Denoising Cryo-Electron Tomograms Using Deep Learning

Master's Thesis

student	Douwe van Gijn
number	ICA-5830230
supervisor	Ilja Gubins
first examiner	Remco Veltkamp
second examiner	Friedrich Forster

Utrecht University

Abstract

Cryo-Electron Tomography (Cryo-ET) gives structural biologists the tools to inspect biological samples in situ at a near atomic resolution is 3D. The three dimensional volumes called 'tomograms' that this process produces suffer from a very low signal-to-noise ratio, and therefore need denoising. This thesis explores 5 neural networks in increasing complexity for their ability to denoise the 3D tomograms. The performance of the network was measured by using multiple metrics including the Structural Similarity (SSIM) score, where 0 is worst and 1 is best. All networks are able to transform noisy tomograms with a mean SSIM score of 0.258 to denoised volumes with a mean SSIM score of 0.937 for the worst network and 0.993 for the best network. The networks show that they generalize very well to unfamiliar particles, moderately well to different noise models, and poorly to multiple particles in a volume. Further research must conclude if the poor performance for the multiple particles is due to the change in scale.

Contents

1 Introduction

"In natural science the principles of truth ought to be confirmed by observation."
— Carl Linneaus, *Philosophia Botanica* (1751)

Observations from the living world around us form the foundation of biology, and better instruments allow for better observations.

The invention of optical microscopes in the 17th century lead to the discovery of living micro-organisms and cells. Now however, because the relatively large wavelength of visible light, objects smaller than 200 nanometer, around the length of the measles virus, cannot be resolved.

The smaller wavelengths of X-ray allowed for the discovery of the double-helix structure of DNA in the first half of the 20th century using X-ray crystallography [1]. X-ray crystallography can be used to reveal three dimensional spatial structure of specific macromolecules. While it produces high-quality images, it requires that a sample be crystallised, which is not possible for every macromolecule and requires that the sample is extracted and isolated from their environment.

For imaging macromolecules, Cryo-Electron Tomography(CET) is an alternative to X-ray crystallography. The advantage of CET over X-ray crystallography is that it allows for *in-situ* (in place) viewing of the sample, although it suffers from a low signal-to-noise ratio. Instead of using crystals, CET uses plunge-freezing to rapidly fixate samples as large as entire cells. The sample is then put in an ultra-high vacuum chamber between an electron gun and a detector to obtain high resolution projections of the sample. By making projections at sequentially changing angles we obtain tilt-series images. These tilt-series images can be reconstructed into a three dimensional volume called a tomogram (figure 1).

Various tomographic reconstruction algorithms can be used to solve an inverse problem and estimate the volumetric structure of the sample. This reconstruction requires a Contrast transfer function (CTF) correction to reverse-engineer the original three dimensional shape from the recorded images.

Finding the optimal correction is made more difficult because the tilt-series does not cover a full rotation and therefore has a *missing wedge*. This missing wedge is a major source of noise and counteracting it is an ill-posed problem called the *missing wedge problem*.

Apart from the systematic reconstruction artifacts mentioned above, the tomogram also contains noise. The major source of noise comes from the stochastic nature of electron scattering off the sample and onto a detector. The electron bombardment causes structural damage to the sample, so the electron dose of the sample must be limited when acquiring the tilt-series. However, less electrons hitting the detector means less spatial information, manifesting as noise in the tomogram.

After obtaining the tomogram from the tilt-series, edge-detection convolution is applied to make structures in the noisy tomogram more visible. Template matching is Next, template matching is used to match particles in a database with the particles in the tomogram. Finally, a diagram with the position and orientation of the matched particles is generated for structural biologists to interpret.

This way of working is time consuming and requires a lot of skilled labor. Innovation in CET is required to overcome these issues. This thesis will investigate the application of deep learning to denoise tomograms to help in the analysis of the tomograms.

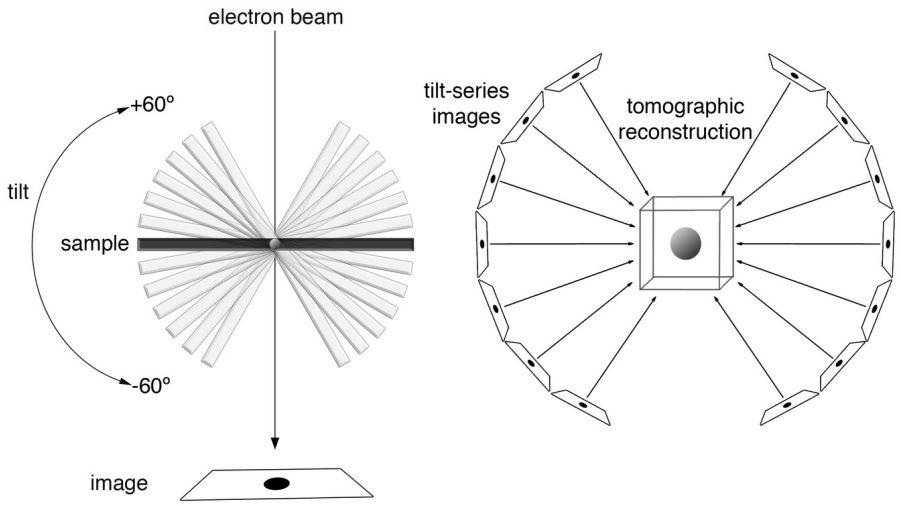

Figure 1: **left:** tilt-series images are acquired by putting the sample at various angles between an electron beam and a detector. **right:** the 2D tilt-series images can be reconstructed into a 3D tomogram.

2 Background

Machine learning is an umbrella term for all algorithms and techniques that allow the program to figure out how to solve a problem instead of the programmer. For the purpose of denoising tomograms, we are interested in the subset of machine learning called *supervised learning*. With supervised learning, we know what the correct answer should be, but we don't know how to compute the answer. For example, we have a noisy and noise-free image, and we want to turn the noisy image into a noise-free image, we know what the correct result should be, but not how to get there.

Supervised learning, where training set contains the correct answers, stands in contrast to unsupervised learning, where there is no training set with correct answers available. An example of unsupervised learning would be a machine that looks through a database that describes which user account watched which video and learns to predict which videos specific users are interested in.

Specifically, we are interested in deep learning, the branch of machine learning that uses neural networks.

2.1 Neural Networks

Neural networks are *universal function approximators*, as proven by Csáji[2] in 2001. This means that a sufficiently large and well-configured neural network can approximate any continuous function.

Neural networks are a network of neurons organized into layers. Each neuron receives signals from the previous layer, takes a weighted sum, and runs it through a fixed activation function (figure 2). By tweaking the weights of each neuron, different functions can be approximated.

In practice, networks have many layers with many neurons, each with many weighted connections. The number of weights in a network is typically very large: around 62.3 million in the case of AlexNet (section 2.3).

To solve the problem of tweaking all the weights, Rumelhart et al.[3] popularized *backpropagation* in 1988 . Backpropagation is a technique that automatically ad-

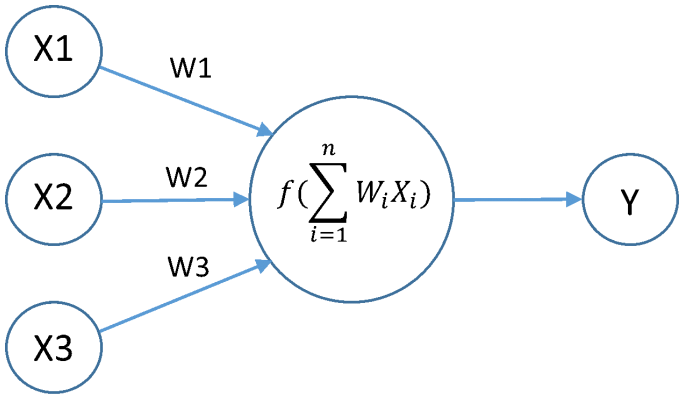

Figure 2: Each neuron sum their weighted inputs and apply an activation function to produce their output.

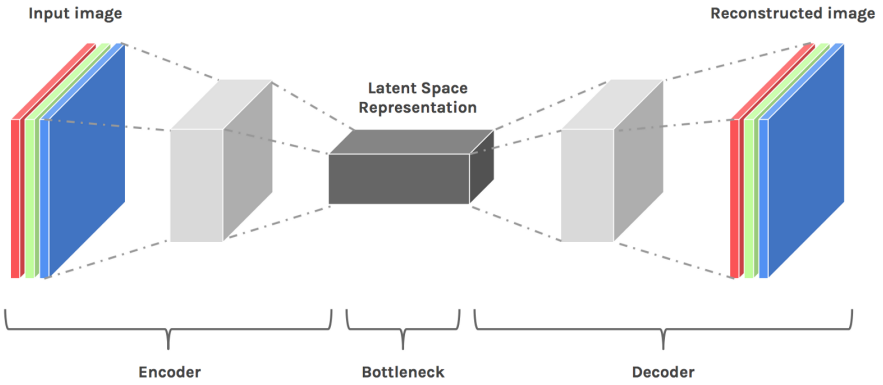

Figure 3: Auto-Encoder architecture

justs the weights from the last layer back to the first layer to minimize the error between what the network predicted and what the correct answer was. Repeating this procedure will lead the network minimise the error function, leading to more correct results.

A disadvantage of neural networks is the large computational cost of training. This, together with the availability of large datasets, is why machine learning has only become popular in the last decade. A big advantage of neural networks is their scalability and parallelism. This allows for efficient implementations on graphics cards, which makes it practical for real word applications.

2.2 Auto-Encoders

Auto-encoders were introduced in 1985 by Rumelhart et al.[4] and are a network architecture where the input is narrowed to a smaller resolution before being expanded again (figure 3). Narrowing to a smaller resolution helps keep the number of weights down and it forces the network to create a compressed encoding of the input data, discarding redundant information.

Using auto-encoders for denoising was first done by Vincent et al.[5] in 2008, by adding noise to images and training the network to recover the original image.

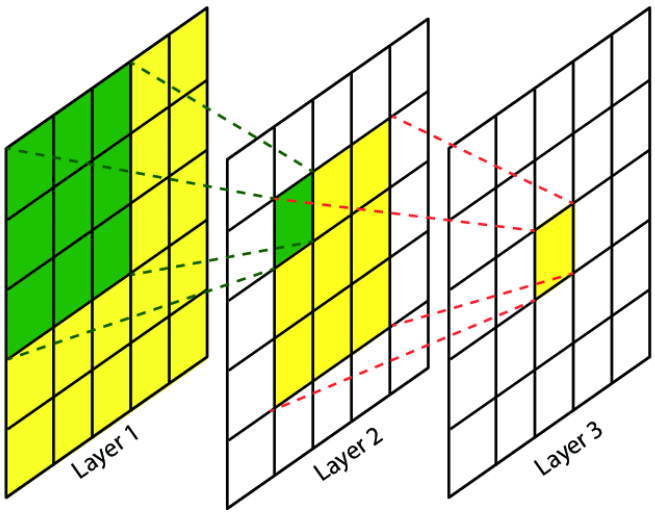

Figure 4: Three layers of a convolutional neural network with a receptive field of 3^2.

2.3 Convolutional neural networks

As the amount of neurons per layer increases, fully connected neural networks encounter a problem. The amount of weighted connections between layers grows quadratically with the amount of neurons per layer. For large data such as 2D images or as in our case 3D tomograms, this quadratic growth causes large network to quickly become intractable. As a solution to this problem, we can exploit the spatial nature of our data using *convolutional* neural networks.

Convolutional were introduced by Fukushima[6] in 1980 and are networks where instead of each neuron being connected to each of the previous layer's neurons, each neuron is only connected to its previous layer's neighbouring neurons (figure 5). This area is called a neuron's receptive field and is typically the square or cube with side length of 3, 5, 7 or 9 (figure 4).

In 2006, Chellapilla et al.[7] were the first to accelerate their convolutional neural network on the GPU. Convolutional networks have a relatively small amount of weights, but a large computation cost. The parallel nature of convolutional neural networks means that it is very well suited to be computed on graphics cards, which in general have a small amount memory compared to the CPU's RAM, but large, parallel computation throughput.

Convolutional networks were popularised in 2012 by Krizhevsky et al.[8]'s AlexNet that won the ImageNet LSVRC-2012 competition with an error rate of 15.3%, compared to the 26.2% of the runner-up.

Ede et al.[9] use convolutional auto-encoders for denoising two dimensional electron-microscope images in 2019, showing both increases in denoising quality and training speed compared to the non-convolutional denoisers.

2.4 U-Nets

As an alternative to conventional auto-encoders, U-Nets were developed. A known problem with Auto-encoders is that they lose details through the bottleneck. Ronneberger et al.[11] solved this problem in 2015 with U-Net. It was developed for biomedical image segmentation, a field where the loss of detail is a large issue. U-Nets are networks where the early layers get a shortcut to the later layers (figure 7) The first layer is connected to the last layer, the second to the second to last

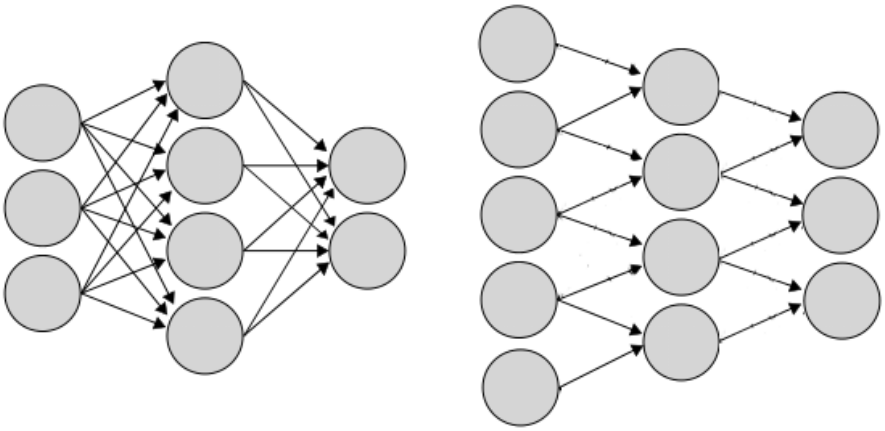

Figure 5: **left:** fully connected network, **right:** convolutional network

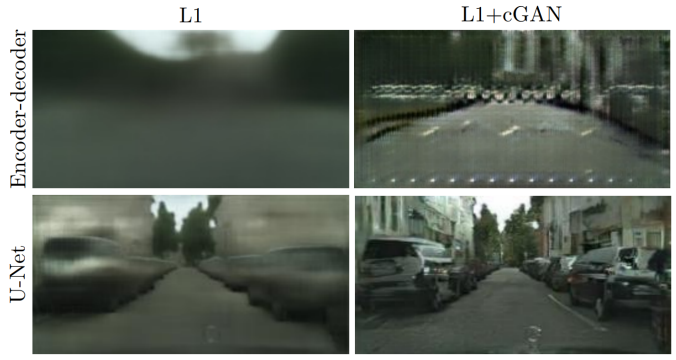

Figure 6: The result of Isola et al.[10]'s Pix2Pix network, comparing U-Nets with auto-encoders and the L1 loss function with GANs (section 2.5)

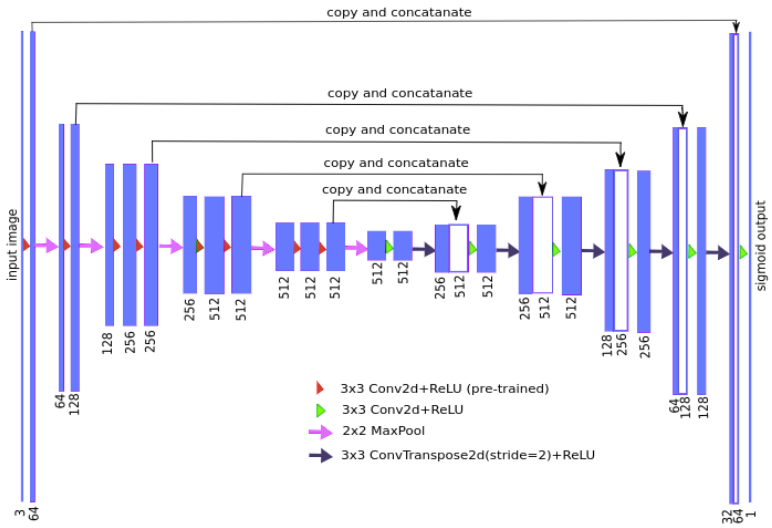

Figure 7: The original U-Net architecture[11]. Note the skip connections along the top.

et cetera. This allows details from earlier layers to bypass the bottleneck. Isola et al.[10] found that U-Nets produced images of higher quality than conventional auto-encoders (figure 6).

2.5 Generative Adversarial Networks (GANs)

Neural networks need manually selected loss functions to tell them how different their answer is to the correct answer. The gradient of the loss function can then instructs the network in which direction it needs to modify itself to get closer to the correct answer.

Creating loss functions is an open problem requiring expert knowledge. In the case where a good loss function is not known, it is possible to discover a loss function by using a second neural network in an architecture called Generative Adversarial Networks (GANs)

GANs consists of two networks: a generator and a discriminator. The generator produces images, and the discriminator either gets a 'real' image from the dataset, or a 'fake' image from the generator. The discriminator's task is to distinguish real from fake images. As training goes on, the generator will produce better images to fool the discriminator, and the discriminator will get better in discerning real from fake images.

In the original GAN formulation in 2014 by Goodfellow et al.[12], the generator gets random noise as input and has to generate any believable image. Equation 1 describes the original GAN formulation with G is the generator, D the discriminator, p_{data} the dataset and p_z random noise.

$$\min_G \max_D E_{x \sim p_{data}}[\log D(x)] + E_{z \sim p_z}[\log(1 - D(G(z))] \tag{1}$$

In 2014, Mirza et al.[13] introduced Conditional GANs (CGANs) by adding a label to the input of the generator, which instructs the generator to generate an image of a specific class. The discriminator also gets the label and then gets either a real image with that label from the dataset, or the generated one.

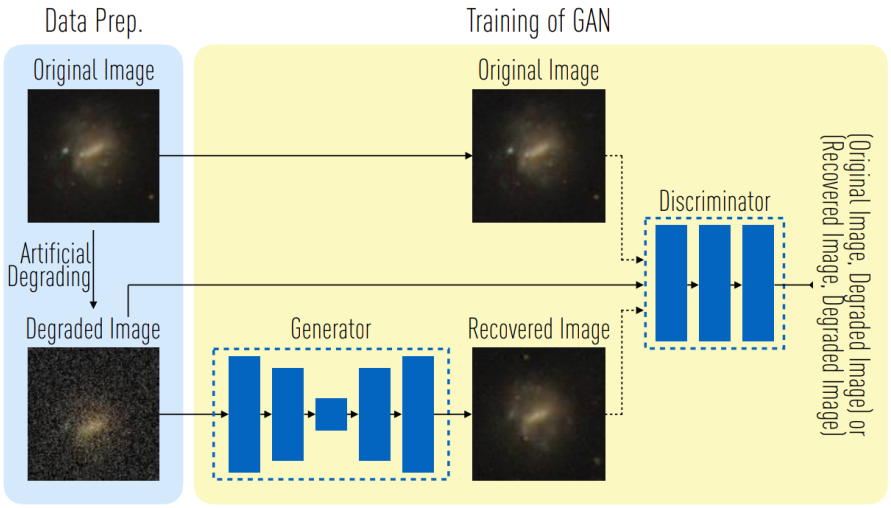

Figure 8: GalaxyGAN is an archetypal modern CGAN for denoising.

This idea of giving the generator input has since been generalized to giving the generator any form of input instead of noise. The generator then applies a transformation to the input data, instead of generating the data. Such an archetypal modern CGAN was exemplified by GalaxyGAN in 2017 by Schawinski et al.[14] (figure 8).

In GalaxyGAN, an original image from the dataset is degraded. The generator reconstruct the degraded image, and the discriminator gets the degraded image and either the 'real' original image or the 'fake' reconstructed image. Like other GANs, the discriminator then has to tell whether it got the degraded image together with either the 'real' or 'fake' image.

In 2017, Wolterink et al.[15] made tomograms at 20% of the routine-dose passable as routine-dose tomograms using a CGAN architecture, showing that denoising low-dose tomograms is feasible. An interesting difference with GalaxyGAN is that instead of teaching the generator network to generate the clean image directly, the generator tries to generate the noise component from the low-dose CT input, and then subtracts that from the image to get the denoised CT.

2.6 Wasserstein GANs

The feedback that the discriminator gives to generator is a continuous value between 0 (the image is 'fake') and 1 (the image is 'real'). In practice, this value is always near either extremes. This makes the difference between 'almost real' and 'very fake' disappear, and that in turn makes it difficult for the generator to know how it should change to better fool the discriminator.

To make the training process of GANs more stable, Arjovsky et al.[16] introduced Wasserstein GANs (WGANs) in 2017. Instead of telling generator how sure it is that an image is real or fake, it uses the *wasserstein distance* to tell the generator how far off it was to what it considers real images (figure 9). Because of this change in behaviour, the discriminator is renamed to a critic.

Wasserstein distance, also known as earth mover's distance, is the difference of two distributions or 'piles of dirt'. The metric is calculated by minimizing the amount of dirt times the distance the earth has to be moved to turn one distribution into the other. The Wasserstein distance has a closed form solution the case the distributions are represented by a point-set.

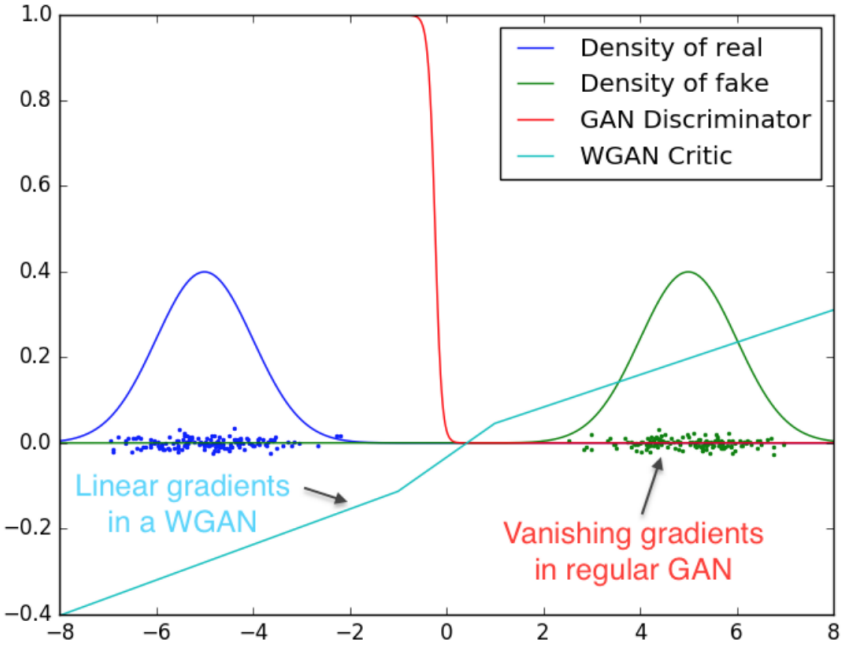

Figure 9: The cyan WGAN critic is more linear and gradual, while the red GAN discriminator is a steep cliff with a gradient of zero for most values. The x-axis represents the axis along which the two point sets differ, and the y-axis represents the output of the discriminator or critic.

The advantage of Wasserstein distance is that the smooth gradient makes training more consistent. It is worth noting however, that a smooth gradient does not necessarily mean that WGANs converge to a better result than GANs[16].

In 2018, Yang et al.[17] improved on Wolterink et al.[15]'s low-dose tomogram reconstructions by using the Wasserstein distance metric and a perceptual loss calculator.

The perceptual loss calculator was introduced to combat the blurriness that the denoising GAN produced. The loss calculator uses Simonyan et al.[18]'s pre-trained "VGG" network as an additional image similarity metric, completely separate from the GAN. During training, the output from the discriminator and the perceptual loss calculator are combined to form the loss function of the generator.

Later in 2018, Yi et al.[19] introduced "SAGAN", which combated the blurriness by combined WGANs with U-Nets. They show that their generator preserves more detail than using a perceptual loss calculator proposed by Yang et al.[17].

2.7 WGANs with Gradient Penalty (WGAN-GPs)

WGANs have a constraint on the critic which must hold in order for the WGAN to remain stable. This constraint is called the Lipschitz constraint, and it specifies that the gradient of the critic cannot exceed a chosen constant steepness.

In Arjovsky et al.[16]'s original WGAN paper, this was done by clipping the gradient if it exceeds the constant. However, clipping the gradient is a crude method to enforce this constraint and tuning the precise clipping threshold has a huge effect on how well the WGAN performs.

A more robust way to enforce the Libschitz constrained was introduced later the same year in 2017 by Gulrajani et al.[20] called a Gradient Penalty. Gradient Penalty

penalizes steep gradients proportionally such that they will never become too steep. As a result WGAN-GPs are more stable than their WGAN counterpart while keeping the smooth gradient of the WGAN.

The gradient penalty is calculates the gradient norm of the discriminator and is an additional term to the WGAN's loss function as shown in equation 2, where λ is a chosen constant, D is the discriminator, and P is the dataset.

$$L_{\text{WGAN}-\text{GP}} = L_{\text{WGAN}} + \lambda E_{\hat{x}\sim P_{\hat{x}}}[(||\nabla_{\hat{x}} D(\hat{x})||_2 - 1)^2] \tag{2}$$

3 Methodology

Instead of a single experiment with a single result, this research consisted of an iterative process with many experiments.

To denoise tomograms using machine learning, we required a large amount of data to train. We used a database of biological molecules and produced pairs of degraded and reference tomograms (section 3.1).

In order to start as simple as possible, the first network was a small auto-encoder (section 3.2). After viewing the generated tomograms (figure 10), two observations stood out. The first was a checkerboard pattern, which remained unsolved until UpsampleUnet48 (section 3.7). The second observation was that the particle reconstruction had a simpler, blob-like shape than the reference particle. A known weakness of simple auto-encoders is that the input details get lost in the bottleneck, which would explain the blob-like reconstruction. To combat this, the network was changed to a bigger, U-Net style network.

The first U-Net network that was attempted was a faithful adaption of the original U-Net by Ronneberger et al.[11] (section 3.4). However, because our network operated on three dimensional tomograms instead of the two dimensional images that the original U-Net operated on, the intermediate results of the network grew too large to fit into the 6 GiB of available graphics memory. This resulted in the network only being able to run on the main processor, which hindered the progress.

The first U-Net network was therefore redesigned so that a batch of 8 tomograms could fit into the graphics memory (section 3.5). The channels were limited to 48, leading to the name Unet48. Unet48 performed with more accurate particle shapes than the simple auto-encoder, although it still had a checkerboard pattern on the particle (figure 12).

According to Odena et al.[21], the checkerboard pattern was an artifact of the striding in the decoder part of the network. To remedy this, the striding in the decoder of Unet48 was replaced with nearest-neighbor upsampling (section 3.6). The new UpsampleUnet48 had no checkerboard patterns and approximated the reference particle more accurately than Unet48.

Next, we used UpsampleUnet48 together with a discriminator to form a GAN (section 3.7). This yielded the most accurate particle reconstructions yet (figure 15).

To improve accuracy further, we switched to using Wasserstein distance to turn our GAN into a WGAN. However, we did not manage to get the Wasserstein GAN with original weight clipping stable. Because these stability issues were addressed by Gulrajani et al.[20]'s WGAN-GP, we decided to skip WGAN and move directly to WGAN-GP (section 3.8) Counter to expectations, the WGAN-GP did not perform as well as the GAN.

Not only the networks were changed during these experiments. The optimizers, hyper-parameters, training methods and loss functions were also changed. In sec-

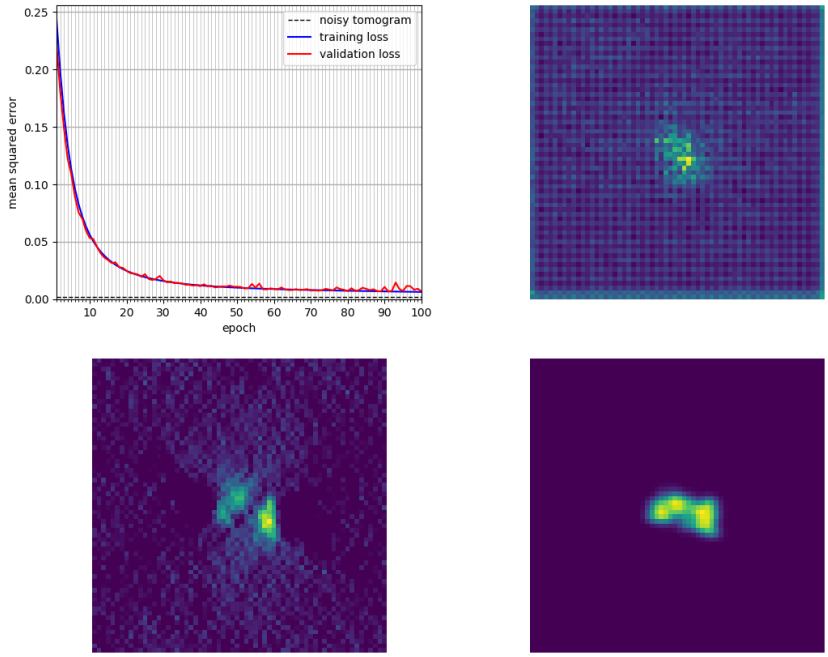

Figure 10: results of the simple auto-encoder at epoch 100. **top right:** the generated image, **bottom left:** the noisy input, **bottom right:** the ground truth.

tion 4, we measured all the networks with a variety of metrics and with everything else being equal.

3.1 Dataset

We used the following proteins from the Protein Data Bank: 1u6g, 3cf3, 4b4t, 1qvr, 3h84, 2cg9, 3qml, 3gl1, 3d2f and 4d8q. They were chosen for their variety in shape and size.

These particles are isolated particles that are centered in a 64^3 voxel volume with a resolution of 6Å per voxel. We then augment the data by copying each particle several times and rotating them into a random orientations. Finally, each particle gets bundled with its degraded version that has a combination of missing-wedge noise and Gaussian noise applied. The data is then split 80/20 into a training and a testing dataset.

3.2 Simple Auto-Encoder

The first network was a simple auto-encoder with Mean Squared Error between the generated and the reference tomogram as loss function.

The auto-encoder consisted of eight sequential layers. The first four layers convoluted the $64^3 \times 1$ tomogram to a $8^3 \times 256$ representation, halving the resolution of the x, y and z axes of the tomogram while quadrupling the amount of channels each layer. The last four layers mirror the first four, and convolute the $8^3 \times 256$ representation back to a $64^3 \times 1$ tomogram.

The method used for halving the resolution was by using *striding*, skipping every other input neuron, and the method used for doubling the resolution was transposing, or skipping every other output neuron.

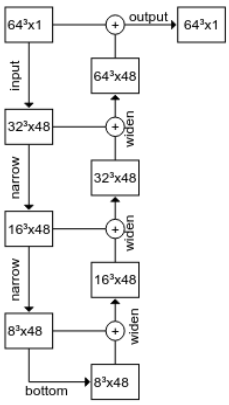

Figure 11: Schematic representation of Unet48. Intermediate results shown in boxes with their dimensions.

3.3 U-Net Auto-Encoder

A weakness of simple auto-encoders is that the details of the input tomogram get lost during the later layers, leading to blobby, unsharp shapes. U-net architectures are well suited to remedy this in a way that still keeps connectivity down. U-Net are networks where the early layers get a shortcut to the later layers (figure 7) The first layer is connected to the last layer, the second to the second to last etc. U-Nets are commonly used in image segmentation, with the goal of pixel-perfect object classification.

Our implementation of Unet48 is a direct evolution of the simple auto-encoder, adjusted to accomodate the skip connections (figure 11).

The error metric was also changed from mean squared error to absolute error, also called L1 error. This is a linear error metric instead of a logarithmic error one, making it better at penalising large errors.

3.4 Original U-Net Adaptation

The first attempt at making a U-net style architecture was an adaptation of the original U-Net as described by Ronneberger et al.[11].

A downside of U-Nets compared to the usual auto-encoders is that U-Nets are required to store all intermediate results from the first half of the network for use in the second half of the network. This means that the image size dominates the memory requirements of the network, far more than the amount of weights in the network.

The original U-net was designed to work on two-dimensional images. Because our dataset contains larger 3D volumes, the sum of all intermediate results did not fit in the 6GB of available graphics memory, and could only be run on the CPU. This was unacceptably slow, so a network with fewer layers needed to be built in order to keep the number intermediate results low.

3.5 Unet48

The second attempt was Unet48, a network with the constraint that it should be able to run with a batch size of 8 tomograms in graphics card memory.

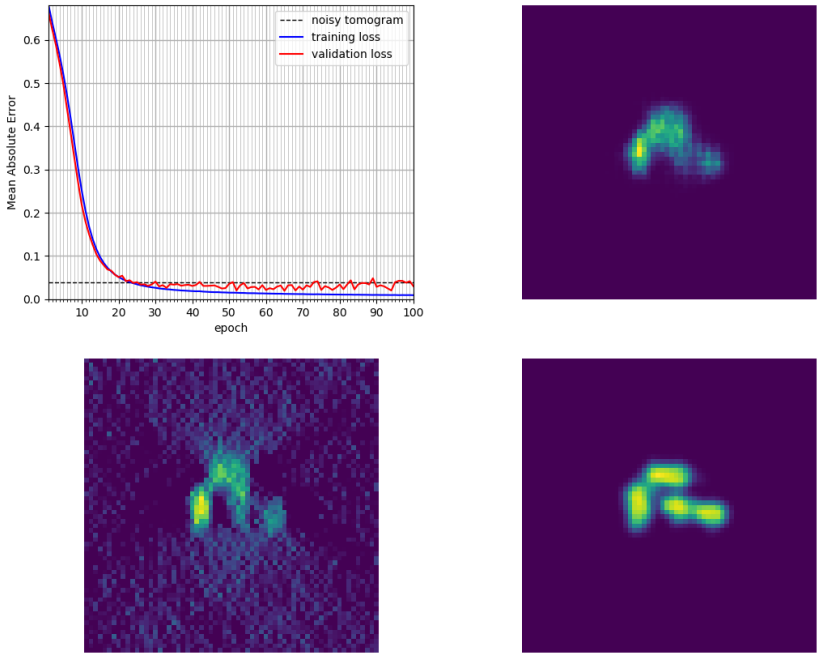

Figure 12: Unet48 results at epoch 100. **top right:** the generated image, **bottom left:** the noisy input, **bottom right:** the ground truth.

Batching multiple tomograms together improves the speed and stability of the network at the cost of extra memory consumption. The gradient of each element in the batch is averaged across the batch, leading to a more stable and less erratic gradient descent. The speed is improved because all elements in the batch can be processed in parallel on the graphics card.

Unet48 was limited to 48 channels and only required three intermediate results to be stored. As a result, it fit in the budgeted 6GB with a batch size of 8 and produced better results than the Simple Auto-Encoder, although a checkerboard pattern was still visible (figure 12).

3.6 UpsampleUnet48 Auto-Encoder

The Unet48 still generated checkerboard artifacts, and according to Odena et al.[21], these patterns are inherent to using striding. They measured convolutional network performance with multiple ways upsampling and found that nearest-neighbor upsampling worked best. UpsampleUnet48 therefore uses nearest-neighbor upsampling instead of striding in it's decoder half (figure 13). The results in figure 14 show that it was effective and that the checkerboard pattern is gone.

3.7 UpsampleUnet48 GAN

As UpsampleUnet48 was producing decent results and the current state of the art was using GANs (section 2.5) we decided to extend the auto-encoder to a GAN. The advantage of GANs is that it discovers its own similarity function, instead of relying on non-optimal expert knowledge for similarity functions.

The most challenging aspect of GANs is balancing the discriminator with the generator. A GAN that balances this well is called 'stable'. If the discriminator becomes

better at its job than the generator at its job, the gradients vanish, as the generator never gets the chance to fool the discriminator. Inversely, if the generator becomes better at its job than the discriminator, the gradients also vanish, as the discriminator never detects a difference between the real and fake images.

Factors that enhance GAN stability have been compiled into a list by Chintala et al.[22] for their presentation "How to Train a GAN?" at the Neural Information Processing Systems 2016 (NIPS2016) conference. When our GAN exhibited stability issues we implemented some of the recommendations such as using the ADAM optimizer and replacing ReLU with LeakyReLU. Among the recommendations was switching from stochastic gradient descent to the Adam optimizer by Kingma et al.[23]. This improved stability and reduced artifacts and noise.

The discriminator that was developed for the GAN was Unet48Discriminator, which was essentially the first half (the encoder) of Unet48 with four fully connected layers at the end. The results were visually the most accurate yet (figure 15).

3.8 UpsampleUnet48 WGAN-GP

After the succesful GAN results, we wanted to try upgrading it to a WGAN. However, the WGAN was not stable during training, leading to degenerate results. To address this stability issue, we moved toward WGAN-GPs.

We cycled through several discriminators as WGAN-GP is extra sensitive to how well the discriminator performs. We started with the Unet48Discriminator from our previous GAN, but got worse results than our GAN. We then modified it to be bigger which did not improve it's performance. Then we adapted an existing discriminator from Isola et al.[10]'s pix2pix from its original use-case to 3D tomograms, which produced good results. We also adapted Kodali et al.[24]'s DRAGAN critic and found that the adapted pix2pix discriminator worked better.

The results are found in figure 15.

4 Evaluation

During the experimentation process, we did not only change the networks, but also the training methods and optimizers. To see which networks reconstructs particles most accurately, we tested all networks with the exact same optimizer, learning rates, training method, metrics, and dataset in the control evaluation.

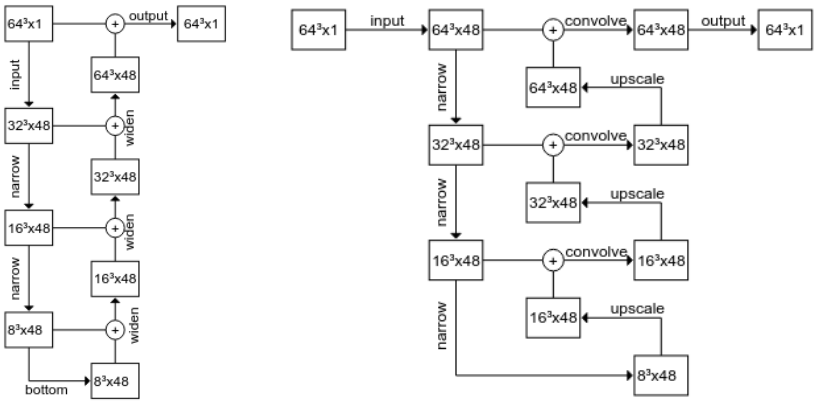

Figure 13: **left**: Unet48, **right**: UpsampleUnet48

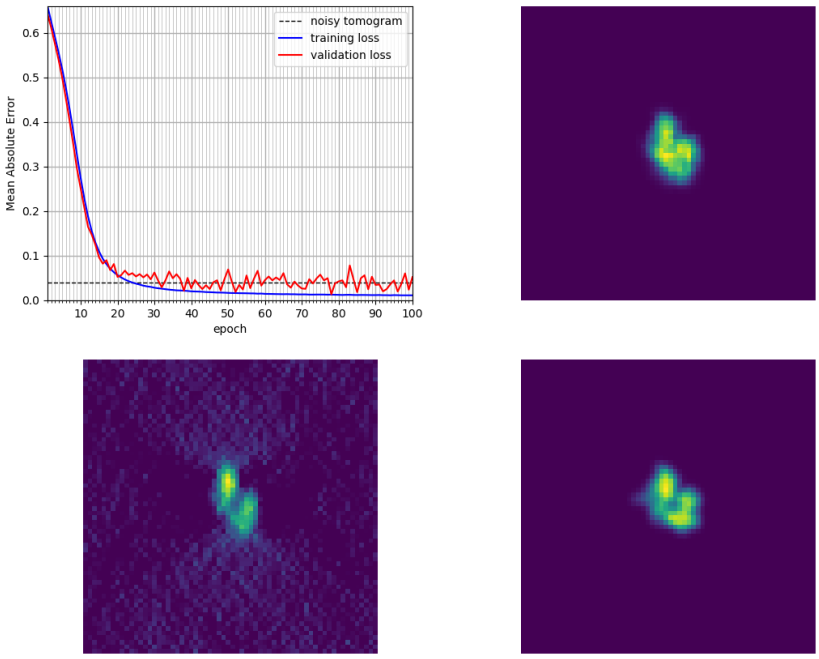

Figure 14: UpsampleUnet48 AE results at epoch 100. **top right:** the generated image, **bottom left:** the noisy input, **bottom right:** the ground truth.

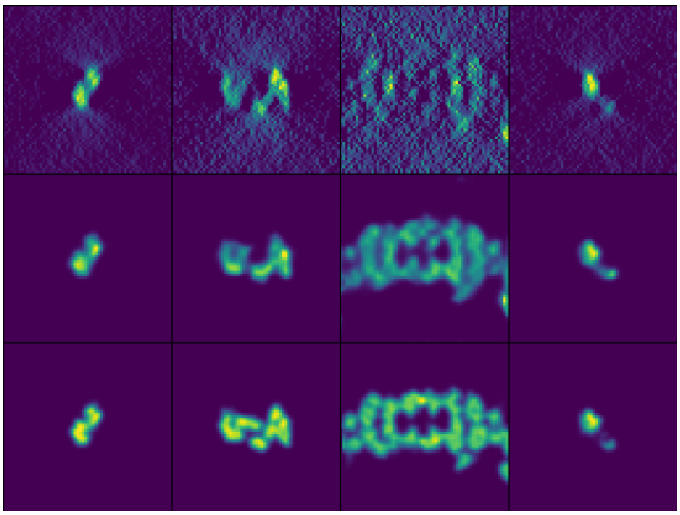

Figure 15: GAN results of four particles at epoch 20. **top row:** degraded tomogram, **middle row:** reconstructed tomogram, **bottom row:** ground truth.

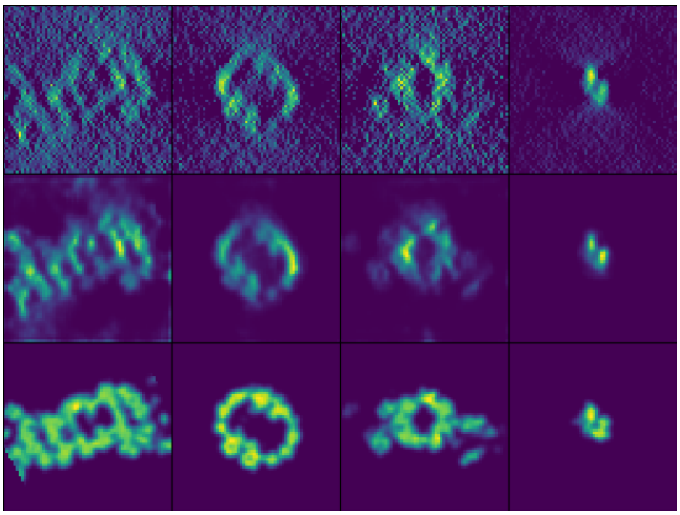

Figure 16: WGAN-GP results of four particles at epoch 20. Note that the reconstruction is less accurate than the previous GAN reconstruction in figure 15. **top row:** degraded tomogram, **middle row:** reconstructed tomogram, **bottom row:** ground truth.

GPU	Nvidia GeForce GTX 1060 6GB
CPU	Intel Core i7-8750H @ 2.20GHz
Python	3.7.3
PyTorch	1.0.1
Linux	5.1.5
Cuda	10.1
Nvidia Driver	430.14

Table 1: The hardware and software used for the evaluation.

In order to understand how well the networks generalises, the control evaluation section is followed by three experiments. The first experiment changed the type of noise, to see if the network only learned the noise. The second experiment introduced a particle that was never seen before by the networks to see if the network learned the shapes of the particles in the dataset. The third experiment gave the networks multiple particles in a single tomogram.

The hardware and software used for all evaluations can be found in table 1.

4.1 Control Evaluation

The results of the control evaluation can be found in table 2, and are visualized in figure 17.

The four metrics are used are: MSE, NRMSE, SSIM and PSNR.

MSE or the mean of the squared error between the pixels in the generated and reference image.

NRMSE or normalised root-mean-square error, is the root-mean-square error divided by the range of the data.

PSNR or peak signal-to-noise is a metric often used in signal processing to express the ratio between the maximum possible signal power and the noise power. It is conventionally expressed logarithmically in decibels (dB).

SSIM or structural similarity is an index developed as an image quality metric for use by lossy image compression. It is a metric with hand picked weights designed to mimic human perception.

From these four metrics, other metrics like the L1 loss can be extrapolated.

According to the metrics (table 2, figure 17), the best method for denoising our dataset is either the UpsampleUnet48 GAN when looking at SSIM, or Unet48 auto-encoder when looking at PSNR or (NR)MSE.

Method	P %	SSIM	PSNR dB	MSE ×1000	NRMSE
(input)	5	0.1615	18.57	1.368	0.7787
	50	0.2581	24.09	3.900	0.8949
	95	0.4638	28.64	13.890	1.0960
Simple auto-encoder	5	0.8440	20.51	0.271	0.3657
	50	0.9783	29.89	1.027	0.4319
	95	0.9914	35.68	8.893	0.6285
Unet48 auto-encoder	5	0.9386	25.53	**0.058**	**0.1734**
	50	0.9931	**35.72**	**0.268**	**0.2143**
	95	**0.9985**	**42.33**	2.809	0.4169
UpscaleUnet48 auto-encoder	5	0.8295	16.65	1.655	0.9740
	50	0.9366	23.73	4.238	0.9794
	95	0.9727	27.81	21.637	0.9848
UpsampleUnet48 GAN with SimpleDiscriminator	5	**0.9846**	**29.56**	0.070	0.2010
	50	**0.9934**	34.64	0.343	0.2475
	95	0.9981	41.56	**1.107**	**0.3030**
UpsampleUnet48 WGAN-GP with Pix2PixDiscriminator	5	0.8471	20.67	0.544	0.4548
	50	0.9750	28.54	1.400	0.5308
	95	0.9899	32.64	8.607	0.6923

Table 2: The 5th, 50th and 95th percentile of four metrics for each method, as well as for their input (the degraded tomogram). Best scores in bold: higest for SSIM and PSNR, lowest for MSE and NRMSE.

4.2 Evaluation with Different Noise

For the first experiment after the control evaluation, the noise that is used to produce the degraded tomogram was changed.

The noise added to the degraded tomograms in the main results consisted of Gaussian noise and a simulated missing wedge. In order to measure how well the networks can generalize across different kinds of noise, the simulated missing wedge was replaced with salt-and-pepper noise.

Salt-and-pepper noise involves randomly setting voxels to the minimum or maximum value. In our implementation, every voxel had a 90% chance of being unchanged. Of the 10%, every voxel had a 50/50 chance of either being set to either the maximum or minimum value of the dataset.

Salt-and-pepper noise was chosen as it introduces many places with large contrasts in the image, which is different to the localised missing-wedge noise.

The results in table 3 show that all networks are still able to denoise the tomograms, although the results have a higher variance.

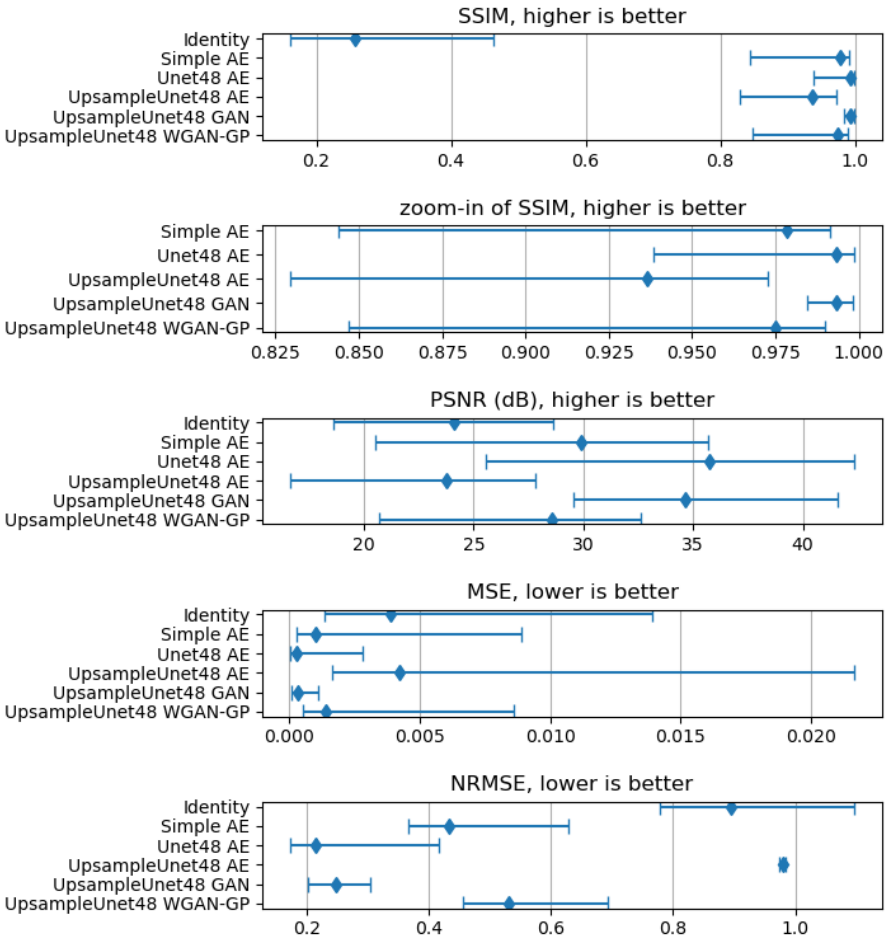

Figure 17: results of table 2 visualized. Error bars are 5th and 95th percentile, meaning 90% of samples fall within the error bars. The Identity in the input to the networks (the degraded tomogram).

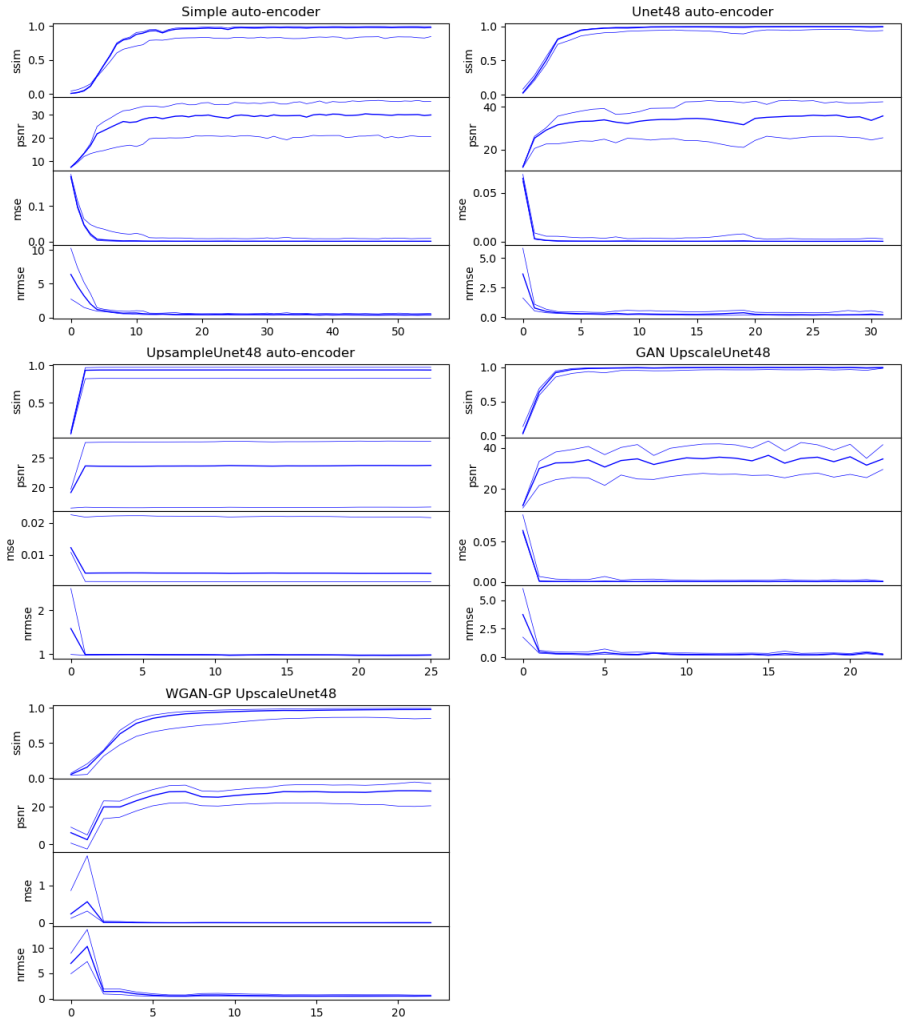

Figure 18: The training characteristics of all networks. Note the axis scaling.

Method	P %	SSIM	PSNR dB	MSE ×1000	NRMSE
(input)	5	0.021390	12.547260	0.906	0.643978
	50	0.234836	20.417239	9.084	0.714106
	95	0.564355	30.427395	55.626	4.936448
Unet48 AE	5	0.122044	17.013821	0.417	0.320498
	50	0.589574	22.754101	7.579	0.654959
	95	0.984291	33.826001	19.895	1.941162
UpsampleUnet48 AE	5	0.736181	16.553573	1.446	0.873749
	50	0.939127	24.092397	3.897	0.911100
	95	0.973684	28.397099	22.113	0.948704
UpsampleUnet48 GAN	5	0.344218	18.435151	0.532	0.400451
	50	0.909915	23.730463	4.236	0.749309
	95	0.987981	32.743983	14.338	1.675199
UpsampleUnet48 WGAN-GP	5	0.496877	12.298138	0.946	0.433812
	50	0.948350	22.714686	5.352	0.918864
	95	0.990838	30.239707	58.910	2.339938

Table 3: Results of replacing the missing wedge noise with salt-and-pepper noise.

An interesting observation is that the WGAN-GP performed best when looking at SSIM. This implies that while the WGAN-GP did not perform as well as the GAN with the original dataset, it learned to generalize noise better. Similarly, Unet48 out performed UpsampleUnet48 on the original dataset, but it does not do well with a different noise.

In conclusion, while there is a slight decrease in performance, the networks were still able to denoise with a different type of noise.

4.3 Evaluation with Unseen Particle

The second experiment after the control evaluation was denoising a particle that was unknown to the networks.

While the training and testing data is separated, both the training and testing data is generated from the same particle set. This means that while the testing data and training data do not share tomograms and are all unique, the tomograms do contain randomly oriented versions of the same particles.

The particle used was 1bxn from the Protein Data Bank (PDB), and prepared the same way as the particles described in section 3.1.

The goal of this experiment is to determine ff the networks just learned to recognize particles and reproduce them, or if the networks learned a general rule that even works on particles that the networks have not seen before. If the networks performs well in this experiment, it would indicate that it learned a general rule, and conversely if it performs poorly it indicates that it memorized the particle shapes.

The results in table 4 show results that are very similar to that of to the control evaluation (table 2). This supports the hypothesis that the network learned to generalize across particles.

Method	P %	SSIM	PSNR dB	MSE ×1000	NRMSE
(input)	5	0.472539	28.878986	1.038	0.875278
	50	0.513856	29.373947	1.155	0.934461
	95	0.553102	29.836429	1.294	1.015726
Unet48 AE	5	0.988111	35.957500	0.179	0.381121
	50	0.988857	36.741990	0.212	0.397486
	95	0.989270	37.475165	0.254	0.418359
UpsampleUnet48 AE	5	0.980214	28.817467	1.050	0.937350
	50	0.980797	29.229374	1.194	0.939507
	95	0.980942	29.789756	1.313	0.944049
UpsampleUnet48 GAN	5	0.995277	36.797838	0.139	0.328697
	50	0.995858	37.561350	0.175	0.363230
	95	0.996468	38.575766	0.209	0.412114
UpsampleUnet48 WGAN-GP	5	0.991085	32.648985	0.292	0.481257
	50	0.992536	34.423190	0.361	0.516713
	95	0.993144	35.345065	0.544	0.646941

Table 4: new particle, same scale, same noise

4.4 Evaluation with Multiple Particles

All previous evaluations have had a single centered particle to denoise. In order to measure how important this is, a 64^3 tomogram 5 from the SHREC dataset [**shrec**]. According to the results in table 5, none of the networks performed well, and the auto-encoders produced results that had a lower SSIM score than their inputs.

The tomograms from the SHREC dataset had a resolution of 10Å per voxel, compared to the 6Å per voxel for the training set. Further research is required to measure the influence of the change in scale on the result.

5 Discussion

Several interesting observations can be made from the graphs of the performance of the methods while training (figure 18). Interesting to note here is that the WGAN-GP is an outlier by being significantly slower to converge than the other methods. On the other side of the spectrum, the UpsampleUnet48 auto-encoder instantly converges to a good solution and stays there.

Also interesting to note is that SSIM smoothly improves for the GANs, this would mean that the similarity function that the GANs have independently discovered correlate closely with the SSIM.

5.1 Conclusion

In this thesis, we have shown that machine learning can be applied to the problem of low signal-to-noise in cryo-electron tomograms. We have found that a more advanced network architecture does not necessarily improve the denoising performance, and that the loss function that the GANs discover closely correlates with SSIM.

In the control evaluation we saw that the GAN and auto-encoder were tied in first place. The GAN performed best according to the SSIM score. SSIM correlates best

Method	P %	SSIM	PSNR dB	MSE ×1000	NRMSE
(input)	5	0.272971	18.265276	6.816	0.768397
	50	0.398185	20.007546	9.983	0.922379
	95	0.495846	21.664652	14.910	1.171134
Unet48 AE	5	0.327585	19.323848	9.238	0.958179
	50	0.352040	19.673591	10.781	0.961710
	95	0.388073	20.344218	11.685	0.968931
UpsampleUnet48 AE	5	0.328848	19.266815	9.241	0.958684
	50	0.350522	19.658857	10.817	0.962829
	95	0.370920	20.342693	11.856	0.968777
UpsampleUnet48 GAN	5	0.426113	20.787865	4.597	0.645596
	50	0.556883	22.423775	5.723	0.712523
	95	0.638112	23.375956	8.341	0.825840
UpsampleUnet48 WGAN-GP	5	0.376663	20.235483	5.858	0.744691
	50	0.447510	21.453568	7.156	0.794043
	95	0.526778	22.322621	9.472	0.869376

Table 5: multiple particles, different scale, same noise. With best performing auto-encoder and the GAN and WGAN.

with visual quality among the metrics. It also had a lower variance than the auto-encoder.

The auto-encoder produced some of the best individual images, but the variance is higher. Also in favor of the auto-encoder is that GANs are more difficult to train and to keep stable.

The three experiments that followed the control evaluation showed that the networks generalize very well to different particles, moderately well to different noise, and poorly to multiple particles in a single tomogram.

The additional experiment in the appendix indicates that the GAN generalizes wel to realistic data, although it is only a single datapoint.

5.2 Future Work

There are interesting leads that are worth investigating, given the results obtained so far.

First piece of future research is to test the performance of different template matching programs with the denoised and degraded particles.

Second piece of future research might be to try to train an auto-encoder on SSIM instead of L1. As we found that the GANs discover SSIM, and that our best auto-encoder ties with our best GAN, it stands to reason that combining the best of both could yield great results.

Third piece of research would be to investigate the relatively poor performance of the WGAN-GP compared to the GAN. A reasonable step forward would be pre-training the WGAN-GP critic for better performance, as WGAN-GP's performance is extra sensitive to how well the critic does.

Another lead would be to investigate why the networks perform poorly on multiple particles in a single tomogram, and find alternative approaches that improve it.

Using more metrics like the Fourier shell correlation (FSC) for network evaluation could also extend this research.

References

[1] MS Smyth and JHJ Martin. "x Ray crystallography". In: *Molecular Pathology* 53.1 (2000), p. 8.

[2] Balázs Csanád Csáji. "Approximation with artificial neural networks". In: (2001).

[3] David E Rumelhart, Geoffrey E Hinton, Ronald J Williams, et al. "Learning representations by back-propagating errors". In: *Cognitive modeling* 5.3 (1988), p. 1.

[4] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. *Learning internal representations by error propagation*. Tech. rep. California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[5] Pascal Vincent et al. "Extracting and composing robust features with denoising autoencoders". In: *Proceedings of the 25th international conference on Machine learning*. ACM. 2008, pp. 1096–1103.

[6] Kunihiko Fukushima. "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position". In: *Biological cybernetics* 36.4 (1980), pp. 193–202.

[7] Kumar Chellapilla, Sidd Puri, and Patrice Simard. "High performance convolutional neural networks for document processing". In: *Tenth International Workshop on Frontiers in Handwriting Recognition*. Suvisoft. 2006.

[8] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems*. 2012, pp. 1097–1105.

[9] Jeffrey M Ede and Richard Beanland. "Improving Electron Micrograph Signal-to-Noise with an Atrous Convolutional Encoder-Decoder". In: *Ultramicroscopy* (2019).

[10] Phillip Isola et al. "Image-to-image translation with conditional adversarial networks". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 1125–1134.

[11] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. "U-net: Convolutional networks for biomedical image segmentation". In: *International Conference on Medical image computing and computer-assisted intervention*. Springer. 2015, pp. 234–241.

[12] Ian Goodfellow et al. "Generative adversarial nets". In: *Advances in neural information processing systems*. 2014, pp. 2672–2680.

[13] Mehdi Mirza and Simon Osindero. "Conditional Generative Adversarial Nets". In: *CoRR* abs/1411.1784 (2014). arXiv: 1411.1784. URL: http://arxiv.org/abs/1411.1784.

[14] Kevin Schawinski et al. "Generative adversarial networks recover features in astrophysical images of galaxies beyond the deconvolution limit". In: *Monthly Notices of the Royal Astronomical Society: Letters* 467.1 (2017), pp. L110–L114. DOI: 10.1093/mnrasl/slx008. eprint: /oup/backfile/content_public/journal/mnrasl/467/1/10.1093_mnrasl_slx008/1/slx008.pdf. URL: http://dx.doi.org/10.1093/mnrasl/slx008.

[15] Jelmer M Wolterink et al. "Generative adversarial networks for noise reduction in low-dose CT". In: *IEEE transactions on medical imaging* 36.12 (2017), pp. 2536–2545.

[16] Martin Arjovsky and Léon Bottou. "TOWARDS PRINCIPLED METHODS FOR TRAINING GENERATIVE ADVERSARIAL NETWORKS". In: *stat* 1050 (2017), p. 17.

[17] Qingsong Yang et al. "Low dose CT image denoising using a generative adversarial network with Wasserstein distance and perceptual loss". In: *IEEE transactions on medical imaging* (2018).

[18] Karen Simonyan and Andrew Zisserman. “Very Deep Convolutional Networks for Large-Scale Image Recognition”. In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. 2015. URL: http://arxiv.org/abs/1409.1556.

[19] Xin Yi and Paul Babyn. “Sharpness-aware low-dose CT denoising using conditional generative adversarial network”. In: *Journal of digital imaging* (2018), pp. 1–15.

[20] Ishaan Gulrajani et al. “Improved Training of Wasserstein GANs”. In: *Advances in Neural Information Processing Systems 30*. Ed. by I. Guyon et al. Curran Associates, Inc., 2017, pp. 5767–5777. URL: http://papers.nips.cc/paper/7159-improved-training-of-wasserstein-gans.pdf.

[21] Augustus Odena, Vincent Dumoulin, and Chris Olah. “Deconvolution and Checkerboard Artifacts”. In: *Distill* (2016). DOI: 10.23915/distill.00003. URL: http://distill.pub/2016/deconv-checkerboard.

[22] Soumith Chintala et al. *Gan Hacks*. https://github.com/soumith/ganhacks. from ”How to Train a GAN?” presented at NIPS2016. accessed 2019-05-30. 2016.

[23] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2014. arXiv: 1412.6980 [cs.LG].

[24] Naveen Kodali et al. “How to Train Your DRAGAN”. In: *CoRR* abs/1705.07215 (2017). arXiv: 1705.07215. URL: http://arxiv.org/abs/1705.07215.

A Additional realistic data experiment

As a quick test of how well our best network generalizes to more realistic noise, we tested our best network with a single particle from SHREC dataset, but this time used their degraded tomograms instead of generating our own.

This experiment is put in the appendix because it only measures a single datapoint, and therefore does not have the statistical weight of the other experiments in section 4.

The results in table 6 show that the network is able to denoise the tomogram, as the SSIM score improves by 20×, and the PSNR improves by 3×. This implies that the networks generalize well to realistic data.

Additionally, FSC plots of the noisy tomogram (figure 19) and the generated tomogram (figure 20) were made.

Method	P %	SSIM	PSNR dB	MSE ×1000	NRMSE
(input)	50	0.0164	7.064	398.14	21.154
UpsampleUnet48 GAN	50	0.3300	22.182	0.086	0.310

Table 6: Results from the realistic data experiment

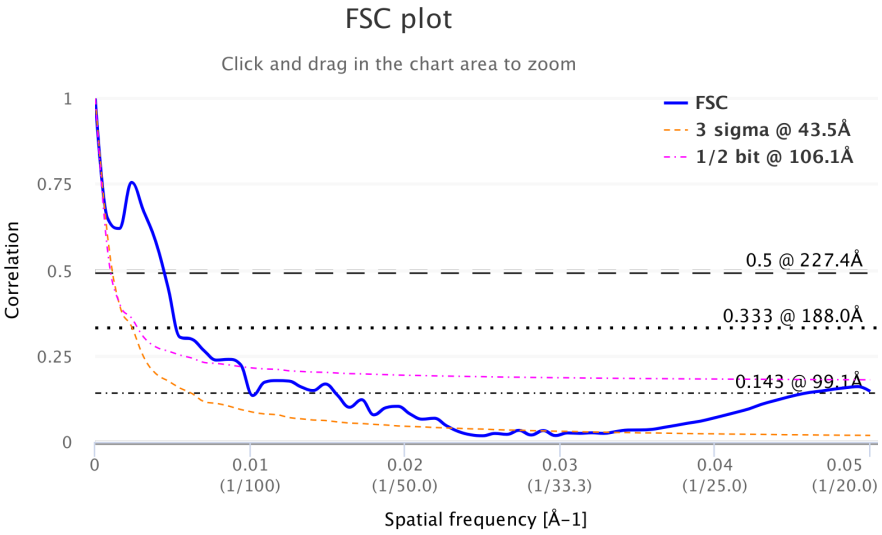

Figure 19: FSC graph of the noisy tomogram.

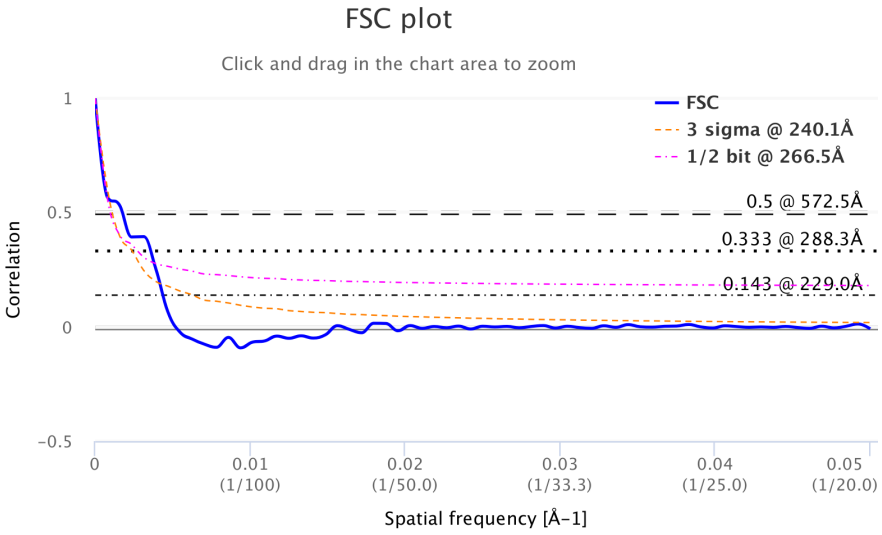

Figure 20: FSC graph of the denoised tomogram.

B Pytorch Models

```
import torch
from torch import nn

class Unet(nn.Module):
    def __init__(self, ch):
        super(Unet, self).__init__()
        self.e0 = nn.Sequential(
            nn.Conv3d( 1, ch, (5,5,5), stride=2, padding=2), nn.ReLU(True), nn.BatchNorm3d(ch))
        self.e1 = nn.Sequential(
            nn.Conv3d( ch, ch, (5,5,5), stride=2, padding=2), nn.ReLU(True), nn.BatchNorm3d(ch),
            )
        self.bot= nn.Sequential(
            nn.Conv3d( ch, ch, (5,5,5), stride=2, padding=2), nn.ReLU(True), nn.BatchNorm3d(ch),
            nn.ConvTranspose3d(ch, ch, (5,5,5), padding=2, output_padding=1, stride=2), nn.ReLU(True), nn.BatchNorm3d(ch))
        self.d0 = nn.Sequential(
            nn.Conv3d(2*ch, ch, (3,3,3), stride=1, padding=1), nn.ReLU(True), nn.BatchNorm3d(ch),
            nn.ConvTranspose3d(ch, ch, (5,5,5), padding=2, output_padding=1, stride=2), nn.ReLU(True), nn.BatchNorm3d(ch))
        self.d3 = nn.Sequential(
            nn.Conv3d( ch+1, ch, (3,3,3), stride=1, padding=1), nn.ReLU(True), nn.BatchNorm3d(ch),
            nn.Conv3d( ch, 1, (3,3,3), stride=1, padding=1), nn.Softplus())

    def forward(self, x64):
        x32 = self.e0(x64)
        x16 = self.e1(x32)
        x8 = self.e1(x16)
        xout = self.bot(x8)
        xout = self.d0(torch.cat((xout,x8), 1))
        xout = self.d0(torch.cat((xout,x16), 1))
        xout = self.d0(torch.cat((xout,x32), 1))
        xout = self.d3(torch.cat((xout,x64), 1))
        return xout

class UpsampleUnet(nn.Module):
    def __init__(self, ch):
        super(UpsampleUnet, self).__init__()
        self.input = nn.Sequential(
            nn.Conv3d( 1, ch, (3,3,3), padding=1),
            nn.ReLU(True),
            nn.BatchNorm3d(ch) )
        self.narrow = nn.Sequential(
            nn.Conv3d( ch, ch, (5,5,5), stride=2, padding=2),
            nn.ReLU(True),
            nn.BatchNorm3d(ch) )
        self.convolve = nn.Sequential(
            nn.Conv3d(2*ch, ch, (3,3,3), padding=1),
            nn.ReLU(True),
            nn.BatchNorm3d(ch) )
        self.output = nn.Sequential(
            nn.Conv3d(ch, 1, (3,3,3), padding=1),
            nn.Softplus())

    def forward(self, input):
        x64 = self.input(input)
        x32 = self.narrow(x64)
        x16 = self.narrow(x32)
        xout = self.narrow(x16)
        xout = self.convolve(torch.cat((nn.functional.interpolate(xout,scale_factor=2,mode='nearest'),x16),1))
        xout = self.convolve(torch.cat((nn.functional.interpolate(xout,scale_factor=2,mode='nearest'),x32),1))
        xout = self.convolve(torch.cat((nn.functional.interpolate(xout,scale_factor=2,mode='nearest'),x64),1))
        return self.output(xout)


class SimpleAutoEncoder(nn.Module):
    def __init__(self):
        super(SimpleAutoEncoder, self).__init__()

        # reduction rule: half the neccesairy channels to keep total voxel count the same
```

```
        # (64³ = 32³*8, so instead of 8, use 8/2=4 channels)

        a, b, c, d = 4, 16, 64, 256
        self.network = nn.Sequential( # weights = 8.3 MiB
            # nn.BatchNorm3d(1),
            nn.Conv3d(1, a, (5,5,5), stride=2, padding=2), nn.ReLU(True), nn.BatchNorm3d(a), # 64³x1 --> 32³x4
            nn.Conv3d(a, b, (5,5,5), stride=2, padding=2), nn.ReLU(True), nn.BatchNorm3d(b), # 32³x4 --> 16³x16
            nn.Conv3d(b, c, (5,5,5), stride=2, padding=2), nn.ReLU(True), nn.BatchNorm3d(c), # 16³x32 --> 8³x64
            nn.Conv3d(c, d, (5,5,5), stride=2, padding=2), nn.ReLU(True), nn.BatchNorm3d(d), # 8³x64 --> 4³x256
            nn.ConvTranspose3d(d, c, (5,5,5), padding=2, output_padding=1, stride=2), nn.ReLU(True), nn.BatchNorm3d(c),
            nn.ConvTranspose3d(c, b, (5,5,5), padding=2, output_padding=1, stride=2), nn.ReLU(True), nn.BatchNorm3d(b),
            nn.ConvTranspose3d(b, a, (5,5,5), padding=2, output_padding=1, stride=2), nn.ReLU(True), nn.BatchNorm3d(a),
            nn.ConvTranspose3d(a, 1, (5,5,5), padding=2, output_padding=1, stride=2), nn.Sigmoid())

    def forward(self, x):
        return self.network(x)

class Flatten(nn.Module):
    def forward(self, input):
        return torch.flatten(input, 1, -1)


class SimpleDiscriminator(nn.Module):
    def __init__(self):
        super(SimpleDiscriminator, self).__init__()

        a, b, c, d = 48,48,48,64 # channel size in narrowing phases
        e = d*4**3 # flat layer size
        f,g,h = e//8**1, e//8**2, e//8**3 # fully connected phase layer size
        self.network = nn.Sequential(
            nn.Conv3d(2, a, (5,5,5), stride=2, padding=2), nn.LeakyReLU(0.2,True), nn.BatchNorm3d(a), # 64³ --> 32³
            nn.Conv3d(a, b, (5,5,5), stride=2, padding=2), nn.LeakyReLU(0.2,True), nn.BatchNorm3d(b), # 32³ --> 16³
            nn.Conv3d(b, c, (5,5,5), stride=2, padding=2), nn.LeakyReLU(0.2,True), nn.BatchNorm3d(c), # 16³ --> 8³
            nn.Conv3d(c, d, (5,5,5), stride=2, padding=2), nn.LeakyReLU(0.2,True), nn.BatchNorm3d(d), # 8³ --> 4³
            Flatten(),
            nn.Linear(e, f), nn.LeakyReLU(0.2,True), nn.BatchNorm1d(f),
            nn.Linear(f, g), nn.LeakyReLU(0.2,True), nn.BatchNorm1d(g),
            nn.Linear(g, h), nn.LeakyReLU(0.2,True), nn.BatchNorm1d(h),
            nn.Linear(h, 1)#, nn.Sigmoid()
        )

    def forward(self, input):
        return self.network(input)

class ZeroPad3d(nn.Module):
    def forward(self, input):
        return torch.nn.functional.pad(input,(1,0,1,0,1,0))

class Pix2PixDiscriminator(nn.Module):
    def __init__(self):
        super(Pix2PixDiscriminator, self).__init__()
        def block(cin,cout,norm=False):
            ret = [ nn.Conv3d(cin, cout, 4, stride=2, padding=1), nn.LeakyReLU(0.2,True) ]
            if norm: ret.append(nn.BatchNorm3d(cout))
            return ret
        self.model = nn.Sequential(
            *block(2,64,norm=False),
            *block(64,128),
            *block(128,256),
            *block(256,512),
            ZeroPad3d(),
            nn.Conv3d(512,1,4,padding=1, bias=False),
            nn.Sigmoid() # manual edit
        )
    def forward(self, noisy, fake_or_real):
        pack = torch.cat((noisy,fake_or_real),dim=1)
        return self.model(pack)
```