Project 3

Deep Learning in Computer Vision

Authors

Group 4

Daniel Juhász Vigild - s161749

Lau Johansson - s164512

Frederik Kromann Hansen - s161800

June 24, 2020

1 Project purpose and datasets

The purpose of this project is to explore the possibilities of GANs on two problems: creating fake hand-written digits and converting pictures of horses to zebras and vice versa.

Hand-written digits: The GANs are trained on the MNIST dataset, here with 938 pictures used for training. We first implement a simple vanilla GAN architecture consisting of fully connected networks.
Horse-to-zebra converting: We implement a LS-CycleGAN to convert pictures of horses to zebras and vice versa. The dataset consists of pictures from two classes: zebras and horses. The training set has 1067 images containing horses and 1334 images of zebras. The test set has 120 horses and 140 zebras. To balance the dataset in order to not generalize to zebras, we only select the same number of images from the zebra set as is in the horse set.
Hand-written digits (revisited): Finally, we train a convolutional LSGAN discriminator and generator on the MNIST-digit problem and compare the performance with the vanilla GAN presented earlier.

2 Simple GAN on MNIST dataset

2.1 Simple vanilla GAN arcitecture

The generator is a fully connected neural network with one hidden layer and 5000 neurons. Between each layer are ReLU activations. The discriminator is also a fully connected neural network with one hidden layer and 5000 neurons. Between each layer are LeakyReLU activations (alpha 0.2) and dropout 0.3 to avoid sparse gradients. During training Adam optimizer is used with a relatively high learning rate of 0.02 on both the generator and discriminator. The loss function is binary cross entropy.

2.2 Training the network

The model is fed with batches of 64 images of images from MNIST, which is passed through the discriminator to evaluate if the images are true or generated. Then, an

image of random noise is passed through the generator which in turn manipulates this in the direction of a digit in order to fool the discriminator. This image is passed to the discriminator the same way as the initial image to be evaluated for authenticity. Both the discriminator and the generator learn based on the performance of the discriminator. In any case, the generator will learn what features were significant in the discriminator's decisions and change its behaviour accordingly. Simultaneously, the discriminator improves in distinguishing real from fake images. The training procedure is illustrated below:

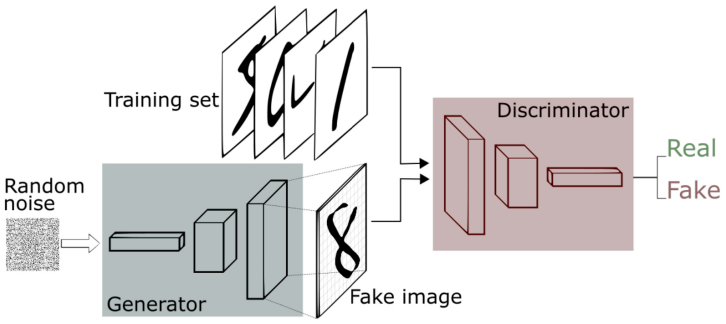

Figure 1: MNIST GAN training

2.3 Results

The network has been trained for 30 epochs and fixed noise is passed to the generator in every epoch. Figure 2 shows one image from each epoch and illustrates how the network improves. A lot of the images do not look like real digits, but e.g. the two images in the middle look like a "9" and a "1".

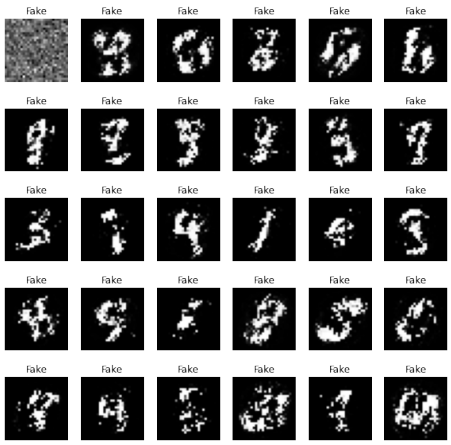

Figure 2: Fake digits generated. Showing one image pr. epoch from upper left corner in reading direction down to lower right corner.

3 CycleGAN - horses and zebras

3.1 Implementation CycleGAN

Conceptual idea Our implementation of CycleGAN follows the conceptual idea in the figure below:

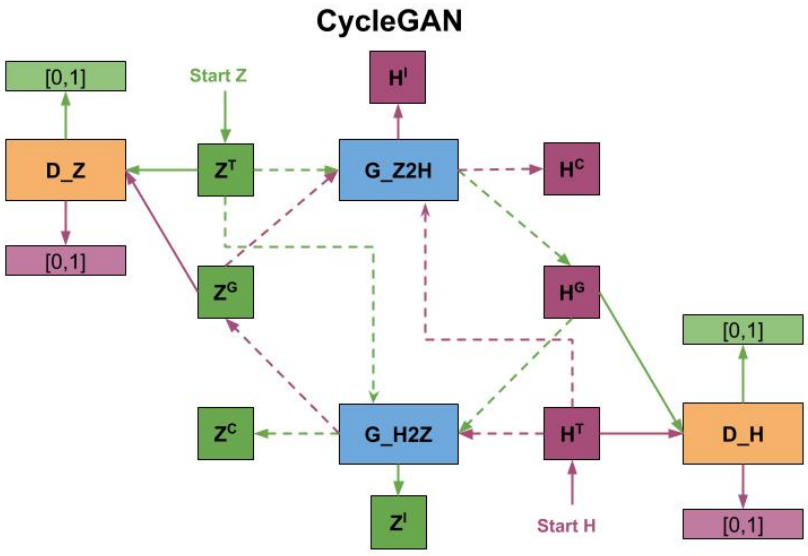

Figure 3: CycleGAN architecture

The models ability to learn is driven by the "fight" between the generators (blue) and discriminators (orange). They have different and conflicting purposes, but letting them compete against one another will make each of them better at their specific task. The generators' tasks are to convert images of horses to zebras and vice versa. The discriminators' task is to tell whether an input picture is generated or genuine. The CycleGAN uses a least squares loss.

Evaluation The generators are tested on three parameters: **1)** fool discriminator: the degree to which it can fool the discriminator, that is make it believe that a fake image is real, **2)** cycle consistency: how close a real image (Z^T/H^T) is to a reconstructed image (Z^C/H^C), that is an image that has been converted from - for example - horse-to-zebra (Z^G) and then from zebra-to-horse (H^C), **3)** identity loss: how close a horse put through a zebra-to-horse generator (H^I) is to the original horse image (and vice versa), meaning that an image belonging to a class put through a generator that converts to that same class, shouldn't be changed. The discriminators

are tested on two parameters: **1)** their ability to classify fake images as fake, **2)** their ability to classify real images as real.

3.2 Results

Training the CycleGan model for 15 epochs with 9 ResBlocks in the generators. By visually inspection of Figure 4 the horse is quite well transformed into a zebra (actually both of the horses). The generator find it more difficult to transform the zebras into horses.

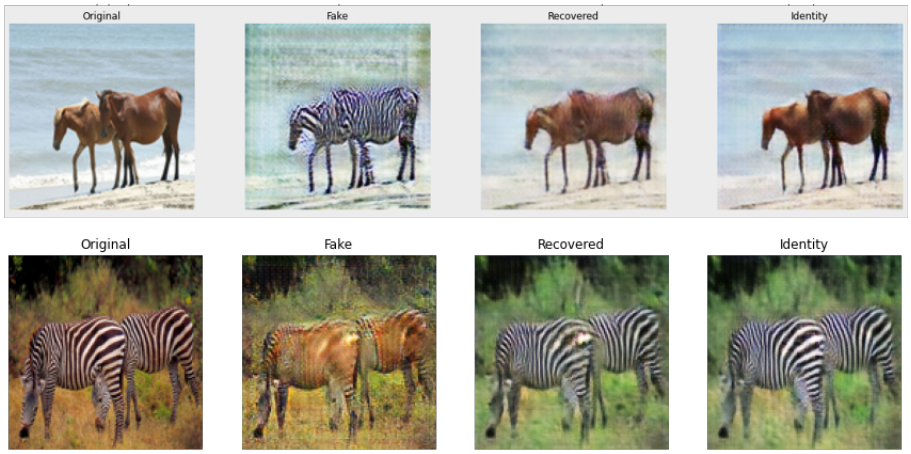

Figure 4: Plot of horses and zebras (Original, fake, recovered and identity)

Frechet Inception Distance (FID) is used to evaluate the performance of the Horse2Zebra CycleGAN model. FID calculates distance between learned features and scores how similar the compared images are. A low FID score indicates similar images. The table shows FID scores between the original image and the fake, recovered and identity images.

Class	**Fake**	**Recovered**	**Identity**
Horses	268.82	172.11	136.58
Zebras	204.98	84.91	63.42

Table 1: Frechet Inception Distance between original image and fake, recovered, identity.

The fake images have the highest FID scores. This is a expected results since it is the generators attempt to convert the real horse to a zebra (and vice versa). A lower FID score for the recovered image is very good, because the network should try to convert the fake horse back to the original horse. Having a score of 172.11 (horse) and 84.91 (zebra) it seems that the network could still be improved since a perfect FID score is 0. FID score of the identity images has a lower score than both

the fake and recovered images. This is good, since passing a real horse image to the zebra-to-horse generator, should ideally not change the image.

4 MNIST revised

To test our convolutional generator and discriminator (using least square loss) on a different domain - MNIST data - we revised the GAN model, and substituted the fully-connected generator and discriminator with the ones used in CycleGAN. This resulted in a successful implementation where the created digits are visibly better than the ones created by the simple GAN, see figure below where we show fake digits created from the same random noise, but with generators updated twice for each epoch:

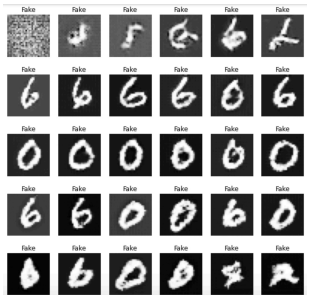

Figure 5: Plot of fake digits (LSGAN)

In the figure (2 pictures pr. epoch, 15 epochs total), it can be seen that the generators output gets better and better for the first 9 epochs or so, and then seems to get a bit worse. But the digits are in general much better than the ones created by the simple GAN. We also interpolated between two different image classes (created from two different random noises):

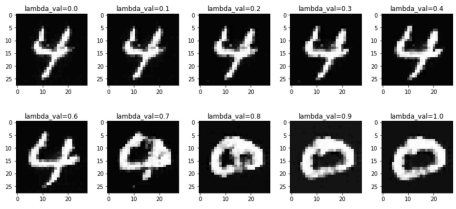

Figure 6: Interpolation between digit 4 and 0.

Here we see how the GAN moves gradually from a 4 to a 0, making the straight lines of the 4 rounder and rounder until taking the form of a 0.

Appendix

Diary for project 3

We began coding up the CycleGAN in concert while screen sharing, and took turns trying to make it work in alternating fashion until we managed to make it work.

While doing so, Lau implemented the simple GAN on the MNIST data.

Daniel took this implementation, and changed the discriminator and generator so they could be used on the MNIST problem.

Finally, Lau and Frederik managed (with help from TAs and Morten) to make the CycleGAN on zebra-to-horse (and vice versa) output some visually pleasing results.

Discriminator

```
class Discriminator(nn.Module):
    def __init__(self, f=64):
        super(Discriminator, self).__init__()

        self.conv = nn.Sequential(
            nn.Conv2d(3,f,4,stride=2,padding=1), #1 layer
            nn.InstanceNorm2d(f),
            nn.LeakyReLU(alpha),
            nn.Conv2d(f,f*2,4,stride=2,padding=1), #2 layer
            nn.InstanceNorm2d(f*2),
            nn.LeakyReLU(alpha),
            nn.Conv2d(f*2,f*4,4,stride=2,padding=1), #3 layer
            nn.InstanceNorm2d(f*4),
            nn.LeakyReLU(alpha),
            nn.Conv2d(f*4,f*8,4,stride=1,padding=1), #4 layer
            nn.InstanceNorm2d(f*8),
            nn.LeakyReLU(alpha),
            nn.Conv2d(f*8,1,4,stride=1,padding=1), #5 layer
        )

    def forward(self, x):
        return self.conv(x)
```

Loss functions

```
def LSGAN_loss_Dt(input_1):
    return 0.5*torch.mean((input_1 - b)**2)

def LSGAN_loss_Df(input_2):
    return 0.5* torch.mean((input_2 - a)**2)

def LSGAN_loss_G(input_1):
  return 0.5*torch.mean((input_1 - c)**2)

def Imloss(pred,target):
  crit=nn.L1Loss()
```

```
  return crit(pred,target)
```

Train

```
def train(A2B,B2A,Dis_A,Dis_B, opt_dis_A,opt_dis_B,opt_A2B,opt_B2A, epochs, data_tr, data_val):
  X_val, Y_val = next(iter(data_val))

  for epoch in range(epochs):
    print('* Epoch %d/%d' % (epoch+1, epochs))

    #Scale from [0:1] -> [-1:1]
    X_val = X_val*2-1
    Y_val = Y_val*2-1

    A2B.train()
    B2A.train()
    Dis_A.train()
    Dis_B.train()

    ######## TRAINING ########
    for X_batch, Y_batch in data_tr:
      X_batch = X_batch.to(device)
      Y_batch = Y_batch.to(device)

      #Scale from [0:1] -> [-1:1]
      X_batch = X_batch*2-1
      Y_batch = Y_batch*2-1


      # set parameter gradients to zero
      opt_A2B.zero_grad()
      opt_B2A.zero_grad()
      opt_dis_A.zero_grad()
      opt_dis_B.zero_grad()


      #Initialize (from slide 8 i lecture 3.2)

      #real images = X/Y batch
      a_real=X_batch
      b_real=Y_batch

      #Relates to horse
      b_fake = A2B(a_real)
      a_rec = B2A(b_fake)

      #Relates to zebra
      a_fake = B2A(b_real)
      b_rec = A2B(a_fake)


      ######PROCESS 1#######

      ####Discriminator ####
      output1=Dis_A(a_real) #Pass real Horse to discriminator A...
      loss1=LSGAN_loss_Dt(output1)

      output2=Dis_B(b_real) #Pass real Zebra to discriminator B...
      loss2=LSGAN_loss_Dt(output2)


      output3=Dis_A(a_fake.detach()) #Pass fake Horse (generated by B2A) to discriminator A...
      loss3=LSGAN_loss_Df(output3)


      output4=Dis_B(b_fake.detach()) #Pass fake Zebra (generated by A2B) to discriminator B...
      loss4=LSGAN_loss_Df(output4)
```

```
loss_dis_A_total=loss1+loss3 #Add all losses for discriminator A
loss_dis_A_total.backward()
opt_dis_A.step() #Update Discriminator A

loss_dis_B_total=loss2+loss4 #Add all losses for discriminator B
loss_dis_B_total.backward()
opt_dis_B.step() #Update Discriminator B


####GENERATOR ####
##FOOL discriminator
output5=Dis_B(b_fake) #Pass fake Zebra (generated by A2B) to discriminator B...
loss5=LSGAN_loss_G(output5)


output6=Dis_A(a_fake) #Pass fake Horse (generated by B2A) to discriminator A...
loss6=LSGAN_loss_G(output6)


##Cycle consistency
loss7=Imloss(b_rec,b_real) #Calculate loss between reconstructed and real Zebra
loss8=Imloss(a_rec,a_real) #Calculate loss between reconstructed and real Horse


##Identity loss
loss9=Imloss(B2A(a_real),a_real) #Calculate loss between real Horse and when real Horse is passed to B2A
loss10=Imloss(A2B(b_real),b_real) #Calculate loss between real Zebra and when real Zebra is passed to A2B


#Add all generator losses
loss_total=loss5+loss6+lambda_cycle*loss7+lambda_cycle*loss8+lambda_identity*loss9+lambda_identity*loss10
loss_total.backward()
opt_A2B.step() #Update generator A2B
opt_B2A.step() #Update generator B2A
```