# BME 646/ ECE695DL: Homework 7

Varun Aggarwal

April 11, 2022

## 1 Introduction

This assignment was aimed at implementing Generative Adversarial Networks (GAN) network logic. In addition, training of GAN had to be demonstrated.

## 2 Methodology

This assignment was well defined. Majority of the training code came from DLStudio V2.0.6.My contribution was designing a network that works both with WGAN training loop and DCGAN training loop.

List of tasks are as follows:

1. Creating and training DCGAN inspired model
2. Creating and training WGAN inspired model

## Task 1

There were various design choices to make for DCGAN model. The discriminator from HW4 was used at first. Although, I was unable to design a reliable generator. It worked only for some pseudo random seed (12134). To overcome this limitation, I started identifying the difference in my network and the one available in DLStudio. I found that major difference is the presence of fully connected layers in the discriminator. From thereon, I decided to review literature to find out GAN network which utilize both convolution and fully connected layers. I stumbled upon a Fully Connected and Convolutional Net Architecture (FCC-GAN) [1]. Inspired by the network, I decided to implement. The network still failed to converge. I made certain modifications to the network, like, including extra convolution layer. This addition helped in increasing the input image size from 32x32 to 64x64.

Finally, the network converged. The network training graph and generated images are given in Figure 1 and Figure 2.

## Task 2

For WGAN, I did not make significant modifications to the network in Task 1. I replaced sigmoid in the last layer with mean. The model didn't generate the best results. These results of the training are given in Figure 3 and Figure 4.

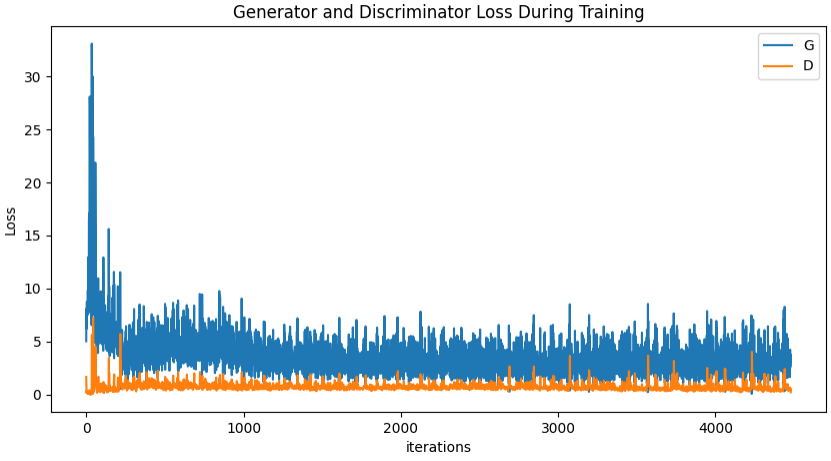


Figure 1: Loss curve for modified FCC-GAN


Figure 2: Generated images from FCC-GAN

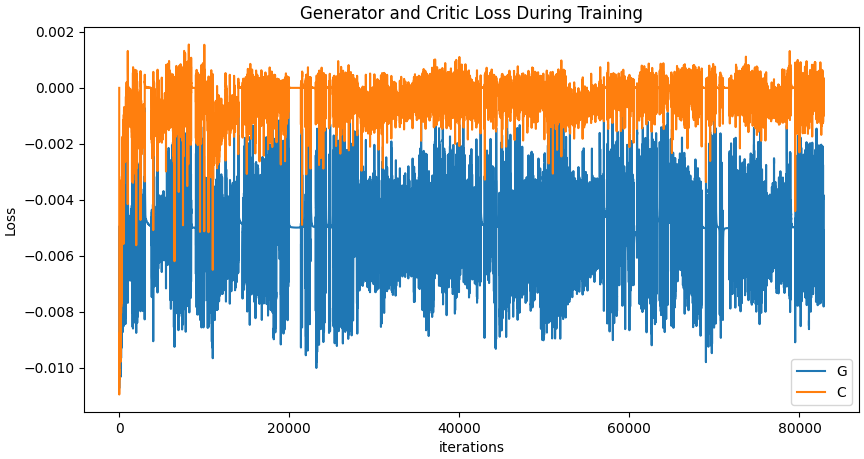


Figure 3: Loss curve for modified WGAN


Figure 4: Generated images from WGAN

# 3 Implementation

**CODE-hw07_training.py**

```
"""
Homework 7: Create GAN network
Author: Varun Aggarwal
Last Modified: 11 Apr 2022
Modifed from DLStudioV2.1.6
"""

import random
import numpy
import torch
import os, sys

# # """
seed = 12134
# random.seed(seed)
# torch.manual_seed(seed)
# torch.cuda.manual_seed(seed)
# numpy.random.seed(seed)
# torch.backends.cudnn.deterministic=True
# torch.backends.cudnn.benchmarks=False
# os.environ['PYTHONHASHSEED'] = str(seed)
# # """

from modelsv2 import discrimination, generation
from modelsv4 import discrimination, generation
## watch -d -n 0.5 nvidia-smi

sys.path.append("/home/varun/work/courses/why2learn/hw/DLStudio-2.1.6")
from DLStudio import *
from AdversarialLearning import *

import sys

dls = DLStudio(
                dataroot = "/home/varun/work/courses/why2learn/hw/hw7/data/
                    ↪ train",
                image_size = [64,64],
                path_saved_model = "./runs/saved_model",
```

```
                learning_rate = 5e-5, ## <== try smaller value if mode
                    ↪ collapse
                epochs = 500,
                batch_size = 64,
                use_gpu = True,
            )

adversarial = AdversarialLearning(
                dlstudio = dls,
                ngpu = 1,
                latent_vector_size = 100,
                beta1 = 0.5, ## for the Adam optimizer
                clipping_threshold = 0.005, # remove this wgan
            )

dcgan = AdversarialLearning.DataModeling( dlstudio = dls, adversarial =
    ↪ adversarial )


# discriminator = dcgan.DiscriminatorDG1()
# generator = dcgan.GeneratorDG1()

discriminator = discrimination()
generator = generation()

num_learnable_params_disc = sum(p.numel() for p in discriminator.parameters
    ↪ () if p.requires_grad)
print("\n\nThe␣number␣of␣learnable␣parameters␣in␣the␣Discriminator:␣%d\n" %
    ↪ num_learnable_params_disc)
num_learnable_params_gen = sum(p.numel() for p in generator.parameters() if
    ↪ p.requires_grad)
print("\nThe␣number␣of␣learnable␣parameters␣in␣the␣Generator:␣%d\n" %
    ↪ num_learnable_params_gen)
num_layers_disc = len(list(discriminator.parameters()))
print("\nThe␣number␣of␣layers␣in␣the␣discriminator:␣%d\n" % num_layers_disc)
num_layers_gen = len(list(generator.parameters()))
print("\nThe␣number␣of␣layers␣in␣the␣generator:␣%d\n\n" % num_layers_gen)

dcgan.set_dataloader()

# dcgan.show_sample_images_from_dataset(dls)
```

```
dcgan.run_wgan_code(dls, adversarial, critic=discriminator, generator=
    ↪ generator, results_dir="/home/varun/work/courses/why2learn/hw/hw7/
    ↪ runsv4/results_DG1_face_v2-"+str(seed)+"-"+str(dls.learning_rate))
```

**CODE-model-fcc.py**

```
"""
Homework 4: Create Convolution Neural Network (CNN)
Author: Varun Aggarwal
Last Modified: 10 Feb 2022
"""

from tkinter import X
import torch
import torch.nn as nn
import torch.nn.functional as F

class discrimination(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(3, 32, kernel_size=4, stride=2, padding=1)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=4, stride=2, padding=1)
        self.conv3 = nn.Conv2d(64, 128, kernel_size=4, stride=2, padding=1)
        self.conv4 = nn.Conv2d(128, 256, kernel_size=4, stride=2, padding=1)

        self.bn32 = nn.InstanceNorm2d(32, affine=True)
        self.bn64 = nn.InstanceNorm2d(64, affine=True)
        self.bn128 = nn.InstanceNorm2d(128, affine=True)
        self.bn256 = nn.InstanceNorm2d(256, affine=True)

        self.pool = nn.AvgPool2d(2, 2)
        self.sig = nn.Sigmoid()

        self.fc1 = nn.Linear(4*4*256, 512)
        self.fc2 = nn.Linear(512, 64)
        self.fc3 = nn.Linear(64, 16)
        self.fc4 = nn.Linear(16, 1)

    def forward(self, x):
        x = F.leaky_relu(self.bn32(self.conv1(x)), 0.2)
        x = F.leaky_relu(self.bn64(self.conv2(x)), 0.2)
        x = F.leaky_relu(self.bn128(self.conv3(x)), 0.2)
```

```
        x = F.leaky_relu(self.bn256(self.conv4(x)), 0.2)
        x = x.view(-1, 4*4*256)
        x = self.fc1(x)
        x = self.fc2(x)
        x = self.fc3(x)
        x = self.fc4(x)
        x = self.sig(x)
        # x = x.mean(0)
        # x = x.view(1)
        return x


class generation(nn.Module):
    def __init__(self):
        super().__init__()
        #self.task = task
        self.convTrans1 = nn.ConvTranspose2d( 256, 128, kernel_size=4,
            ↪ stride=2, padding=1)
        self.convTrans2 = nn.ConvTranspose2d( 128, 64, kernel_size=4, stride
            ↪ =2, padding=1)
        self.convTrans3 = nn.ConvTranspose2d( 64, 32, kernel_size=4, stride
            ↪ =2, padding=1)
        self.convTrans4 = nn.ConvTranspose2d( 32, 3, kernel_size=4, stride
            ↪ =2, padding=1)

        self.bn32 = nn.BatchNorm2d(32)
        self.bn64 = nn.BatchNorm2d(64)
        self.bn128 = nn.BatchNorm2d(128)

        self.tan = nn.Tanh()

        self.fc1 = nn.Linear(100, 64)
        self.fc2 = nn.Linear(64, 512)
        self.fc3 = nn.Linear(512, 4096)


    def forward(self, x):
        # print(x.shape)
        x = x.view(-1, 100)
        x = self.fc1(x)
        x = self.fc2(x)
        x = self.fc3(x)
```

```
        x = torch.reshape(x, (x.shape[0],256,4,4))
        x = F.relu(self.bn128(self.convTrans1(x)))
        x = F.relu(self.bn64(self.convTrans2(x)))
        x = F.relu(self.bn32(self.convTrans3(x)))
        x = self.convTrans4(x)
        x = self.tan(x)

        return x
```

**CODE-model-wgan.py**

```
from tkinter import X
import torch
import torch.nn as nn
import torch.nn.functional as F

class discrimination(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(3, 32, kernel_size=4, stride=2, padding=1)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=4, stride=2, padding=1)
        self.conv3 = nn.Conv2d(64, 128, kernel_size=4, stride=2, padding=1)
        self.conv4 = nn.Conv2d(128, 256, kernel_size=4, stride=2, padding=1)

        self.bn32 = nn.InstanceNorm2d(32, affine=True)
        self.bn64 = nn.InstanceNorm2d(64, affine=True)
        self.bn128 = nn.InstanceNorm2d(128, affine=True)
        self.bn256 = nn.InstanceNorm2d(256, affine=True)

        self.pool = nn.AvgPool2d(2, 2)
        self.sig = nn.Sigmoid()

        self.fc1 = nn.Linear(4*4*256, 512)
        self.fc2 = nn.Linear(512, 64)
        self.fc3 = nn.Linear(64, 16)
        self.fc4 = nn.Linear(16, 1)

    def forward(self, x):
        x = F.leaky_relu(self.bn32(self.conv1(x)), 0.2)
        x = F.leaky_relu(self.bn64(self.conv2(x)), 0.2)
        x = F.leaky_relu(self.bn128(self.conv3(x)), 0.2)
        x = F.leaky_relu(self.bn256(self.conv4(x)), 0.2)
        x = x.view(-1, 4*4*256)
```

```
        x = self.fc1(x)
        x = self.fc2(x)
        x = self.fc3(x)
        x = self.fc4(x)
        # x = self.sig(x)
        x = x.mean(0)
        x = x.view(1)
        return x


class generation(nn.Module):
    def __init__(self):
        super().__init__()
        #self.task = task
        self.convTrans1 = nn.ConvTranspose2d( 256, 128, kernel_size=4,
            ↪ stride=2, padding=1)
        self.convTrans2 = nn.ConvTranspose2d( 128, 64, kernel_size=4, stride
            ↪ =2, padding=1)
        self.convTrans3 = nn.ConvTranspose2d( 64, 32, kernel_size=4, stride
            ↪ =2, padding=1)
        self.convTrans4 = nn.ConvTranspose2d( 32, 3, kernel_size=4, stride
            ↪ =2, padding=1)

        self.bn32 = nn.BatchNorm2d(32)
        self.bn64 = nn.BatchNorm2d(64)
        self.bn128 = nn.BatchNorm2d(128)

        self.tan = nn.Tanh()

        self.fc1 = nn.Linear(100, 64)
        self.fc2 = nn.Linear(64, 512)
        self.fc3 = nn.Linear(512, 4096)


    def forward(self, x):
        # print(x.shape)
        x = x.view(-1, 100)
        x = self.fc1(x)
        x = self.fc2(x)
        x = self.fc3(x)
        x = torch.reshape(x, (x.shape[0],256,4,4))
        x = F.relu(self.bn128(self.convTrans1(x)))
```

```
x = F.relu(self.bn64(self.convTrans2(x)))
x = F.relu(self.bn32(self.convTrans3(x)))
x = self.convTrans4(x)
x = self.tan(x)

return x
```

# 4 Lessons Learned

The programming homework was straightforward and covered the basics of GAN implementation for object detection. Although, the programming took a while to complete due to model collapse.

# 5 Suggested Enhancements

No enhancements for this homework. Very well designed !

# References

[1] Sukarna Barua, Sarah Monazam Erfani, and James Bailey. Fcc-gan: A fully connected and convolutional net architecture for gans. *arXiv preprint arXiv:1905.02417*, 2019.