image translation, i.e. given an image from one domain (like dogs) transform it to another domain (like cats) while preserving the structure of the input image.
In this assignment, you will get hands-on experience implementing and training GANs. The assignment is divided into two sections. In the first section, you will train a vanilla generative convolutional GAN. In the second part, you are given a dataset from two different domains. One consists of emojis used in apple devices and another consists of emojis used in Windows. You will use CycleGAN [2]to implement an image-to-image model on this dataset.
This assignment imitates a similar assignment from univerity of toronto's CS321.

Part 1: Deep convolutional GAN (DCGAN) [40 points]

This class of GANs are models that use fully convolutional models both in the generator and in the discriminator. The discriminator consists of stacks of convolutional layers that map the input image to a single number, The probability of the image being real or fake, While the generator consists of stacks of transpose convolutional layers that map a random noise sample to an image. Figure 1

In this part, you need to complete the implementation of the discriminator, the generator, and the training loop. for implementing the discriminator and generator

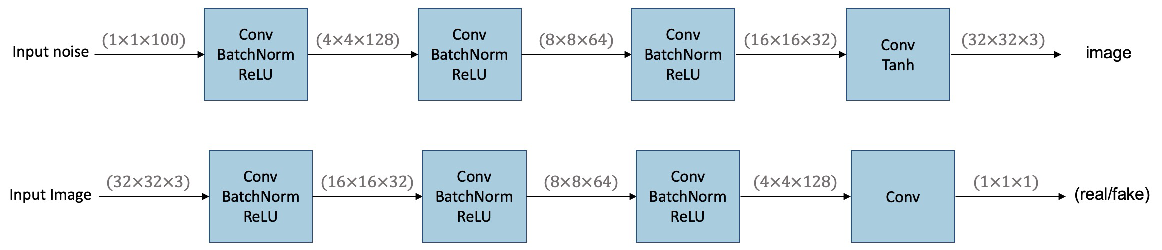

Figure 1: Architecture of the Generator (top) and Discriminator (bottom) in DCGAN

you should complete the **__init__** and **forward** functions of the **Discriminator** and **Generator** classes in the models.py file. For the training loop, you should fill in the **TODO** parts in the train_vanillagan.py file. In this file, you should implement the losses, as well as helper functions to evaluate and save the model.

To complete the training script you should:

1. complete the **train_step** function. This function handles the training iterations. To implement this function you should follow Algorithm 1.
2. complete the **eval_step** function. In this function, you should sample a number of images using your generator and save them.
3. complete the **save_step** function. In this function, you should save the weights of both the discriminator and the generator.

Part 2: CycleGAN [60 points]

So far we have trained a GAN network to generate samples from a domain (e.g. Apple emojis) from random noise vectors. Now we turn to use the same framework to solve a more complex problem, image-to-image translation. The problem statement is as follows:

We are given samples of images from two different domains, for example, emojis from Apple and Windows, and the samples from the two sets aren't ordered. This means that the *i*th sample from the Apple emojis does not necessarily correspond to the *i*th sample from the Windows emojis. Can we design a framework that given a sample from one of the sets, outputs a corresponding sample from the other set?

Algorithm 1 GAN Training Loop Pseudocode

```
1: procedure TrainGAN
2:     Draw $m$ training examples $\{x^{(1)}, \dots, x^{(m)}\}$ from the data distribution $p_{data}$
3:     Draw $m$ noise samples $\{z^{(1)}, \dots, z^{(m)}\}$ from the noise distribution $p_z$
4:     Generate fake images from the noise: $G(z^{(i)})$ for $i \in \{1, \dots .m\}$
5:     Compute the (least-squares) discriminator loss:
           $J^{(D)} = \frac{1}{2m}\sum_{i=1}^{m}\left[\left(D(x^{(i)}) - 1\right)^2\right] + \frac{1}{2m}\sum_{i=1}^{m}\left[\left(D(G(z^{(i)}))\right)^2\right]$
6:     Update the parameters of the discriminator
7:     Draw $m$ new noise samples $\{z^{(1)}, \dots, z^{(m)}\}$ from the noise distribution $p_z$
8:     Generate fake images from the noise: $G(z^{(i)})$ for $i \in \{1, \dots .m\}$
9:     Compute the (least-squares) generator loss:
           $J^{(G)} = \frac{1}{m}\sum_{i=1}^{m}\left[\left(D(G(z^{(i)})) - 1\right)^2\right]$
10:    Update the parameters of the generator
```

A possible solution to this problem is to have two generators, one that takes Apple emojis as input and outputs Windows emojis ($G_{A\rightarrow W}$) and another that takes Windows emojis as input and outputs Apple emojis ($G_{W\rightarrow A}$). Additionally, we can have two discriminators. D_A that judges apple emojis and tells us whether an image is an Apple emoji and a similar one for the Windows emojis D_W. The training process would be then similar to training a vanilla GAN, i.e. we train the discriminators and generators with an adversarial objective. The only difference would be that the generators map images from one domain to another instead of mapping random noise vectors to an image domain.

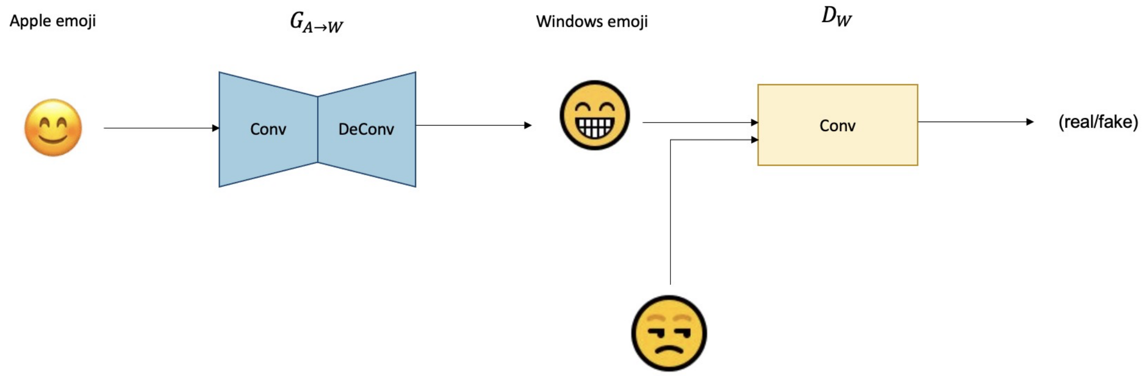

Figure 2: Adversarial training for image-to-image translation

However, the problem with this framework is that there is no guarantee that when mapping image A to image W the two actually correspond to each other. The brilliant idea of CycleGAN is that in order to keep the structure of the input when changing it to another domain, we introduce an additional loss function called a **cycle consistency loss**. The intuition behind this loss is that if we change image A to image W using $G_{A \to W}$, and then change W back to A using $G_W \to A$, then the result A' should be similar to A. and we can compute the similarity as simple as computing an L_2 loss between A and A'.

To complete this section you should first complete the **CycleGenerator** in the models.py file by implementing its **__init__** and **forward** functions. The architecture is as follows:

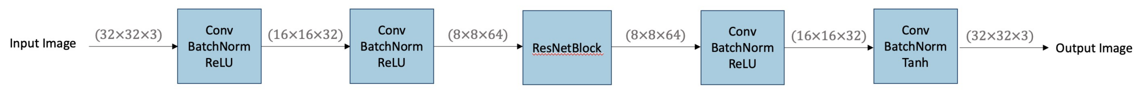

Figure 3: Architecture of the CycleGAN generator

Furthermore, you should complete the training script in the train_cyclegan.py file. For this similar to the training script of the vanilla gan you should do the following:

1. complete the **train_step** function. This function handles the training iterations. To implement this function you should follow Algorithm 2. Notice that the whole training process is similar to the adversarial training you implemented for the vanilla GAN. Here you just have two discriminators and generators and also you should compute the cycle consistency loss between a reconstructed image X' and a real image X as $||X - X'||^2$.

2. complete the **eval_step** function. In this function, You should sample a number of images from the Apple test set and a number of images from the Windows test set and compute their translations and save the outputs.

3. Complete the **save_step** function. In this function, you should save the weights of the discriminators and the generators.

Algorithm 2 CycleGAN Training Loop Pseudocode

```
1: procedure TrainCycleGAN
2:     Draw a minibatch of samples $\{x^{(1)}, \dots, x^{(m)}\}$ from domain $X$
3:     Draw a minibatch of samples $\{y^{(1)}, \dots, y^{(m)}\}$ from domain $Y$
4:     Compute the discriminator loss on real images:
           $$\mathcal{J}^{(D)}_{real} = \frac{1}{m}\sum_{i=1}^{m}(D_X(x^{(i)}) - 1)^2 + \frac{1}{n}\sum_{j=1}^{n}(D_Y(y^{(j)} - 1)^2$$
5:     Compute the discriminator loss on fake images:
           $$\mathcal{J}^{(D)}_{fake} = \frac{1}{m}\sum_{i=1}^{m}(D_Y(G_{X\to Y}(x^{(i)})))^2 + \frac{1}{n}\sum_{j=1}^{n}(D_X(G_{Y\to X}(y^{(j)})))^2$$
6:     Update the discriminators
7:     Compute the $Y \to X$ generator loss:
           $$\mathcal{J}^{(G_{Y\to X})} = \frac{1}{n}\sum_{j=1}^{n}(D_X(G_{Y\to X}(y^{(j)})) - 1)^2 + \mathcal{J}^{(Y\to X\to Y)}_{cycle}$$
8:     Compute the $X \to Y$ generator loss:
           $$\mathcal{J}^{(G_{X\to Y})} = \frac{1}{m}\sum_{i=1}^{m}(D_Y(G_{X\to Y}(x^{(i)})) - 1)^2 + \mathcal{J}^{(X\to Y\to X)}_{cycle}$$
9:     Update the generators
```

Extra details

1. The script for loading data is provided in the dataloader.py script.
2. The data,model and training configs are all stored as classes in the options.py file. There is a class for configs of the VanillaGAN model and another class for the CycleGAN model. You can use instances of these classes and pass them to the data, model or training classes. However, if you do not fill comfortable with this setting, please fill free to change your config handling to argparse, yaml or any system of your choosing.
3. To be able to train the models with the resources that we have, we have chosen smaller networks and also the data is too small. Therefore your results do not have to be perfect. They just have to make sense visually.
4. The final grading will be based on:
    - your code and implementation.

- the results. the demo sessions and your ability to answer questions about the framework.

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