Universiteit Leiden
The Netherlands

Opleiding Informatica

Generating CT scan images of the heart using Generative adversarial networks

Roy Timman (s2304716)

Supervisors:
Daan Pelt & Mary Chris Go

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl

28/05/2025

Abstract

Generative adversarial networks are deep neural networks that can be trained to produce synthetic images. In this thesis we will evaluate 3 models (GAN, WGAN and DCGAN) in their ability to generate CT scan images of the heart. Furthermore we will experiment with their hyperparameters and evaluate how changing these parameters affect model performance. The hyperparameters tested are learning rate (which controls the rate of which the model learns), the latent dimension (which controls how much noise is introduced in the generator) and batch size (which controls how many images the model generates before updating the model). We will analyze the results using both visual inspection and their Fréchet Inception Distance (FID) scores. FID is a commonly used evaluation metric for analyzing the quality of generated images. Our results show that DCGAN produces the most realistic CT scan images of the heart. However, all 3 models suffer from specific flaws that cause them to not succeed in generating realistic CT scan images of the heart. Additionally, our results show that changing the hyperparameters learning rate, latent dimension or batch size affect the performance of all 3 models. But how significantly as well as whether they positively or negatively impact performance, is model specific.

Contents

1 Introduction

Over the last decade, the medical world increasingly utilized artificial intelligence to assist in medical tasks. An example of this is the use of image classification algorithms in CT scans to assist healthcare professionals in detecting anomalies. The problem, however, lies in the fact that in order for the image classification algorithm to work precisely, a large training dataset is needed. Frid-Adar et al [FADK+18] proposed a possible solution to this by expanding the trainings dataset with synthetic augmented medical images. In their paper, they increased the performance of Liver Lesion Classification by using a generative adversarial network (GAN) to create synthetic medical images of liver lesions.

A generative adversarial network [GPAM+14] uses two models that compete against each other, the discriminator model that trains to distinguish real images from synthetic images and the generator model that trains to deceive the discriminator by generating realistic synthetic images from random noise.

In this thesis we will evaluate the performance of 3 generative adversarial networks, specifically GAN [GPAM+14], WGAN [ACB17] and DCGAN [RMC15] in generating synthetic CT scan images of the heart. We will also experiment with the hyperparameters learning rate, latent space dimension and batch size of all 3 GANs and compare their performances.

The dataset we will be using for training is the public heart CT scan dataset from Tafforeau et al. [TWW+21]. This dataset was obtained using Hierarchical Phase-Contrast Tomography (HiP-CT), an imaging technique developed by the authors of the dataset. HiP-CT is able to produce high resolution 3D images of human organs using synchrotron radiation. We chose this dataset due to the high quality of the images, which is perfect for our aim to generate realistic synthetic heart CT scan images. We will be using high resolution 2D slices from the dataset and convert them into grayscaled 256x256 images before feeding them in our GAN models. This conversion helps making training our GANS models less computationally expensive.

The focus of this thesis is to answer the following research question:

- **How do generated CT scan images of the heart using adversarial generative networks compare to real CT scan images of the heart?**

In order to answer this research question, we will introduce the following sub-research questions:

- **How do FID scores compare to the visual quality of generated images based on human visual assessment?**
- **Which specific generative adversarial network is able to generate to most realistic CT-scan images of the heart?**
- **How does changing the learning rate affect the performance of the generative adversarial networks?**

- **How does changing the latent space dimension affect the performance of the generative adversarial networks?**
- **How does changing the batch size affect the performance of the generative adversarial networks?**

2 Background

2.1 Generative adversarial network (GAN)

A Generative adversarial network is a network where 2 models compete against each other using adversarial training. These two models are the generator and the discriminator. The generator is tasked with generating a realistic fake sample that fools the discriminator into believing that it is a real sample. The discriminator is tasked to determine if a given sample is real or a fake sample. Generative adversarial network was first introduced in the paper by Goodfellow et al [GPAM+14]. In their paper, they describe in detail how both models work and how they learn from each other. For the generator, the input of the model consists of a random noise vector that gets passed into the network that produces, in our case, an image. The produced output will then be passed to the discriminator model, which will evaluate it. The input of the discriminator is, in our case, either an generated image or a real image. This input get passed through the network and produces a single output value ranging from 0 to 1. 1 meaning that the discriminator predicts the sample is real and 0 meaning that the sample is fake.

Both models play a minimax game with the following value function:

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{\text{data}}(x)}[\log D(x)] + \mathbb{E}_{z\sim p_z(z)}[\log(1 - D(G(z)))] \tag{1}$$

Here:

- x : Sample from the real dataset.
- z: Random noise sample.
- $G(z)$: Generated fake sample, given a random noise sample z as input.
- $D(x)$:The output of the discriminator given a real sample x as input.
- $D(G(z))$: The output of the discriminator given the generated fake sample $G(z)$.
- $\mathbb{E}_{x\sim p_{\text{data}}(x)}[\log D(x)]$: expected value of $\log D(x)$ given sample x from the data distribution $p_{\text{data}}(x)$ (the dataset).
- $\mathbb{E}_{z\sim p_z(z)}[\log(1 - D(G(z)))]$: expected value of $\log(1 - D(G(z)))$ given random noise sample z from some distribution $p_z(z)$.

The goal of the generator is to minimize the value function $V(D, G)$ by getting the discriminator output $D(G(z))$ to 1 as close as possible, making $\log(1 - D(G(z)))$ as negative as possible, thus decreasing the value of $V(D, G)$.
The goal of the discriminator is to maximize the value function $V(D, G)$ by getting $D(x)$ close to one as possible and $D(G(z))$ close to zero as possible, making both $\log D(x)$ and $\log(1 - D(G(z)))$ as close to zero as possible ($\log(1) = 0$), thus maximizing the value of $V(D, G)$.

From the value function $V(D, G)$ we can deduce the loss functions of both the discriminator model and generator model. For the discriminator, the loss function is:

$$\mathcal{L}_D = -\left(\mathbb{E}_{x\sim p_{\text{data}}(x)}[\log D(x)] + \mathbb{E}_{z\sim p_z(z)}[\log(1 - D(G(z)))]\right) \tag{2}$$

and for the generator, the loss function is:

$$\mathcal{L}_G = -\mathbb{E}_{z\sim p_z(z)}[\log(D(G(z)))] \tag{3}$$

The negative sign in both loss functions is used to make both loss functions output a value ≥ 0. Thus making it possible to utilize gradient descent to minimize both loss functions.

2.2 WGAN

Wasserstein GAN is a GAN variant that makes use of a different method to evaluate the discriminator and generator compared to the traditional GAN. This is done by using a different method to calculate the distance between the real distribution (the real dataset) and the generator's distribution. Before looking into WGAN, we will look into how the traditional GAN calculates the distance between two distributions.

GAN uses Jensen-Shannon divergence to calculate the distance between distributions which in the WGAN paper [ACB17] is defined as:

$$\text{JS}(\mathbb{P}_r, \mathbb{P}_g) = \frac{1}{2}\text{KL}(\mathbb{P}_r \| \mathbb{P}_m) + \frac{1}{2}\text{KL}(\mathbb{P}_g \| \mathbb{P}_m) \tag{4}$$

Where:

- $\mathbb{P}_r$: the real distribution
- $\mathbb{P}_g$: the generated distribution
- $\mathbb{P}_m = \frac{1}{2}(\mathbb{P}_r + \mathbb{P}_g)$: A combined distribution of the real and generated distribution.
- $\text{KL}(\mathbb{P}_r \| \mathbb{P}_m)$: the Kullback-Leibler divergence is a function to measure the difference between two probability distributions.

According to the paper [ACB17], GAN's discriminator can become too good at distinguishing real from fake samples that due to Jensen-Shannon divergence this will lead to vanishing gradients, meaning the generator is not able to learn anymore. Another flaw according to the WGAN paper and GAN paper [ACB17] [GPAM+14] is that the generator can be trained too much before the discriminator gets trained causing the generator to exploit the discriminator weakness. The weakness being the discriminator having trouble distinguishing a small subset of the dataset's real samples from generated fake samples. As a result the generator will mostly focus on generating those specific samples which will cause the generator to collapse and generate less diverse samples. This is called mode collapse (Helvetica scenario).

WGAN aims to fix these issues by using the Wasserstein distance (also known as Earth mover distance) to calculate the distance between two distributions [ACB17]. The Wasserstein distance is defined as:

$$W(\mathbb{P}_r, \mathbb{P}_\theta) = \sup_{\|f\|_L \leq 1} \mathbb{E}_{x \sim \mathbb{P}_r}[f(x)] - \mathbb{E}_{x \sim \mathbb{P}_\theta}[f(x)] \tag{5}$$

where:

- $\|f\|_L \leq 1$: the function being 1-Lipschitz meaning that the change of outputs can never be greater than the difference between their corresponding inputs ($|f(x_2) - f(x_1)| \leq |x_2 - x_1|$)
- sup: supremum, meaning the highest possible value in the range of possible values. in the case of the Wasserstein distance, it picks the largest distance over all possible distribution functions, which are 1-Lipschitz.

According to the WGAN paper [ACB17] the advantage of using the Wasserstein distance is that it is continuous and thus differentiable everywhere, meaning that the gradients can be calculated everywhere and thus always provide useful gradients.
Because the Wasserstein distance is over all 1-Lipschitz functions, the weights cannot be changed too much. the authors of the WGAN paper [ACB17] aim to solve this by introducing weight clipping, in which the weights values are forced to stay with a small domain of values. if in a weight update a weight exceeds this domain, it will be forced to be changed to the nearest value of clipped domain. The authors of the WGAN paper do say that weight clipping is not an ideal way of enforcing the 1-Lipschitz property.

2.3 DCGAN

A deep convolutional generative adversarial network (DCGAN) [RMC15] uses convolutional layers instead of fully connected layers in their architecture to learn to generate images. Commonly used in image classifications networks, convolutional layers [DV18] take in an input feature map and apply a convolutional operation on it. This operation can best be visualized by a window called the kernel sliding over the image or feature map and calculating the feature output mapping. The kernel contains weights that perform a dot product between the overlapping area of the input feature map and outputs a single element that gets put in the output feature map after that it slides and performs the same operation to the next overlapping area. This is will repeat itself until the kernel has slid

over the whole input feature map and returns a full output feature map. This output can then be used to perform another convolution operation on if it gets passed on to the next convolutional layer.

In the DCGAN architecture, the disriminator [RMC15] uses strided convolutions in its convolutional layers. These strided convolutions are used to let the discriminator learn to reduce resolution of the image by decreasing the width and height of the image. In the DCGAN paper they call it "spatial downsampling". For the generator the the convolutional layers perform fractionally strided convolutions, which are used to let the generator learn to increase the resolution. In the DCGAN paper [RMC15] they call it "spatial upsampling".

Lastly, DCGAN makes use of the same value function and loss functions as GAN.

2.4 FID

Fréchet inception distance (FID), introduced in the paper "GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium" from Heusel et al. [HRU+17], is a commonly used metric to analyze the quality of the generated images compared to the dataset. In the paper they use a pretrained inception-v3 model [SVI+16] and extract the feature maps from an intermediate layer for every generated image and real image passed through the inception model. According to the paper, these extracted feature maps contain "vision-relevant features" of the images and are assumed to follow a multidimensional Gaussian distribution, also known as a joint normal distribution. After the feature maps have been extracted the means and covariance matrices of both generated images as well as the real images gets computed. The FID is then obtained by calculating the Fréchet distance between the real and generated feature map distributions using these computed means and covariances matrices. This Fréchet distance is defined as [DL82][HRU+17]:

$$d^2\left((\mu_g, \Sigma_g), (\mu_r, \Sigma_r)\right) = \|\mu_g - \mu_r\|_2^2 + \operatorname{Tr}\left(\Sigma_g + \Sigma_r - 2\left(\Sigma_g \Sigma_r\right)^{1/2}\right) \tag{6}$$

Where:

- μ_g: mean of the feature maps from generated samples.
- μ_r: mean of the feature maps from the real samples.
- Σ_g: Covariance matrix of the feature maps from generated samples.
- Σ_r: Covariance matrix of the feature maps from the real samples.
- $\|\mu_g - \mu_r\|_2^2$: squared euclidean distance between the means of the feature maps.
- Tr: trace, an operation that returns the sum of the elements from the main diagonal of the matrix. For example $\operatorname{Tr}\left(\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}\right) = a_{11} + a_{22} + a_{33}$

The lower the FID scores, the closer the generated images are to the ground truth (the dataset), thus the higher the quality of the generated image.

3 Methodology

To evaluate the performance of the three GANs (GAN, WGAN and DCGAN) we propose a series of experiments that experiment with the hyperparameters: learning rate, latent dimension and batch size. The hyperparameters were selected because they directly impact training and all 3 GANs share them.

3.1 Learning rate

Learning rate aims to control the rate in which the discriminator and generator learn. It controls how much the weights in the models change per update. A relatively small learning rate is stable but takes more time to converge. A relatively large learning rate may learn faster but may cause instability, due to heavily fluctuating weights, an example of this is shown in Figure 1. Our aim is to find out if a relatively higher learning rate (two times the default learning rate) or relatively lower learning rate (half the default learning rate) improves model performance.

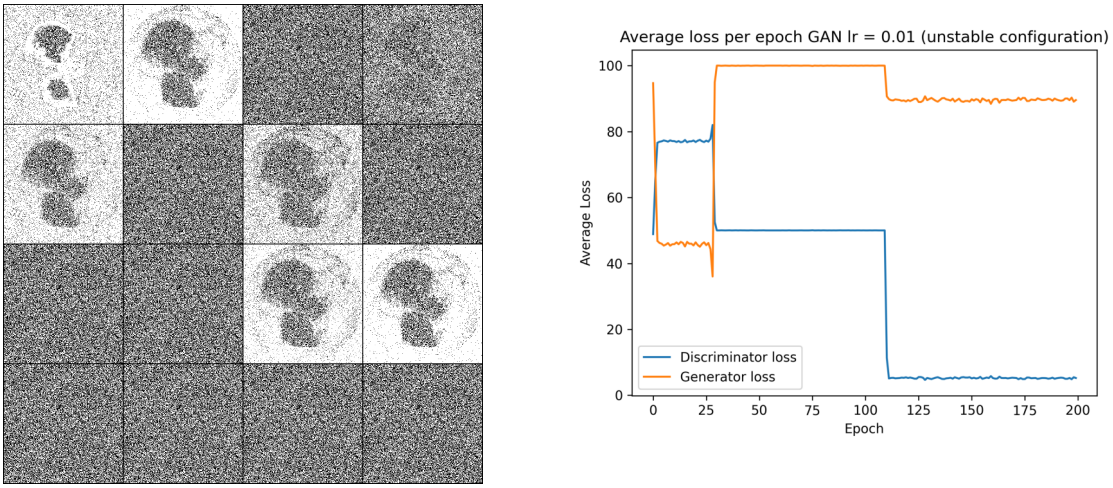


Figure 1: Results of training a GAN using a too large learning rate (learning rate = 0.01), causing instability. Left figure: an image showing 16 generated samples that have been generated after a training of 200 epochs, right figure: A loss graph showing the average loss per epoch of the discriminator and the generator. The losses fluctuate heavily during the early stages and later stabilize to high average generator losses and low average discriminator losses. thus showing the model's instability and collapse.

3.2 Latent dimension

Latent dimension describes the size of the latent space. In Generative adversarial networks [GPAM+14], latent space represents the random noise input vector from the generator. This could mean that by increasing the latent dimension size, more noise can be introduced, thus more diverse data can be generated. Our aim is to find out if increasing or decreasing the latent dimension size will affect a GAN's performance. We will test this by training All GANS with their default latent dimension size (usually 100), half their default latent dimension size, and two times their latent dimension size.

3.3 Batch size

Batch size describes the numbers of samples being generated and evaluated per iteration. By increasing the batch size, more images per iteration gets generated and evaluated before the model gets updated. Our aim is to find out if increasing or decreasing the batch size affects A GAN's performance.

3.4 Experimental design

For each of the GANs we will perform 7 experiments, which will each be ran once for 200 epochs. Experiment 1 will act as the control experiment where we use the default configuration. These default configurations are either based on the original GAN papers [ACB17], [RMC15] or the code we based of our experiments on, which is from the Erik Linder-Norén Pytorch-GAN repository [LN18]. In experiment 2 and 3 we will change the learning rate by decreasing it to half (experiment 2) and increasing it to 2 times the default learning (experiment 3) rate. For Experiment 4 and 5 the latent dimension will be changed to half the default latent dimension (experiment 4) and two times the default latent dimension (experiment 5). Lastly, for experiment 6 and 7 the batch sizes will be changed, where in experiment 6 the batch size will be decreased to half the default batch size and in experiment 7 the batch size will be increased to twice the default batch size. However for DCGAN The batch size experiments will be different, DCGAN's experiment 6 batch size will be a quarter of it's default configuration and DCGAN's experiment 7 batch size will be half of the default configuration. This is because the GPU hardware used to run these experiments (RTX 4070 laptop GPU), cannot handle handle a batch size of 128 due of insufficient GPU memory.
Below we will provide a list of all the experiment's configurations for each GAN:

GAN

- Experiment 1: Default: learning rate = 0.0002, latent dimension = 100, batch size = 64
- Experiment 2: learning rate = 0.0001, latent dimension = 100, batch size = 64
- Experiment 3: learning rate = 0.0004, latent dimension = 100, batch size = 64
- Experiment 4: learning rate = 0.0002, latent dimension = 50, batch size = 64
- Experiment 5: learning rate = 0.0002, latent dimension = 200, batch size = 64
- Experiment 6: learning rate = 0.0002, latent dimension = 100, batch size = 32
- Experiment 7: learning rate = 0.0002, latent dimension = 100, batch size = 128

WGAN

- Experiment 1: Default: learning rate = 0.00005, latent dimension = 100, batch size = 64
- Experiment 2: learning rate = 0.000025, latent dimension = 100, batch size = 64
- Experiment 3: learning rate = 0.0001, latent dimension = 100, batch size = 64
- Experiment 4: learning rate = 0.00005, latent dimension = 50, batch size = 64

- Experiment 5: learning rate = 0.00005, latent dimension = 200, batch size = 64
- Experiment 6: learning rate = 0.00005, latent dimension = 100, batch size = 32
- Experiment 7: learning rate = 0.00005, latent dimension = 100, batch size = 128

DCGAN

- Experiment 1: Default: learning rate = 0.0002, latent dimension = 100, batch size = 64
- Experiment 2: learning rate = 0.0001, latent dimension = 100, batch size = 64
- Experiment 3: learning rate = 0.0004, latent dimension = 100, batch size = 64
- Experiment 4: learning rate = 0.0002, latent dimension = 50, batch size = 64
- Experiment 5: learning rate = 0.0002, latent dimension = 200, batch size = 64
- Experiment 6: learning rate = 0.0002, latent dimension = 100, batch size = 16
- Experiment 7: learning rate = 0.0002, latent dimension = 100, batch size = 32

Other hyperparameters like beta values from the Adam optimiser, RMS optimizer and the clip value will stay on their default values. These values can be found the code on Github: https://github.com/royio3/Heart_CT_GAN. The code has been based of Erik Linder-Norén Pytorch-GAN repository code [LN18]. The dataset used for the discriminators are CT scan images used are a modified version of the public heart CT scan dataset of dataset of Tafforeau et al. [TWW+21]. We modified the jp2 images to 256x256 png images, to lower training time. in our code we also converted them to grayscale images

To analyze the results of our experiments, we will evaluate the Fréchet inception distances and visual inspect generated images from all experiments. The Fréchet inception distances (FID) will be computed using Seitzer's implementation [Sei20], this implementation utilizes a pretrained inceptionv3 network [SVI+16] trained on ImageNet images [DDS+09]. For this computation, 3000 images are generated from the evaluated model and get compared to the dataset containing 3307 images. This process will be repeated 3 times, using the same evaluated model, to observe how the FID will vary. The resulting mean FID as well as the standard deviation will be reported. The image results from all experiments will be presented in a grid of 16 generated samples and compared to a grid of 16 random samples drawn from the dataset.

3.5 GAN model architectures

In order to run these experiments all 3 model's architectures need to be defined. Below we provide a detailed description of how these architecture have been defined. For all 3 models architectures, we used Linder-Norén's implementation [LN18].

3.5.1 GAN architecture

The generator of the GAN architecture consists of an input layer of size latent dim, 4 hidden layers with respective sizes 128, 256, 512 and 1024 and an output layer of the same size as the image size. The first hidden layer only uses the activation function Leaky Relu with value 0.2. The other hidden layers include a batch normalization layer which is put before the Leaky Relu activation function. The output layer uses the Tanh activation function. The discriminator consists of an input layer of the same size as the image size, 2 hidden layers of respective sizes 512 and 256, and an output layer of size 1. The hidden layers utilize Leaky Relu with value 0.2 as their activation function and the output layer utilizes the sigmoid activation function. The Adam optimizer is used to update the weights of both the generator and discriminator.

3.5.2 WGAN architecture

Both the generator and discriminator utilize the same architecture as GAN. The only differences are the use of the RMS optimizer instead of the Adam optimizer and that the discriminator is trained 5 times before the generator gets updated.

3.5.3 DCGAN architecture

The DCGAN architecture's generator starts with an input layer of size latent dim, representing the noise vector. The input first goes through a fully connected layer the size of $128 \times \frac{H}{4} \times \frac{W}{4}$, H and W being the height and width of the image. The fully connecting laying serves as a the "project and reshape" layer [RMC15], converting the noise vector to a higher dimensional tensor, which can be viewed as 128 feature maps of size $\frac{H}{4} \times \frac{W}{4}$. After that, the tensor get put through a 2D batch normalization layer. Next, the 128 feature maps go through an upsample layer with a scaling factor of 2 using the nearest neighbor algorithm and a 2D convolutional layer of size 128 with a kernel of size 3, a stride of 1 and a padding layer wrapped around the feature maps of size 1. This results in a tensor of size $128 \times \frac{H}{2} \times \frac{W}{2}$. The combination of the upsampling and 2D convolutional layer serve as an alternative to the transposed convolution, deviating from the original DCGAN implementation from Radford et al. [RMC15]

While the author of this snippet of code (Erik Linder-Norén, [LN18]) does not explain why he used this instead of following the original DCGAN implementation of using transposed convolutions [RMC15]. There is however a paper that might explain why. The paper from Odena et al. [ODO16] explains that the use of fractionally strided convolutions or in our case transposed convolutions causes checkerboard pattern artifacts in generated images. To get rid of these artifacts, the paper proposed to alternative method by first resizing the tensor using nearest neighbor algorithm and then applying a convolution operation on the resized tensor.

After the convolution, a 2D batch normalization is applied followed by the activation function leaky Relu with a value of 0.2. Then the tensor goes through another upsample with a scale factor of 2 and a 2D convolutional layer using a kernel size 3, stride 1 and padding 1, this time, resulting in 64 feature maps of size $H \times W$. After that it goes through another 2D batchnormalisation layer followed by the activation function Leaky ReLu with value 0.2 applied to it. Lastly it goes through another 2D convolution layer. resulting in an tensor of size $C \times H \times W$, where C is the amount of

image channels, which in our case is 1, because our images are grayscaled. Finally the activation function TanH activation is applied. resulting in the generator's output.

The discriminator starts of with an input layer consisting a tensor representation of the image of size $C \times H \times W$. which goes through a sequence 4 convolution "blocks" which give the following resulting output feature maps:

- Block 1: $16 \times H/2 \times W/2$
- Block 2: $32 \times H/4 \times W/4$
- Block 3: $64 \times H/8 \times W/8$
- Block 4: $128 \times H/16 \times W/16$

These blocks downsample the input while increasing the amount of feature maps. Each convolution block consists a 2D convolutional layer, using a kernel of size 3, stride 2 and a padding layer of 1, followed by the activation function leaky ReLu with an value of 0.2, and a dropout layer with a probability of 0.25. After the dropout layer, a 2D batch normalization layer is applied except for the first block.
Lastly the tensor goes through an fully connected layer of output size 1, using the activation function Sigmoid to produce a value between zero and one serving as the discriminator's output.

The ADAM optimizer is used to update the weights.

4 Experiment results

4.1 FID Scores

Model	Configuration	FID Score
GAN	Default	269.07 ± 1.06
	lr = 0.0001	$\mathbf{244.37 \pm 0.25}$
	lr = 0.0004	259.44 ± 0.28
	ldim = 50	253.99 ± 0.56
	ldim = 200	262.43 ± 0.99
	bsize = 32	257.86 ± 0.93
	bsize = 128	276.69 ± 0.66
WGAN	Default	261.56 ± 1.17
	lr = 0.000025	276.18 ± 1.21
	lr = 0.0001	$\mathbf{252.65 \pm 2.30}$
	ldim = 50	318.75 ± 0.97
	ldim = 200	$\mathbf{253.06 \pm 1.17}$
	bsize = 32	298.14 ± 1.46
	bsize = 128	281.27 ± 0.84
DCGAN	Default	288.91 ± 1.39
	lr = 0.0001	219.83 ± 0.63
	lr = 0.0004	247.57 ± 0.84
	ldim = 50	261.62 ± 0.63
	ldim = 200	326.15 ± 0.77
	bsize = 16	$\mathbf{210.96 \pm 0.55}$
	bsize = 32	370.87 ± 0.74

Table 1: Mean and standard deviations of FID Scores for Different GAN Architectures and Hyperparameter settings. Each FID is computed 3 times for every evaluated model. With every computation, 3000 images get generated from the evaluated model and compared to 3307 images of the dataset using the FID implementation from Seitzer. [Sei20].

4.1.1 FID score comparison of traditional GAN experiments

Looking at the FID scores of GAN in Table 1, We see that out of all traditional GAN configurations, the configuration using a learning rate of 0.0001 produced the lowest FID score which may suggest that lowering the learning rate improves the model's performance. However, the configuration using a higher learning rate of 0.0004 did produce a lower FID score compared to the default configuration which used a learning rate of 0.0002. This implies that both slightly increasing and decreasing the learning rate leads to better model performance.

The FID scores of the latent dimension experiment in Table 1 show that a lower latent dimension of 50 resulted in a better model performance compared to the higher latent dim configurations. However a higher latent dimension of 200 did yield a better model performance compared to the default configuration, which uses a latent dim of 100. This may suggest that lowering and raising

the latent dimension parameter can improve GAN performance.

The batch size experiment FID scores in Table 1 show that the configuration using a lower batch size of 32 resulted in a lower FID score. A higher batch size of 128 produced a higher FID score compared to the default configuration which uses a batch size of 64. This may suggest that reducing the batch size can improve GAN performance.

Out of all GAN FID scores in Table 1, the configuration using a lower learning rate of 0.0001 produced the lowest FID score, which indicates the highest improvement of traditional GAN performance.

4.1.2 FID score comparison of WGAN experiments

Looking at the FID scores of WGAN in Table 1, the configuration using a higher learning rate of 0.0001 produced the lowest FID score. interestingly, the configuration using a lower learning rate yielded the worst FID score compared to the default (lr = 0.00005) and the higher learning rate (lr = 0.0001). This may suggest that a slight increase of the learning rate improve WGAN performance.

The latent dimension experiment FID scores in Table 1 show that a higher latent dimension of 200 produced the lowest FID score compared to the higher latent dimension configurations, suggesting that a higher latent dimension could enhance WGAN performance.

Looking at the batch size experiment FID scores in Table 1, neither lowering the batch size (32) nor increasing the batch size (128) lowered the FID. which may suggest increasing and decreasing the batch size of 64 does not improve WGAN performance.

Looking at all the WGAN FID scores in Table 1, the configuration using a learning rate of 0.0001 produced the lowest FID score. However, comparing it to the second lowest FID score, which used a latent dim of 200, shows an overlap of their standard deviations, which suggest that their difference is statistically insignificant. Therefore the configuration using a lower learning rate of 0.0001 and the configuration using a higher dimension of 200 can both be considered best performers of WGAN.

4.1.3 FID score comparison of DCGAN experiments

Looking at the DCGAN FID scores of the learning rate experiment in Table 1, the configuration using a lower learning rate of 0.0001 produced the lowest FID score compared to the configuration using a higher learning rate. However the configuration using a higher learning rate of 0.0004 did produce a lower FID score compared to the default configuration (lr = 0.0002). Nevertheless, the lower learning rate configuration produced the most significant decrease in FID score compared to the default. Which may suggest that both slightly increasing and slightly decreasing learning rate could lead to DCGAN performance improvement.

DCGAN's latent dimension experiment in Table 1 shows a lower latent dimension of 50 pro-

ducing the lowest FID score compared to the higher default(100) and even higher latent dimension configuration (200). The configuration using a higher latent dimension of 200 produced a worse FID score compared to the default (100). These results may suggest that a lower latent dimension could potentially increase DCGAN performance.

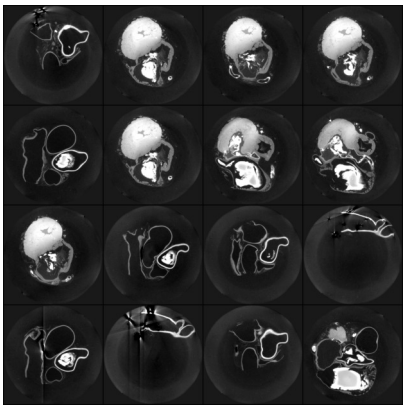
Looking at the DCGAN batch size experiment FID scores in Table 1, the configuration using a batch size of 16 produced the lowest FID score. Interestingly, the configuration using a slightly higher batch size but still smaller than the default configuration (64) produced a worse FID score. This may suggest that significantly decreasing could lead to better DCGAN performance while slight batch size decreases worsen DCGAN performance meaning batch size has a significant impact on DCGAN performance

Out of all DCGAN FID scores in Table 1, The configuration using a much smaller batch size of 16 produced the lowest FID score, closely followed by the configuration using a learning rate of 0.0004. However a batch size slightly higher than 16 but smaller than 64 produced the worst FID score.

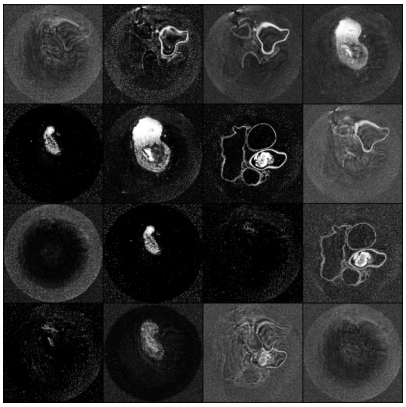
4.1.4 FID score comparison of all models

Looking at the FID scores of all models in Table 1, the DCGAN model configuration using batch size 16 produced the best result and thus could be considered the best performer out of all models. However, the configuration using a slightly higher batch size of 32 but lower than the default DCGAN batch size of 64 produced the worst FID score. This could suggest that changing the batch size in DCGAN has the most significant impact on model performance out of all models both negatively and positively.

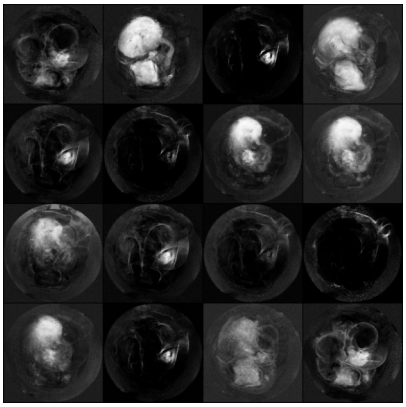
4.2 Image result best performers of every GAN model



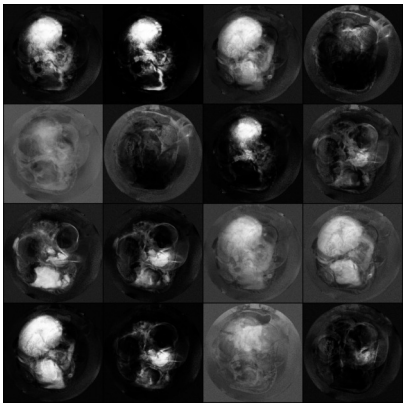
Dataset (ground truth)



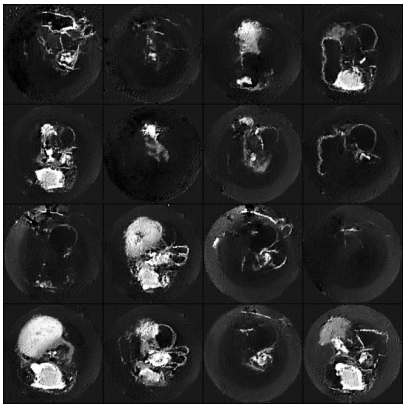
(a) GAN lr = 0.0001



(b) WGAN lr = 0.0001



(c) WGAN ldim = 200



(d) DCGAN bsize = 16

Figure 2: Image results shown in grid of 16 images of the best performers for each model according to their FID scores

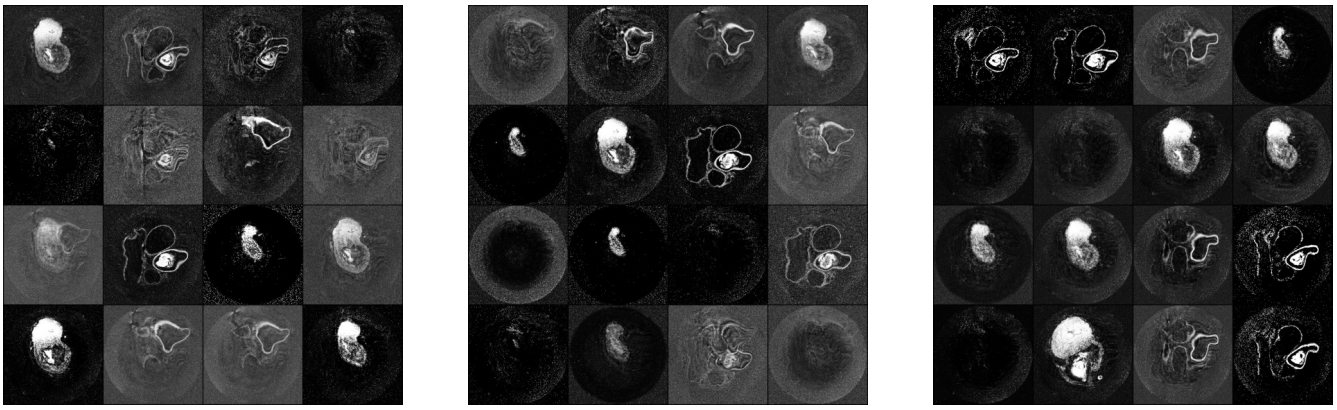
In Figure 2, noticeable differences can be observed comparing the generated images of 3 models, using the best configurations according to their FID scores, to the ground truth. In the GAN images, seen in Figure 2a, a significant amount of noise is present in the generated images. In contrast,

WGAN images, seen in Figure 2b and Figure 2c show no significant noise, however, the structures present in these images look blurry when compared to the ground truth. Looking at DCGAN generated images, seen in Figure 2d, checkerboard pattern artifacts can be seen even though no transposed convolutions have been used. This may suggest that the combination upsampling using nearest neighbor and convolution does not mitigate checkerboard artifacts, even though Odena et al. [ODO16] suggested otherwise.

Comparing the generated images, shown in Figure 2,to each other, WGAN appears to produce the highest resolution images while DCGAN produces images more closely resemble structures seen in the ground truth.

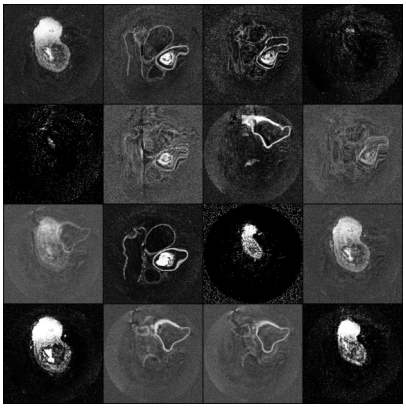
4.3 GAN image results

Below we will show the image results of our GAN experiments

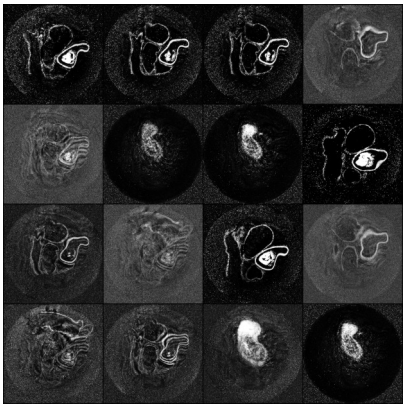


(a) GAN lr = 0.0002 (b) GAN lr = 0.0001 (c) GAN lr = 0.0004

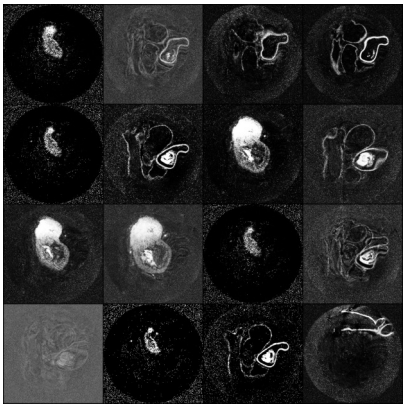
Figure 3: Image results shown in grid of 16 images of the GAN learning rate experiment where learning rate = 0.0002 is the default configuration.



(a) GAN latent dim = 100

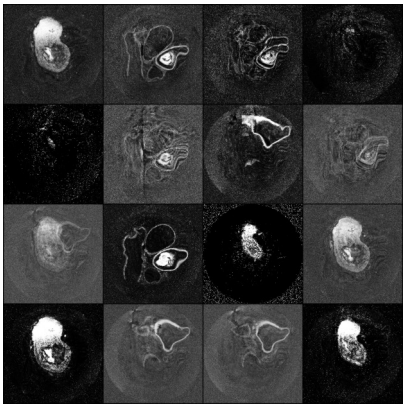


(b) GAN latent dim = 50

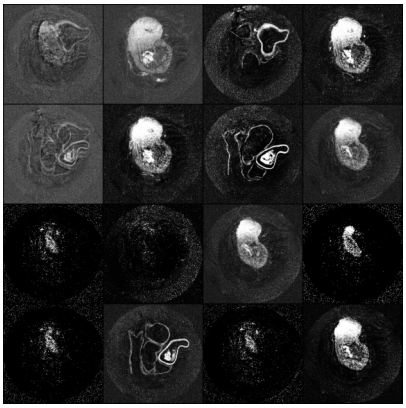


(c) GAN latent dim = 200

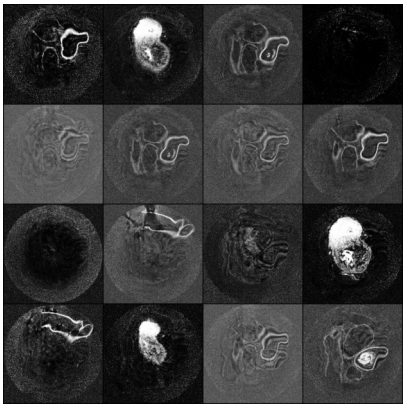
Figure 4: Image results shown in grid of 16 images of the GAN latent dimension experiment where latent dimension of 100 is the default configuration.



(a) GAN batch size = 64



(b) GAN batch size = 32



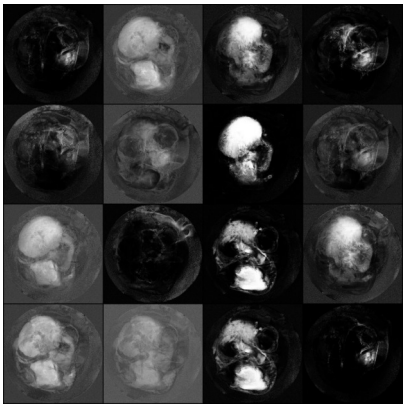
(c) GAN batch size = 128

Figure 5: Image results shown in grid of 16 images of the GAN batch size experiment where batch size of 64 is the default configuration.

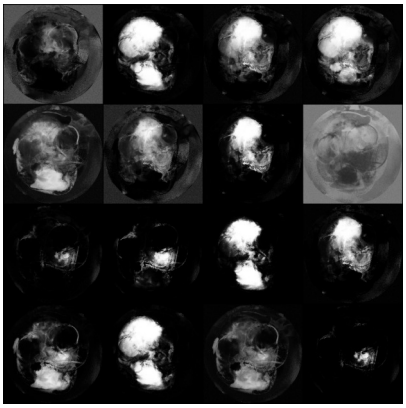
All GAN experiment image results, shown in Figure 3, 4 and 5 show a significant amount of noise in all generated images upon visual inspection. While no visual significant differences can be seen in quality, different structures in the generated images however can be seen but these can be due to the randomness in sampling. Additionally, The image results of the configuration using a batch size of 128 (Figure 5c) appear to show multiple repeating structures.

4.4 WGAN image results

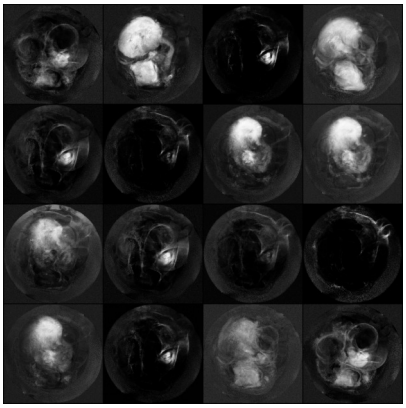
Below we will show the image results of our WGAN experiments



(a) WGAN lr = 0.00005

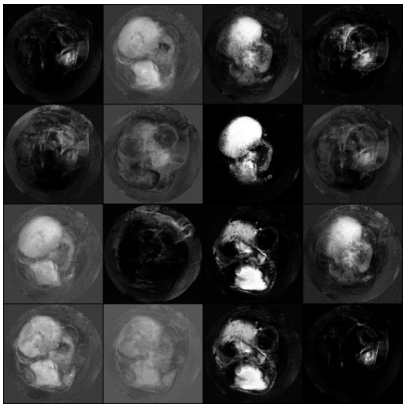


(b) WGAN lr = 0.000025

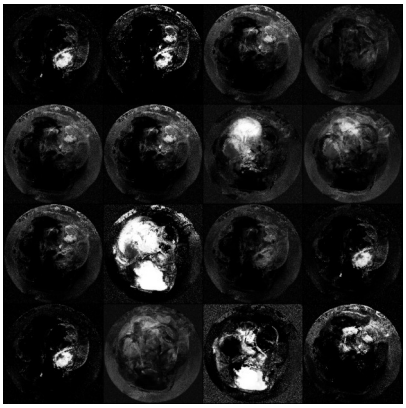


(c) WGAN lr = 0.0001

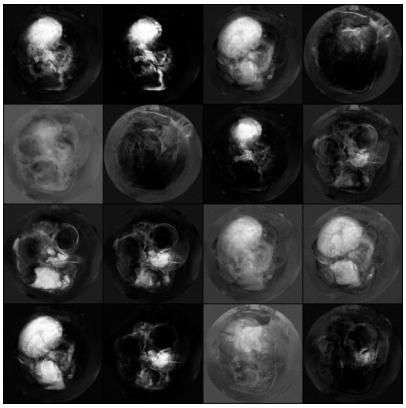
Figure 6: Image results shown in grid of 16 images of WGAN learning rate experiment where learning rate = 0.00005 is the default configuration.



(a) WGAN latent dim = 100

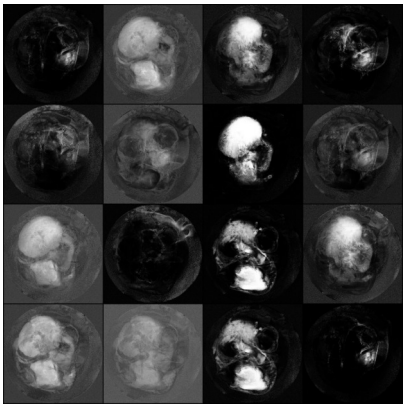


(b) WGAN latent dim = 50

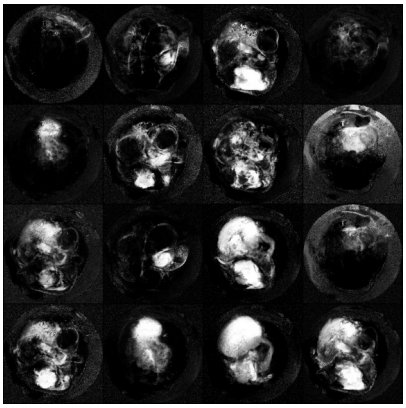


(c) WGAN latent dim = 200

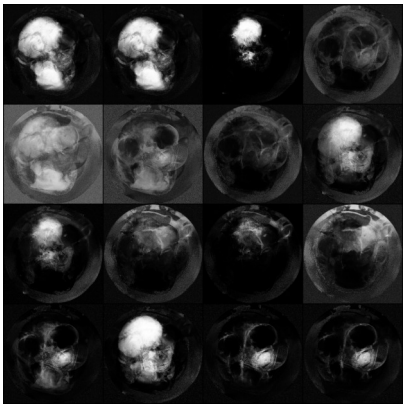
Figure 7: Image results shown in grid of 16 images of WGAN latent dimension experiment where latent dim of 100 is the default configuration.



(a) WGAN batch size = 64



(b) WGAN batch size = 32



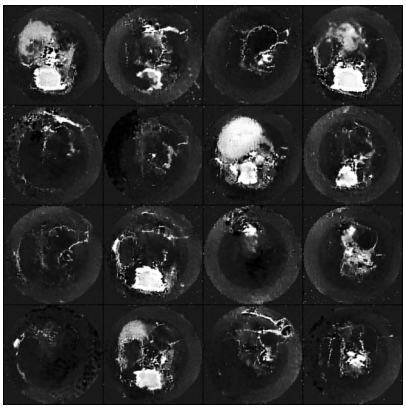
(c) WGAN batch size = 128

Figure 8: Image results shown in grid of 16 images of WGAN batch size experiment where batch size of 64 is the default configuration.

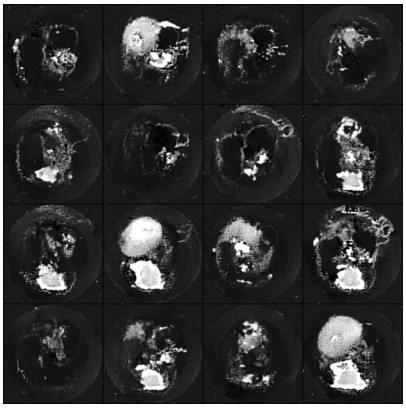
Visual inspection of the generated images of all WGAN experiments, shown in Figure 6,7 and 8 reveal blurry structures. Especially the configuration using a learning rate of 0.0001 (Figure 6c) and the configuration using a batch size of 128 (Figure 8c) appear to have produced multiple notable blurry structures. However these can be due to the randomness of sampling. Images generated by the configuration using a latent dimension of 50 (Figure 7b) appear to contain multiple structures not seen in the dataset, additionally the images generated by this configuration show a significant increase of noise compared to other WGAN configurations. An increase in noise is also seen in the configuration using a batchsize of 32 (Figure 8b).

4.5 DCGAN image results

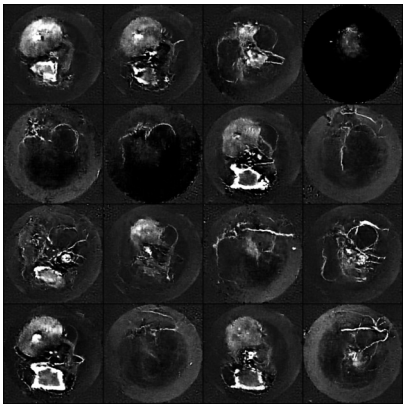
Below we will show the image results of our DCGAN experiments



(a) DCGAN lr = 0.0002

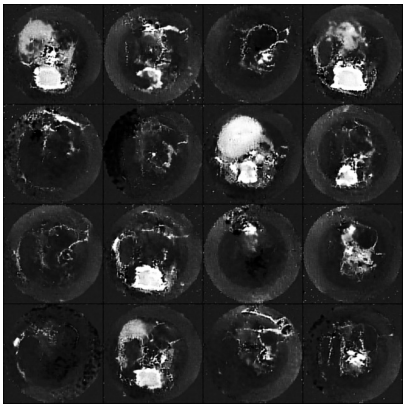


(b) DCGAN lr = 0.0001

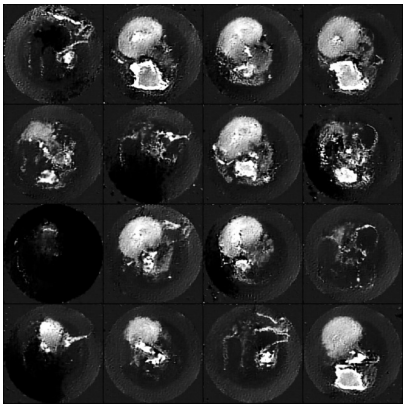


(c) DCGAN lr = 0.0004

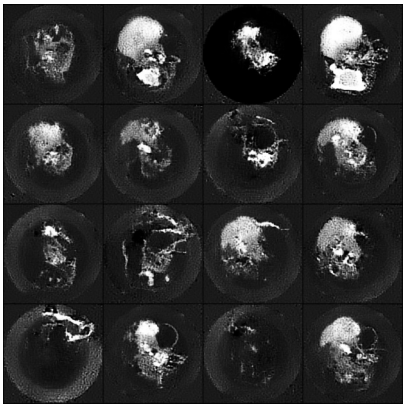
Figure 9: Image results shown in grid of 16 images of DCGAN learning rate experiment where learning rate = 0.0002 is the default configuration.



(a) DCGAN latent dim = 100

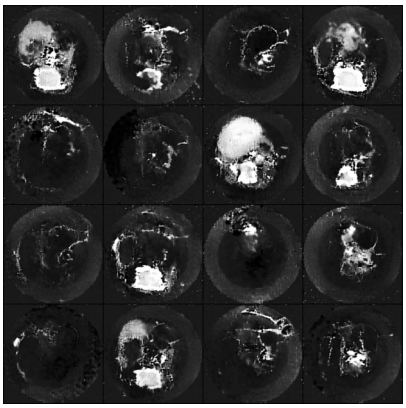


(b) DCGAN latent dim = 50

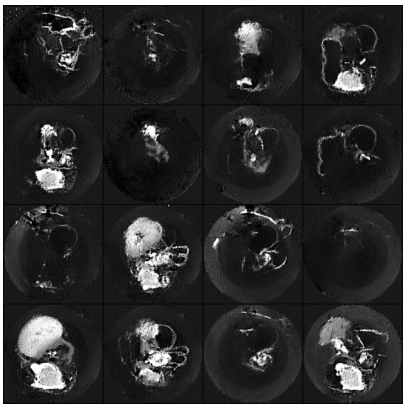


(c) DCGAN latent dim = 200

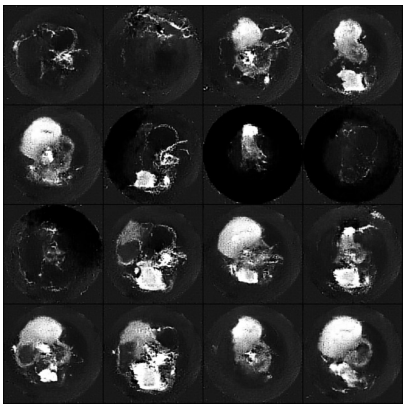
Figure 10: Image results shown in grid of 16 images of DCGAN latent dimension experiment where latent dimension of 100 is the default configuration.



(a) DCGAN batch size = 64



(b) DCGAN batch size = 16



(c) DCGAN batch size = 32

Figure 11: Image results shown in grid of 16 images of DCGAN batch size experiment where batch size of 64 is the default configuration.

The DCGAN experiment image results shown in Figure 9,10 and 11 all show checkerboard patterns artifacts upon close visual inspection. Most notably these artifacts can be seen in Figure 11c, which uses the configuration with a batch size of 32. However, excluding this result, image results of the other configuration show no visual significant differences in quality. Different structures in the generated images of all configurations can be seen but these can be due to the randomness in sampling.

5 Discussion and future research

In this thesis, we analyzed the results of our experiments using FID scores and visual inspections. Comparing both analyses, the FID score appeared mostly consistent with the visual inspections. A higher FID score produced worse image results and a lower FID score produced better results. However, it is necessary to point out that the differences in FID scores have to large enough in

order to be noticed in the image results. Also, the implementation used to compute these FID scores (Seitzer's implemtation [Sei20]) utilized a pretrained inception-v3 network [SVI+16], trained on ImageNet images [DDS+09] and thus the extracted features are based on natural images instead of CT scan images. It would have been better if the images used to train the inception-v3 network were medical CT scan images to get more accurate FID scores.

Our aim in this thesis is to analyze how changing the hyperparameters learning rate, latent dimension and batch size affect model performance. While our experiments did show changes of model performance. We only ran the experiments once instead running them multiple times, this is a limitation because GAN training is a stochastic process [GPAM+14] and thus every run can give varying results. Additionally The limited experiment layout of only 3 experiments per hyperparameter may not give a complete picture. Furthermore, the interactions between these hyperparameters have not been explored. For example we did not do an experiment where we changed more than one hyperparameter. Additionally other more model specific hyperparameters such as the clip value parameter from WGAN and the beta values from the ADAM optimizer, present in GAN and DCGAN have not been experimented with. Possible future research could be to investigate how these model specific hyperparameters affect GAN performance.

Only one network architecture per GAN model was used. Possible future research could be to investigate how different network configurations affect GAN performance. For example, how would increasing the amount of hidden layers affect model performance?

Lastly, only 3 GAN models were experimented with, due to them being less computationally expensive, compared to newer GAN models such as the progressive growing GAN [KALL18] and StyleGAN [KLA19]. Future research could be to experiment with these more recent, advanced models, compare their performance to DCGAN, WGAN, GAN and evaluate their ability to generate realistic samples.

6 Conclusions

In this thesis we evaluated 3 GAN models in their performance of producing realistic synthetic CT scan images of the heart. We used the dataset HiP-CT scan heart dataset from Tafforeau et al. [TWW+21] to train and analyze the networks. We compared real CT scan images and generated images by both visually inspecting the generated images and computing their Fréchet Inception Distances (FID). FID is a commonly used metric to analyze GAN performance [HRU+17]. Below we will answer the research questions:

How do FID scores compare to the visual quality of generated images based on human visual assessment? The FID scores mostly aligned with the visual inspection evaluation. However, the change of FID score has to be significantly high enough to be visible in the image results.

Which specific generative adversarial network is able to generate to most realistic CT-scan images of the heart? Out of the GAN, WGAN and DCGAN models, The DCGAN model generated the most realistic CT-scan images of the heart. However, DCGAN's image results

contained visible checkerboard pattern artifacts, which reduced the overall image quality.

How does changing the learning rate affect the performance of the generative adversarial networks? Experimenting with the learning rate showed that for traditional GAN and DCGAN, slightly increasing and decreasing the learning rate from 0.0002 to 0.0001 or 0.0004 improved performance. For WGAN slightly increasing the learning rate from 0.00005 to 0.0001 positively impacted WGAN performance, while decreasing the learning rate to 0.000025 decreased performance.

How does changing the latent space dimension affect the performance of the generative adversarial networks? Experimenting with the latent dimension showed that for traditional GAN, increasing and decreasing the latent dimension from 100 to 50 or 200 improved performance. For WGAN, increasing the latent dimension from 100 to 200 positively impacted WGAN performance, while decreasing the latent dim to 50 worsened the performance.
For DCGAN, decreasing the latent dimension from 100 to 50 improved performance, while increasing latent dim from 100 to 200 worsened the performance.

How does changing the batch size affect the performance of the generative adversarial networks? Experimenting with the batch size showed that for traditional GAN, a decreasing the batch size from 64 to 32 improved GAN performance, while increasing the batch size from 64 to 128 worsened GAN performance.
For WGAN, both reducing the batch size from 64 to 32 and increasing the batch size from 64 to 128 decreased WGAN performance.
For DCGAN, reducing the batch size slighty from 64 to 32 worsened performance, while further reducing the batch size to 16 significantly increased performance.

To asses the main research question: **How do generated CT scan images of the heart using adversarial generative networks compare to real CT scan images of the heart?** Our experiments of The GAN, WGAN and DCGAN models show that all 3 models were able to generate CT scan images of the heart. However all 3 models showed specific flaws: GAN generated images contained a significant amount of noise, while WGAN generated images showed blurry structures vaguely resembling structures seen in the dataset, DCGAN generated images showed visible checkerboard pattern artifacts. Concluding that none of the 3 models succeeded in generating realistic CT scan images of the heart.

References

[ACB17] M. Arjovsky, S. Chintala, and L. Bottou. Wasserstein generative adversarial networks. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 214–223. PMLR, 06–11 Aug 2017.

[DDS+09] J. Deng, W. Dong, R. Socher, L-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009.

[DL82] D.C Dowson and B.V Landau. The fréchet distance between multivariate normal distributions. *Journal of Multivariate Analysis*, 12(3):450–455, 1982.

[DV18] V. Dumoulin and F. Visin. A guide to convolution arithmetic for deep learning. *arXiv preprint arXiv:1603.07285v2*, 2018.

[FADK+18] M. Frid-Adar, I. Diamant, E. Klang, M. Amitai, J. Goldberger, and H. Greenspan. Gan-based synthetic medical image augmentation for increased cnn performance in liver lesion classification. *Neurocomputing*, 321:321–331, 2018.

[GPAM+14] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems*, volume 27, pages 2672–2680. Curran Associates, Inc., 2014.

[HRU+17] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 6629–6640. Curran Associates Inc., 2017.

[KALL18] T. Karras, T. Aila, S. Laine, and J. Lehtinen. Progressive growing of gans for improved quality, stability, and variation. In *6th International Conference on Learning Representations, ICLR 2018 - Conference Track Proceedings*, 2018.

[KLA19] T. Karras, S. Laine, and T. Aila. A style-based generator architecture for generative adversarial networks. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4396–4405, 2019.

[LN18] E. Linder-Norén. Pytorch-gan. https://github.com/eriklindernoren/PyTorch-GAN, 2018.

[ODO16] A. Odena, V. Dumoulin, and C. Olah. Deconvolution and checkerboard artifacts. *Distill*, 2016.

[RMC15] A. Radford, L. Metz, and S. Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

[Sei20] M. Seitzer. pytorch-fid: FID Score for PyTorch. https://github.com/mseitzer/pytorch-fid, August 2020. Version 0.3.0.

[SVI+16] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the inception architecture for computer vision. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2818–2826, 2016.

[TWW+21] P. Tafforeau, C. Walsh, W. L. Wagner, Daniyal J. Jafree, A. Bellier, C. Werlein, M. P. Kühnel, E. Boller, S. Walker-Samuel, J. L. Robertus, D. A. Long, J. Jacob, S. Marussi, Emmeline Brown, N. Holroyd, D. D. Jonigk, M. Ackermann, and P. D. Lee. Complete heart from the body donor LADAF-2020-27 (Version 1) [dataset]. https://doi.org/10.15151/ESRF-DC-572189991, 2021.