CycleGAN for Removing Field Strength Bias in Alzheimer's Disease Classification Model

Oskar Eiler Wiese Christensen & Anders Henriksen
{s183917, s183904}@dtu.dk

January 14, 2022

Abstract

The progression of Alzheimer's Disease is slow and leads to death within 7-10 years from the date of diagnosis. In addition, AD is the most common precursor to dementia in Denmark and an estimated 85000-90000 people live with a dementia disease. As such, methods for diagnosis have been researched for many decades, especially within the field of medical imaging. A previous study exploring the possibility of classifying AD from MRI images stemming from the ADNI database showed promising results in classifying the disease, however, a bias between the magnetic field strength of the images used in the study was demonstrated. To eliminate the demonstrated bias, this project implements a CycleGAN to translate between 1.5T & 3T MRI scans in order to generate 1.5T version of 3T images and vice versa in order to retrain a classifier to classify AD without the observed magnetic field strength bias. The project finds that a CycleGAN model can successfully be trained to translate between the magnetic field strength domains quantified by a RMSE difference of 0.85064 when mapping from 3T to 1.5T and 0.9243 when mapping from 1.5T to 3T, compared to the baseline. Furthermore, when inspecting the feature representation of a classifier trained purely on original 1.5T & 3T images in a two-dimensional space, by utilizing t-SNE and forward passing cycleGAN generated images along with original images, no significant difference in their distributions can be determined. Based on RMSE, t-SNE and visual inspection, the generated images demonstrate utmost close resemblance to real images. Moreover, the synthetically generated data seems promising in training an unbiased classifier to predict AD. Contingent on the t-SNE plots, the classifier exhibits no bias towards the magnetic field strength. Based on the amount of resources society spends on AD & dementia, a successful prediction model will economically drive down cost and improve life quality of future AD patients. The findings of the project implies that a combination of a cycleGAN & deep learning classifier might help reduce the severity of AD for a multitude of generations to come, and in the long run may aid in finding a cure or reason as to how & why AD occurs.

Glossary

Term	Explanation
AD	Alzheimer's Disease
ADNI	Alzheimer's Disease Neuroimaging Initiative
AI	Artificial Intelligence
Adam	Adaptive Moment Estimation
BCE	Binary Cross-Entropy
CN	cognitively normal
CNN	Convolutional Neural Network
CRS	Coordinate Reference System
CSF	Cerebrospinal Fluid
CycleGAN	Cyclic Generative Adversarial Network
DKK	Danish Kroner
DL	Deep Learning
DTU	Technical University of Denmark
EM-GCA	Electromagnetic Gaussian Classifier Array
FS	FreeSurfer
GAN	Generative Adversarial Network
GCA	Gaussian Classifier Array
GPU	Graphics Processing Unit
HPC	High Performance Computing
KL	Kullback-Liebler
LONI	Laboratory of Neuro Imaging
MCI	Mild Cognitive Impairment
ML	Machine learning
MNI	Montreal Neurological Institute
MNIST	Modified National Institute of Standards and Technology database
MRI	Magnetic Resonance Imaging
NA	Not Applicable
PCA	Principal Component Analysis
PET	Positron Emission Tomography
RGB	Red-Green-Blue
RMSE	Root-Mean-Square Error
ReLU	Rectified Linear Unit
SVD	Singular Value Decomposition
WGAN	Wasserstein GAN
fMRI	Functional Magnetic Resonance Imaging
kSVM-DT	kernel support vector machine decision trees
t-SNE	t-Distributed Stochastic Neighbor Embedding

Contents

1 Introduction

Alzheimer's Disease (AD) is a disease which, in the year 2021, affected an estimated 5.8 million people aged 65 years or older in America alone. By the time an individual shows early symptoms of AD, whether by a screening test or a physical measurement of the brain volume, it is already too late to prevent the rapid development of the disease. Thus, to decrease cost of treatment and increase chance of successful rehabilitation, it is increasingly more important to detect AD. Many studies has been conducted within the field of Alzheimer's with magnetic resonance imaging (MRI) scans stored in the Alzheimers's Disease Neuroimaging Initiative (ADNI). The MRI scans range from an electromagnet field strength of 1.5 tesla (T) to 3T. The quality of the 3T images is higher, but the price unfortunately follows this same trend with MRI electromagnet field strength being determined mostly by funding. Inherently, this could introduce a bias in an algorithm trained on a dataset with MRI images spanning both electromagnet field strengths, resulting in images with different levels of detail. Previous studies have attempted to create a binary classifier for Alzheimer's with the tools of machine learning and computer vision, however, a bias between the magnetic field strength of images was found by Kergel [20], which was illustrated by projecting high dimensional features of the model into a two dimensional space using t-distributed stochastic neighbor embedding, see figure 1.

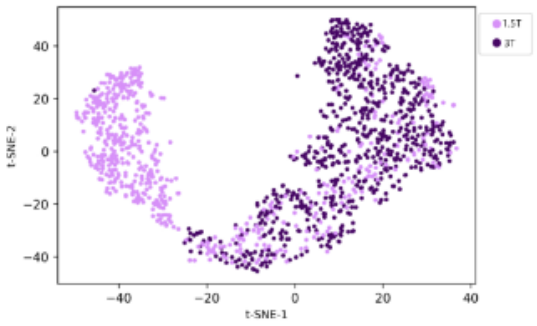


Figure 1: t-distributed stochastic neighbor embedding from Camilla Kergel Pedersen: Demographic bias in public neuroimaging databases, and its effect on AI systems for computer-aided diagnosis [20].

1.1 Scope of the Project

The aim of this project is to train a classifier on MRI images to predict AD as well as training a cycle-generative adversarial network (CycleGAN) to generate 1.5T images from 3T images in order to construct more usable and hopefully unbiased data. Thereby, the model will have more available training data, which should lead to more generalizable classifiers, which will be able to aid doctors and humans in need all around the world. This project will also aim to determine possible biases introduced in the model, and discuss how a classifier can be implemented as a tool for doctors. Furthermore, the project will discuss how this might benefit AD prevention, reduce treatment cost and the possible ethical scenarios that might be present.

similarly, the project will revolve around the possibility to predict Alzheimer's by utilizing CycleGAN generated data. Furthermore, the questions below will be researched in detail.

i Can a CycleGAN be trained to map between the domains 1.5T and 3T in MRI?

ii How effective is the training progress in a prediction model when predicting whether a patient has Alzheimer's disease? Can the CycleGAN data remove previous demonstrated bias from the model?

iii Will the CycleGAN have any societal and economic impacts given the success of the prediction model?

1.2 State of the Art

In recent years, many attempts have been made to both classify and detect Alzheimer's Disease (AD). Many different approaches to this classification problem have been taken. However, as computers get better and the field of machine learning and deep learning grows, an arsenal of new methods to predict and classify AD at early stages have been proposed as of late. This section aims to highlight some of finest work within this field of study, demonstrating the current state of methods used to approach the classification problem.

A novel approach was proposed by Zhang et al. in 2014, using kernel support vector machine decision trees (kSVM-DT). In this study, they use basic preprocessing as well as principle component analysis for feature

extraction. On the basis of these extracted features, the kSVM-DT was constructed. The proposed model has an 80% classification accuracy. Furthermore, the computation of classifying a subject is relatively fast, taking 0.022 seconds [29].

Another approach was taken by Suk and Shen using a stacked auto-encoder. Their method was different from previous methods which focused mainly on low-level features such as gray matter tissue volumes from MRI. A stacked auto-encoder can represent complicated features such as non-linear relationships. Combining the low-level features with latent information, they created a robust model for classification achieving 95.9% accuracy and 75.8% prediction accuracy of MCI to AD conversion [26]. In 2015, the work was extended and they achieved an accuracy of 98.8% for AD/CN classification and 83.7% accuracy for prediction of MCI to AD conversion. This was done with greedy layer-wise pre-training and fine-tuning of the deep learning model [27].

Recently, convolutional neural networks (CNN) have been showing promising results for AD prediction. In general CNNs have been a widely implemented method within the field of image recognition. Cheng et al. propose to construct multi-level CNNs to gradually learn and combine the multi-modality features for AD classification using MRI and PET images. An accuracy of 89.6% was achieved using this method. [28]

Basaia et al. achieved the highest measured accuracy on the ADNI1 data with a CNN classifier. The highest rate achieved was 99%, thus concluding that CNNs serves as a powerful method for Alzheimer's diagnostics. Furthermore, the model demonstrated good predictions with no prior feature engineering nor variability of imaging protocols, and Basaia et al. concludes that it is likely to be a generalizable model even on unseen patient data [25].

1.3 Contributions

This report aims to explore the possibility of using CycleGANs to translate between the 1.5T and 3T domains. Furthermore, it aims to remove the inherent bias of using the two types of images in deep learning models. As such, the reader can expect to get the following from the study:

- Illustrations which show an overview of the data in the different parts of the preprocessing pipeline. What does it look like? How is it done?

- A GAN will be provided in a Jupyter Notebook for easy access. Furthermore, an equivalent model has been run and the results and discussion are added to the appendix of this paper. This aims to give the reader important insight into the mathematical intuition behind the GAN.
- Five different CycleGANs will be presented, both through text and mathematical definition. They are implemented in order to show the effect of the model as a bias corrections tool and to see how efficacious the CycleGAN actually is in mapping between the domains.
- To put the other contributions into a real-world context, an ethical and societal discussion of the CycleGAN and a classifier will be carried out.
- Code for the preprocessing, the implementation of the CycleGANs as well as a section on reproducibility will be provided in the interest of making the results of this paper both as replicable and as reliable as possible. The section on reproducibility supplies the reader with the used seeds, packages, versions of packages and a thorough explanation of the code used in the project.

The general goal of the contributions given above is to create a model which can remove previously demonstrated biases. Furthermore, the report also aims to give an overview of the implication of a successful CycleGAN within the Alzheimer's and medical research area. The economic incentives, societal impact and ethical considerations will all be touched upon in the discussion of this paper.

2 Background

This section will contain relevant information to understand and reproduce the project presented. Magnetic resonance imaging is introduced concisely, as well as magnetic field strength, some trade-offs of using MRI scanners and MNI space.

2.1 Magnetic Resonance Imaging (MRI)

MRI is a technique developed through the 1970s and 1980s. The scanner itself can be seen on figure 2. In simple terms, an MRI scanner works by applying a magnetic field to a patient. The magnetic field will then excite the atoms within an area of the subject's brain. The resulting signal is then measured by the scanner [62].

Figure 2: A classic MRI scanner from MART PRODUCTION. Original image can be found here.

MRI was originally called nuclear magnetic resonance imaging, but was later readopted as MRI due to *"nuclear"* having negative associations with the field of research [64, 63].

2.2 Magnetic Field Strength

It is inherently important to understand the underlying concept and difference of the magnetic field strength of the MRI scanner in use. In this project, two types of images are used: 1.5T and 3T images. Tesla is a unit

in which the strength of magnetic fields are measured. The stronger the magnetic field, and therefore the magnetic resonance, the more detailed the MRI image. The strength of the magnetic field can be thought of as reducing the amount of noise in the image and adding more clarity. Thus this is like increasing the resolution of a normal RGB image. The 3T images are also constructed with more expensive machines and are not available in all countries nor hospitals around the world. This introduces the need for a method to translate between these domains.

2.3 Trade off of Using MRI

MRI data is constructed from an MR scanner in the form of volumes. These 3D images can be compared to centicubes, which together form a big square, where the brain is inside. Each centicube in this image is often referred to as a voxel, and each voxel contains a value that represents the intensity value in the specific area - see figure 3. For T1 weighted or structural images, this determines the boundary between gray matter & white matter from cerebrospinal fluid. In T2 weighted images, it represents more active voxels to less active voxels. However, voxels do have certain issues. A voxel can, for example, encompass information about two anatomically distinct areas, like if a voxel encompasses two different gyri. Furthermore, it is hard to measure structural properties such as gray matter thickness or volume, which might be important for AD classification. FreeSurfer Software can help by turning the 3D representation into a 2D representation using triangles. The points of the triangles are vertices and they are connected by an edge. Each vertex then contain measurements such as gray matter volume and thickness, area, curvature etc. These surfaces can be inflated to visualize where MRI activations and gray matter differences are located more precisely than the voxel representation would have allowed for.

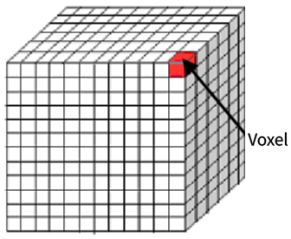


Figure 3: A cube that represents the grid in which the subject brain is located. Each voxel will contain information about the intensity of the area. This representation is prone to loss since one voxel can contain information about multiple areas of the brain.

2.4 MNI Space

Montreal Neurological Institute 305 (MNI305) is an atlas for brain mapping which was introduced by Evans et al. in 1992. The mapping was created from 250 normal subjects that have been MRI scanned. It is a known method for normalization and is an often-used industry practice for an average human brain [59].

The atlas was constructed in two steps. Firstly, anatomical landmarks were outlined for each subject manually. The landmarks were found using the Talairach atlas, which is a 3 dimensional coordinate system used to map several areas of the brain, independent of brain volume and shape. Secondly, each MRI volume from the patients were mapped to an average MRI from the previous subjects, to reduce impact of order effect. The mapping was performed with whole-brain linear (9-parameter) image similarity residual and not according to Talairach's piece-wise linear model [60].

Thus, they created a space, namely the MNI305 space, which is used in this project. The space is useful to compare the brains of different subjects and for a deep learning model to learn meaningful feature representations of such images.

3 Data

The data used in this project is from the Alzheimer's Disease Neurosurgical Initiative (ADNI). This initiative unites researchers who study data to investigate and define the progression of Alzheimer's Disease (AD). ADNI collects, validates and utilizes data, including (MRI), positron emission tomography (PET), genetics, cognitive tests, cerebrospinal fluid (CSF) biomarkers and blood biomarkers as predictors of the disease [21]. More specifically, the project will use MRI data from ADNI 1 & 2, which was launched in October 24, 2004. Originally, the project was designed to find more accurate biomarkers for the early detection of AD. ADNI have analyzed more than a thousand different brain scans from three different subject groups, namely subjects with mild cognitive impairment (MCI), subjects with early AD and cognitively normal (CN) subjects [22].

Additionally, 40 of the subjects have a baseline scan from both 1.5T and 3T MRI scans, which, later in this report, will enable benchmarking by using different evaluation metrics.

All of the data used from the ADNI database can be downloaded here: https://ida.loni.usc.edu/
This website is owned by Laboratory of Neuro Imaging (LONI). The core philosophy of LONI is "to increase the pace of discovery in neuroscience by better understanding of how the brain works when it is healthy and what goes wrong in disease" [61].

3.1 Data Demographics

To get an understanding of some of the summary statistics of subjects in this project, the figure and table below have been constructed. From figure 4, it can be seen that the average subject age is 73.3 years. Furthermore, the proportion of men to women is close to 50%. The potential for an inherent gender bias is thus low considering the ratio of men to women is almost equal.

	CN	AD	MCI	Other	Total
N	698	377	634	473	2182
Male/Female	300/398	196/181	389/245	243/229	1128/1053
Age	75.1	73.24	74.1	71.2	73.3

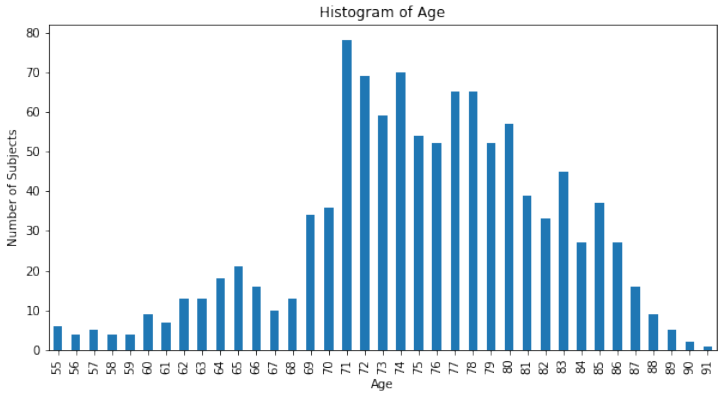


Figure 4: A histogram of the age distribution among the subjects in ADNI1 and ADNI2 [21, 22].

3.2 Description of Data

The images from the ADNI dataset are from several different subjects and a plethora of studies around the world. Each of the patients have had multiple scans throughout the span of the ADNI project, however in this project, only the baseline scan, the first MRI scan for each subject, is used. The MRI scans are three dimensional images of the subject's brain. The original image is in a `.mgz` file format which is a compressed `mgh` file. This file format can store 3D-images along with headers containing information about the subject and a transformation matrix. The raw images stem from the original analysis from 2004 and onwards. There are in total 1562 unique brain scans, all of which are used in this project.

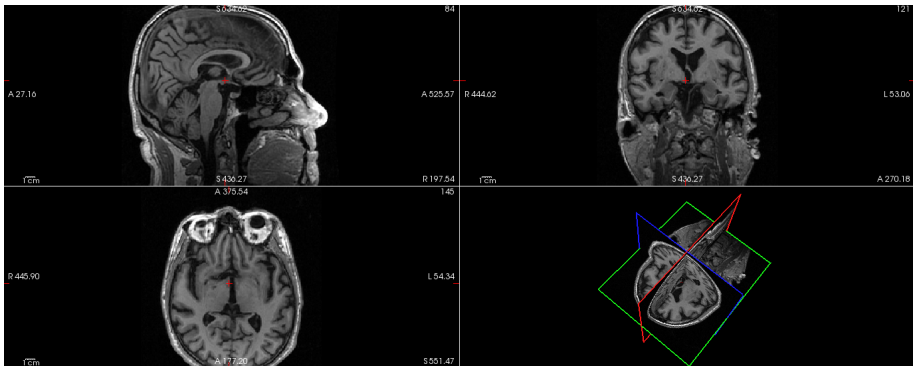


Figure 5: An example of the original raw image file, from subject 002_S_-0295. This is the original raw image from the MRI scan and no processing has been done.

3.2.1 Data Preprocessing

In this project, the area of interest is the brain itself. A lot of different tools from the FreeSurfer Software Suite have been used to produce images of the brain. The constructed brain images and how to construct these for reproduction of the results is described in the following. The raw image is called:
`ADNI_002_S_0413_MR_MPR____N3__Scaled_2_Br_20081001114937668_S14782_I118675.nii` and is shown in figure 5.

The first thing that happens in the preprocessing is motion correction. If multiple images of the source volume has been taken, then correction for motion will be done by averaging all the images. Next step in the preprocessing is non-parametric non-uniform intensity Normalization (N3), which corrects for intensity non-uniformity in the MRI image. Then, a script performing the Talairach coordinate conversion is applied. This script computes the affine transformation from the subject's original space into the MNI305 atlas. The script creates a file called `talairach.xfm`. Then, another normalization of the intensities in the original image is done. The intensity of all voxels are scaled so that the mean intensity of the white matter is 110 [24]. As the brain is the key area of interest as mentioned earlier, the skull is stripped from the image. The skull-stripping process is done by using Freesurfer Software Suite and thresholding [23]. The FreeSurfer software will create files which can be used to check if too much brain matter is removed in the process of skull-stripping the raw image. Files which can be used to inspect this include `brainmask.mgz, T1.mgz` and `brainmask.gcuts.mgz`. Next, electromagnetic gaussian classifier array (EM-GCA) Registration is done, which computes transforms to align `mri/nu.mgz` to the default GCA atlas from the FreeSurfer Home folder. Lastly, the CA Normalization happens, which outputs a normalized version of the skull-stripped image. This is saved as `mri/norm.mgz` and is going to be used as the data for each subject in this project [58].

After the steps above are done, a registration is applied. All human brains are different and the raw images are recorded in one space for each subject. In order to compare several brains, a registration of the images is done to ensure they are all in the same coordinate reference system (CRS). A standard CRS for the common brain is called the Talirach space. Each of the raw images in the ADNI-dataset are registered to the Talirach space using the `mni305.cor` template from FreeSurfer. This is done using `mri_vol2vol`,

which resamples a volume into another CRS. Thus, this project uses an MNI305 transformation by resampling the original normalized images into talairach space using `$subject/mri/transforms/talairach.xfm`

3.2.2 Preprocessing Scripts

For this project, two different type of preprocessing scripts have been used. These will be shown in the appendix, but are of key importance to the project. The first script will apply the many preprocessing algorithms described in the previous section. Most importantly, it creates the `norm.mgz` image, which, as mentioned earlier, is a skull-stripped and normalized version of the subject's brain see code in figure 46 in appendix[9]. The produced `norm.mgz` file can be illustrated using Freeview, a FreeSurfer Software used to display and examine 3D images. When displaying the new `norm.mgz` file, a big difference can be seen between the image before processing in figure 5 and after processing in figure 6.

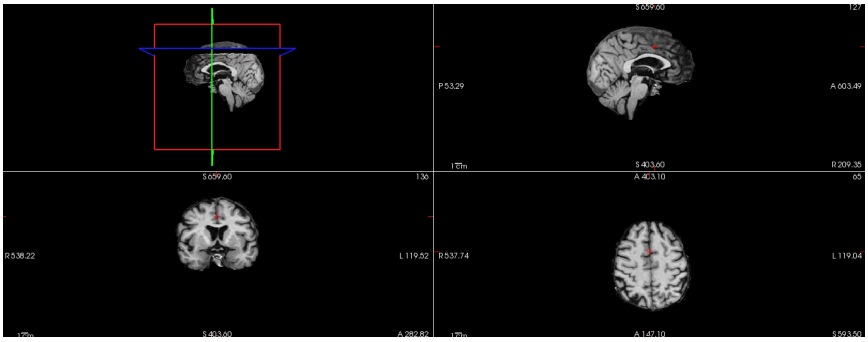


Figure 6: The above illustrates the `norm.mgz` for the subject `002_S_0295` in the subject's own normalized space.

The next script uses the `mri_vol2vol` command, which uses advanced registration to ensure the `norm.mgz` image is in Talairach space using the mni305.cor.template - this will allow for both compatibility between the different subject images and make it possible to train a CycleGAN model to transfer between the two tesla domains. The script can be seen in the appendix. See figure 47 in appendix[9]. The resulting image after FreeSurfer preprocessing is shown in figure 7. This image will resemble figure 6, but it is in a different space, where it has been aligned to the MNI305 atlas.

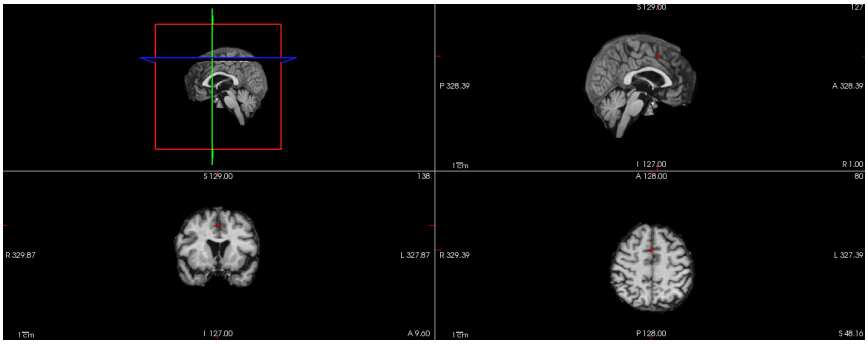
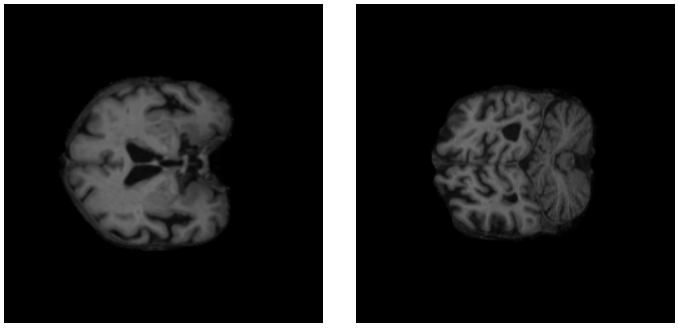


Figure 7: The `norm_mni305.mgz` for the subject `002_S_0295` in the Talairach space.

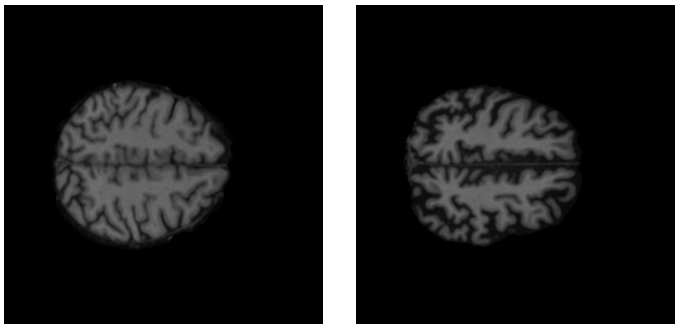
3.3 Visualization of Data

This section aims to give a visualization of the preprocessed data, and illustrate the images used to train the CycleGANs in this project. The images are constructed by thresholding one of the dimensions x, y, z, which creates the following planes: `xy, xz, yz`. The xy-, xz- and yz-planes are 2D-images. If an image is converted to a `numpy nd` array, the slicing would amount to: `slice = np.array([:,:,z])`, where $z \in [1,256]$. Thus, there are 256 2D images for each of the three planes for every 3D image resulting in a total of 1.152 million 2D images. The training images are brain slices, which can be seen in the figures 8a, 8b and 8c [1]. All the slices are created using the `create_imgs_1.5T.py` and `create_imgs_3T.py` python scripts which utilizes numpy and multiprocessing. The scripts can be seen in the repository of this project here: 3T_script and 1.5_script. An important point to note is that completely black images are discarded. This means that only slices with at least one pixel value different from zero are used to train the CycleGAN. This is partially a way to reduce the raw amount of training data, but it is also done because a CycleGAN would never need to learn to translate completely black images from 3T to 1.5T and vice versa, since the black slices can be removed from the image before conversion and reinserted afterwards.

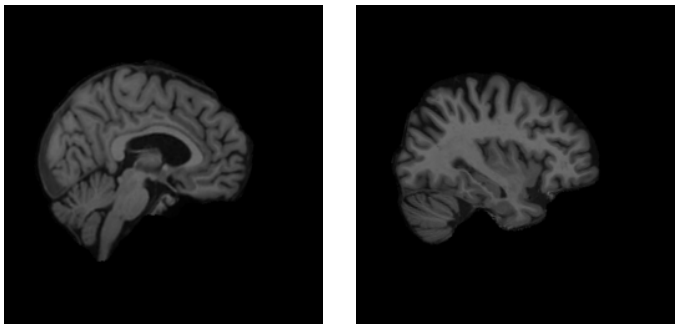
[1] The images in the figures are from randomly selected slices and do not match within 1.5T and 3T



(a) 1.5T (left) and 3T (right) respectively XY MRI Planes from subject 002_S_6695 and 009_S_4359



(b) 1.5T and 3T respectively XZ MRI Planes from subject 002_S_0413 and 002_S_4171



(c) 1.5T and 3T respectively YZ MRI Plane from subject 002_S_4171 and 002_S_0413.

Figure 8: Each of the subfigures show two examples from the three different planes constructed. The images displayed here is a part of the training data.

3.3.1 Data Augmnentation

Besides the immense preprocessing done with the FreeSurfer Software Suite, the images are also resized, center cropped, converted to float and normalized with a $\mu = 0.5$ and $\sigma = 0.5$. Centercrop crops an image in the center, ensuring that when the image is converted to a tensor and has an arbitrary number of leading dimensions, it will be cropped correctly. Furthermore, the intensity values are normalized to avoid the vanishing and exploding gradient problem when doing backpropagation. This augmentation only applies for data used for the CycleGAN model[2].

[2]The data used for the CNN classifier will be explained in section 4.5.2

4 Methods

This section introduces what technical background has been used in the duration of this project as well as which types of models have been trained. Section 4.5.1 introduces the classification network used to find and analyze the bias present in using both 1.5T and 3T MRI images for analysis. Section 4.1 and 4.2 consider the intuition and mathematical theory behind implementing GANs and CycleGANs. The GAN has been trained on the Modified National Institute of Standards and Technology database (MNIST) hand-drawn digit dataset as a preliminary introduction to GANs. The results of training this model is found in section 9.2. As for the CycleGAN, five total models were implemented on various datasets. One of these models was trained on the horse2zebra dataset used in the original CycleGAN paper [35] as another preliminary experiment and as an introduction to CycleGANs. The result of this preliminary run is located in section 9.3. The four other models have been trained on different variations of the MRI files mentioned above. One has been trained on slices from the x-axis (YZ), another from the y-axis (XZ) and yet another on the z-axis (XY). The last model was trained on all (ALL) of these images simultaneously and outputs an average 3D image from predicting each 3D image. The goal was to test the best approach for reconstructing 1.5T images from 3T images, and to see if the ALL-model could hold up to the quality of the other models. The specifics about training variables, hyperparameters, etc. are located in section 4.2.5. Since these models are not easy to train [41], section 4.2.6 relays multiple issues along the training process and how they were eventually fixed. Lastly, section 4.7 discusses what has been done to ensure reproducible results.

4.1 Generative Adversarial Network

Since the meat and potatoes of this project is the CycleGAN, the methods and mathematical understanding behind GANs are important to grasp before diving deep into the theory of CycleGANs. A GAN was implemented on the MNIST dataset as a preliminary trail to help understand the basics. This implementation can be seen in the python script `train_GAN.py`. An older version of the same code is also accessible as a Jupyter Notebook in our GitHub repository here. This implementation in collaboration with this section will hopefully help to give the proper insight into the reasoning and intuition behind CycleGANs later in this section.

In general, GANs can be seen as two different complex neural networks

competing for the best accuracy during the process of generating a class of pictures. In other words, the networks are adversaries in generating or classifying images, from which the source of the name becomes clear. One of the networks will be focusing on generating fake images, called the *generator*, and the other will be trying to distinguish between real and fake images, called the *discriminator*. This competition between networks leads to results comparable or better than many ML and AI image generation techniques, thus making a good foundation for the generation of 1.5T MRI images from 3T. The two networks are covered in depth in the below subsections.

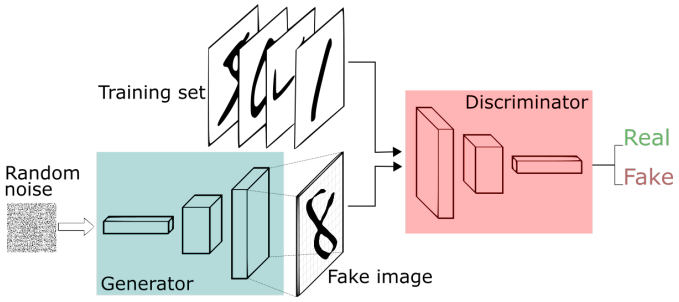


Figure 9: The GAN architecture. The generator decodes random noise from a latent space into a fake image. This fake image as well as an image from the training set is fed into the Discriminator, which encodes data into a binary prediction of whether the image is real or fake. Figure courtesy of [37].

4.1.1 Generator

The generator is the most important part of the GAN, since it is responsible for actually generating convincing images. Sadly, making a neural network generate real images is a difficult task to say the least. For that reason, instead of asking the generator to generate 28x28 hand-written digits by comparing the generator output to real digits using some objective function, the output of the generator is instead fed to the discriminator, which predicts if the generated image is real or not. If the discriminator is tricked by the generator, then the generator deserves a low loss for having made a "real" image (in practice, the generator and discriminator both start off unoptimized, so it takes many steps for real images to emerge). This is conceptually a much easier problem to solve, since the generator learns what is necessary to fool the discriminator instead of what characterizes a hand-written digit [47].

To generate the image, a `(batch_size, 100)` vector of gaussian distributed noise is given as input, and the output is an image of dimension `(batch_size, 1, 28, 28)`, since the image is black-white. The layers in-between consist of five linear layers. All layers but the last one have batch normalization and leaky-ReLU with x set to 0.2. The last layer uses hyperbolic tangent as activation function and is afterwards reshaped from 28^2 to `(batch_size, 1, 28, 28)` to be able to view the output as an image [50]. The architecture described above is also shown in table 1.

Layer	Activation Size	Value	# Parameters
Input	100		0
linear layer	2848		287,648
BatchNorm1d	2848	2848	5,696
LeakyReLU	2848	0.2	0
linear layer	2848		8,113,952
BatchNorm1d	2848	2848	5,696
LeakyReLU	2848	0.2	0
linear layer	2848		8,113,952
BatchNorm1d	2848	2848	5,696
LeakyReLU	2848	0.2	0
linear layer	2848		8,113,952
BatchNorm1d	2848	2848	5,696
LeakyReLU	2848	0.2	0
linear layer	28x28		2,233,616
Tanh	28x28	NA	0
Total Parameters			26,885,904
Trainable Parameters			26,885,904

Table 1: All layers used in the final generator as well as how many parameters are used to train them. The activation size refers to the output of the layer in the same row. As such, applying the first linear layer yields an output vector of size 2848. The input "layer" is added for completeness to show which size input is needed for the model to run correctly.

The general flow of learning in the generator is as follows. First, random noise is fed to the generator as input. The generator uses this input to produce an image as output. This output is then fed to the discriminator as input. Afterwards, the discriminator outputs a prediction of whether

the image is real or fake. The loss from this prediction is calculated and backpropagated through both models. This gives the gradients which are used only for updating the generator for better image generation in next step [47].

4.1.2 Discriminator

The discriminator is not nearly as important as the generator, though it still plays a very large role in generating likely images. If the discriminator did not do a good job predicting real images, the discriminator loss would not reflect accurately whether the generator did a good job creating plausible images. Thus, the generator would never converge towards a lower loss. This means that the discriminator has to improve at approximately the same rate as the generator [48, 49].

The discriminator, like any other classifier, can take as input whatever size and shape of data, as long as the architecture compensates for this change of data. For the case of predicting authenticity of MNIST handwritten digits, the entire `(batch_size, 1, 28, 28)` image is given as input. As output, a `(batch_size, 1)` vector is given. This vector is equivalent to the discriminator outputting a real or fake value for each image in the batch (0 being fake and 1 being real). The model contains six linear layers with an architecture made to be that of an autoencoder (see figure 13). This means that the two first layers create deep features with lower dimension and the next four layers broaden the dimensions again while lowering the depth. Dropout with probability 0.15 is used to avoid overfitting and too fast learning, while leaky-ReLU is used for the first four layers. The sigmoid is used for the last layer to restrict the output values to be between 0 and 1, such that they mimic the probability of the input image being real [50]. The architecture of the discriminator model is visualized in table 2.

Layer	Activation Size	Value	# Parameters
Input	28x28		0
linear layer	32x32		803,840
Dropout	32x32	0.15	0
LeakyReLU	32x32	0.2	0
linear layer	2048		2,099,200
Dropout	2048	0.15	0
LeakyReLU	2048	0.2	0
linear layer	32x32		2,098,176
Dropout	32x32	0.15	0
LeakyReLU	32x32	0.2	0
linear layer	512		524,800
Dropout	512	0.15	0
LeakyReLU	512	0.2	0
linear layer	16x16		131,328
Dropout	16x16	0.15	0
linear layer	1		257
Sigmoid	1	NA	0
Total Parameters			5,657,601
Trainable Parameters			5,657,601

Table 2: All layers used in the final generator and the number of needed parameters. For a detailed explanation, see table 1.

Regarding the learning of the discriminator, one real and one fake image is first given as input. This gives the discriminator both positive and negative examples during training, which it tries to classify as real or fake. The discriminator performance is found using a loss function and this loss quantifies how well the discriminator predicted the validity of the images. This loss is high if the model classifies real images as fake or visa versa. Then, the discriminator weights are updated by backpropagating the loss thoughout the discriminator network. The generator weights are not updated in this step, as it needs to stay constant while generating examples of training images [48, 49].

4.1.3 Loss

For implementing the loss functions necessary to get a proper GAN going, first, some nomenclature needs to define . We assume that sample images are sampled from $x \sim p_{\text{data}}(x)$. The discriminator $D(x)$ outputs a value between zero and one determining whether x is real or fake. The generator takes random noise $z \sim p_{\text{data}}(z)$ and returns a fake sample. The loss function for the discriminator is given as shown below.

$$\begin{aligned}\mathcal{L}(G, D, X, Z)_D = &\frac{1}{2}\mathbb{E}_{x\sim p_x}[1 - D(x)] \\ &+ \frac{1}{2}\mathbb{E}_{z\sim p_z}[D(G(z))]\end{aligned}$$

Here, the first term simply forces the discriminator to try to return large values when the samples are real. As the discriminator output approaches 1, the loss approaches 0. The second term is slightly more complicated. The generated image $G(z)$ is given to the discriminator, and the goal of the discriminator is to catch that this is, in fact, a fake image. Thus, lower discriminator values on the fake image leads to lower loss and as the discriminator output approaches 0, this term of the loss also approaches 0.

Now for the generator loss, which is shown below.

$$\mathcal{L}(G, D, X, Z)_G = -1 \cdot \log \sigma(D(G(Z)))$$

This loss is equivalent to the binary cross-entropy (BCE) loss with logits, where the weights are all set to be equal. The σ term is the sigmoid function. This loss function basically states that the generator gets a large loss when the discriminator gets tricked into thinking that the generated image is real. This is due to the fact that a low probability of it being a real image leads to a high loss and vice versa [17].

4.2 CycleGAN

The GAN allowed for new images in a specific category to be created based on random noise. The next upgrade would be to be able to go from one category to another and back again. This is exactly the problem a CycleGAN aims to fix, which makes it perfect for this paper, since the aim is to go from 3T images to 1.5T images. Going back is not as important in this case, but could prove useful for other purposes. The CycleGAN generally works by implementing two generator networks, each generating one category based on an image from the other category, and two discriminator networks, each

predicting the probability of the input image actually being from the original category. This architecture is explained and showcased further in figure 10 and 11 below.

4.2.1 General Model Architecture

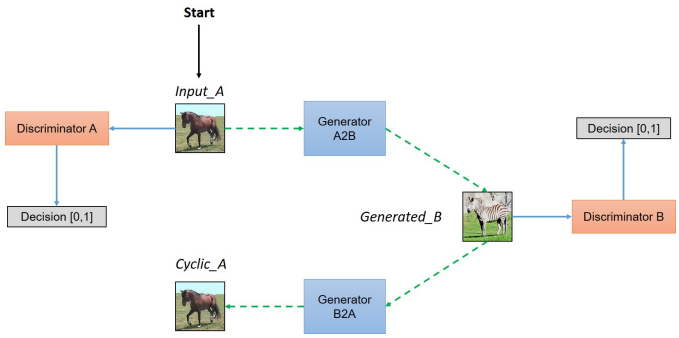


Figure 10: This figure shows the interactions necessary for the four different networks to convert images from A to B. The input image of category A is fed directly to the discriminator and to the generator that converts images from A to B. This generates a fake image from category B, which is fed both to the discriminator to get a prediction and to the other generator to recover the image to one of category A for reasons that will become apparent later in this section. Figure courtesy of [45].

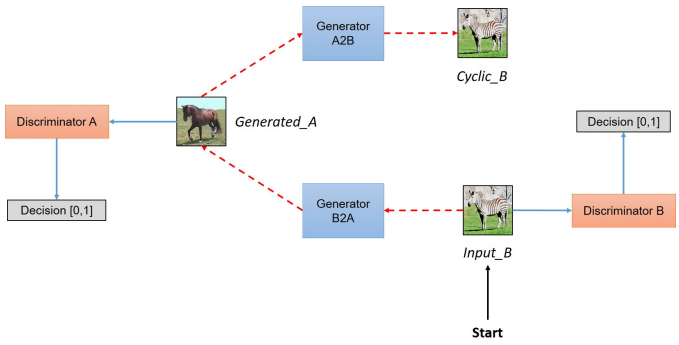


Figure 11: This figure is almost identical to figure 10, although it shows the interactions necessary for the four different networks to convert images from B to A. For a basic explanation of the components involved, see the caption of figure 10. Figure courtesy of [45].

4.2.2 Discriminator in a CycleGAN

The discriminator of the model should take as input an image of size 1x256x256 (since the image is grayscale) and output a prediction of dimension 1x30x30 with values between zero and one. The reason the output is not just one value is because a patch-GAN is used. This move in dimensions is accomplished using up-sampling of the image with 2D-convolutions to extract increasingly abstract features by decreasing the height and width and increasing the depth of the image.

To make sure that the model is as robust as possible and to avoid overfitting, a patchGAN is used before outputting the final image prediction. This works by converting 30x30 blocks of the image into predictions, such that the model is forced to predict the validity of the image on each of these blocks. This ensures that the discriminator image can accurately predict if the image comes from the generator by using any part of the image. Some problems can occur with this method, like the model not being about to predict much from a 30x30 square of black pixels. These kinds of problems will be discovered further in the section 6 on discussing problems with and efficacies of the model.

Of just as much importance as the patchGAN architecture and general model interactions is the layout of each layer and the functions used throughout the discriminator network. A general overview of the layers, kernel size, type of convolution, stride and padding is shown below in figure 12.

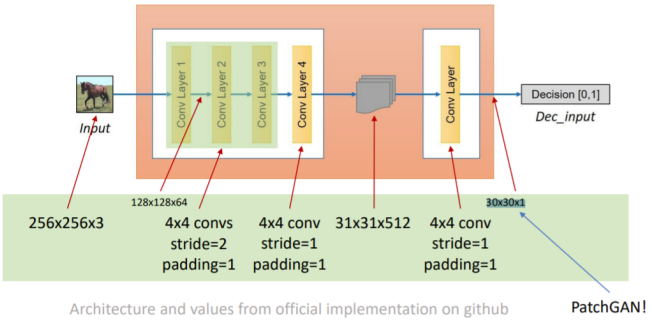


Figure 12: The primary model layout consisting of the input, layers and output. The input is the gray-scale intensity-based MRI input of size 1x256x256. The five consecutive convolutional layers each have a specified kernel size, stride and padding as noted in the figure. The output is constructed from running the patchGAN on the last layer of the model to get a 1x30x30 matrix of values between 0 and 1. The specific values and layer sizes are specifed in table 3. Figure courtesy of [51].

Not shown in the figure is the actions taken between each layer and the number of parameters used for each action. This can be seen below.

Layer	Activation Size	Value	# Parameters
Input	3x256x256		0
64x4x4 conv, stride 2, pad 1	64x128x128		1,088
LeakyReLU	64x128x128	0.2	0
128x4x4 conv, stride 2, pad 1	128x64x64		131,200
InstanceNorm2d	128x64x64	64	0
LeakyReLU	128x64x64	0.2	0
256x4x4 conv, stride 2, pad 1	256x32x32		524,544
InstanceNorm2d	256x32x32	64	0
LeakyReLU	256x32x32	0.2	0
512x4x4 conv, stride 1, pad 1	512x31x31		2,097,664
InstanceNorm2d	512x31x31	64	0
LeakyReLU	512x31x31	0.2	0
1x4x4 conv, stride 1, pad 1	1x30x30		8,193
Sigmoid	1x30x30	NA	0
Total Parameters			2,762,689
Trainable Parameters			2,762,689

Table 3: The convolutions, stride and padding of each layer in the discriminators. The values used in the activation functions and instance normalization as well as when they are used in the network have also been shown. The activation shape channels are equal to the output shape of each layer and the height and width are calculated using the formula from article [52]. For the convolutional layers, the convention in the "layer" column reads `output_size x kernel_height x kernel_width`. The first layer thus outputs a vector with 64 channels after applying a convolution with a 4x4 kernel with a stride of 2 and padding of 1. Other details of how to read the table can be found in table 1.

4.2.3 Generator in a CycleGAN

Unlike in a standard GAN generator, where a random image from a latent space is used to create what ends up looking like a real image, the generators in the CycleGAN takes as input an image from one of the datasets. As such, the input dimension is 1x256x256 (since the image is grayscale). The output is of the exact same dimension, since the generator tries to generate an

image from the opposite category using the input image. Between input and output, the general generator architecture is that of an autoencoder, as pictured below in figure 13. The generator contains three layers in an encoder and a decoder, while 9 residual blocks reside between them. The encoder encodes features by narrowing the image dimensions down and widening the depth of the data. The decoder does exactly the opposite, extracting newfound information back into a new image. The residual blocks between the encoder and decoder serve the purpose of interpreting the encoded features. They also use skip connections, which provide an alternate path for the gradient. This helps the network reach convergence.

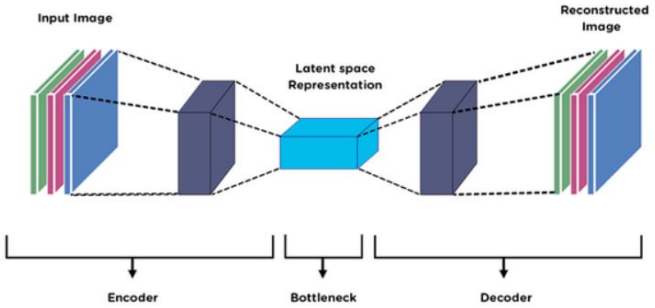


Figure 13: The general autoencoder architecture. A kernel does convolutions to reduce and increase the image dimensions. The smaller the dimensions, the larger the feature complexity. The code between the encoder and decoder is referred to as residual blocks. These learn residual functions from the input of their last layer instead of learning unreferenced functions [33]. Figure courtesy of [46].

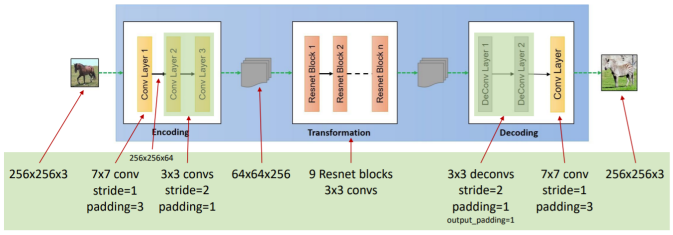


Figure 14: The primary model layout consisting of the input, layers and output. The input is the gray-scale intensity-based MRI input of size 1x256x256. The five encoding and decoding convolutional layers as well as the residual blocks each have a specified kernel size, stride and padding as noted in the figure. The specific values and layer sizes are specifed in table 4. Figure courtesy of [51].

Layer	Activation Size	Value	# Parameters
Input	3x256x256		0
64x6x6 conv, stride 1, pad 3	64x257x257		6,976
ReLU	64x257x257	inplace=True	0
128x3x3 conv, stride 2, pad 1	128x129x129		73,856
InstanceNorm2d	128x129x129	128	0
ReLU	128x129x129	inplace=True	0
256x3x3 conv, stride 2, pad 1	256x65x65		295,168
InstanceNorm2d	256x65x65	256	0
ReLU	256x65x65	inplace=True	0
9 residual blocks with relu and instancenorm	256x65x65		10,621,440
Upsample	256x130x130	2,bilinear	0
128x4x4 Conv2d stride 1, pad 2	128x131x131		524,416
InstanceNorm2d	128x131x131	128	0
ReLU	128x131x131	inplace=True	0
Upsample	128x262x262	2,bilinear	0
64x4x4 Conv2d stride 1, pad 1	64x261x261		131,136
InstanceNorm2d	64x261x261	64	0
ReLU	64x261x261	inplace=True	0
3x8x8 conv, stride 1, pad 1	3x256x256		12,291
Tanh	3x256x256	NA	0
Total Parameters			11,665,283
Trainable Parameters			11,665,283

Table 4: The convolutions, stride and padding of each layer in the generators. The values used in the activation functions and instance normalization as well as when they are used in the network have also been shown. Relevant details of how to read the table are found in the captions of tables 1 and 3.

4.2.4 Objective Functions (Losses)

For this section, a number of notations will be introduced. Since the goal is to map between the two domains X and Y, $G : X \rightarrow Y$ and $F : Y \rightarrow X$ is defined. Furthermore, $y \sim p_{\text{data}}(y)$ and $x \sim p_{\text{data}}(x)$ refer to the distributions of each domain and x and y refer to samples from each domain. Lastly, the discriminators for each domain are given by D_X and D_Y [35].

This section will focus on how the objective functions are constructed

to ensure the correct result from the discriminators and generators. In general, the losses are separated into three categories: adversarial losses, cycle-consistency losses and identity losses.

Here, the adversarial losses aim to ensure that discriminators are able to distinguish between real and fake images and generators are able to generate convincing fake images. The cycle-consistency losses make sure that mapping an image through the two different generators one after the other will result in approximately the same image as the input. Lastly, the identity losses check that the generators do not augment images that are already part of the domain they map towards. More details on each of these losses can be seen in the rest of this section.

The final loss function, containing all the losses mentioned above, will look as shown below.

Our full objective is:

$$\mathcal{L}(G, F, D_X, D_Y) = \mathcal{L}_{\text{GAN}}(G, D_Y, X, Y) + \mathcal{L}_{\text{GAN}}(F, D_X, Y, X) \quad + \lambda_{cyc}\mathcal{L}_{\text{cyc}}(G, F) + \lambda_{iden}\mathcal{L}_{\text{iden}}(G, F)$$

Here, $\mathcal{L}_{\text{GAN}}(G, D_Y, X, Y)$ and $\mathcal{L}_{\text{GAN}}(F, D_X, Y, X)$ are the adversarial losses for G and D_Y and F and D_X respectively. Meanwhile, $\mathcal{L}_{\text{cyc}}(G, F)$ is the cyclic loss for both generators and $\mathcal{L}_{\text{iden}}(G, F)$ is the identity loss for both generators. The values λ_{cyc} and λ_{iden} are hyperparameter weights, controlling the respective importance of each part of the total loss function.

The goal of the final loss function is now to have the generators G and F minimize the function and have the discriminators D_X and D_Y maximize the function. This leads to competition between the four models. This interplay can be seen below.

$$G^*, F^* = \arg\min_{G,F} \max_{D_x, D_Y} \mathcal{L}(G, F, D_X, D_Y). \tag{1}$$

The adversarial losses, shown below, are principally the simplest of the losses to understand. They simply make sure that the discriminators are very sure that real images are real and fake images are fake, while the generator becomes increasingly better at fooling the discriminator (creating convincing fake images). Modeling this behavior into an equation is simple, since the discriminator D_Y approaches minimal adversarial loss as it gets closer to predicting 1 for real images and 0 for generated images. Meanwhile, the opposite is true for the generator. Since the discriminators want to maximize the below equations and the generators want to minimize them, these models have an antagonistic relationship and constantly need to sacrifice the performance of the other model to get better performance

itself. This has been shown countless times in the literature to be fruitful and is the basis of GAN-based techniques.

$$\mathcal{L}_{\text{GAN}}(G, D_Y, X, Y) = \mathbb{E}_{y \sim p_{\text{data}}(y)}[\log D_Y(y)] + \mathbb{E}_{x \sim p_{\text{data}}(x)}[\log(1 - D_Y(G(x))]$$

$$\mathcal{L}_{\text{GAN}}(F, D_X, X, Y) = \mathbb{E}_{x \sim p_{\text{data}}(x)}[\log D_X(x)] + \mathbb{E}_{y \sim p_{\text{data}}(y)}[\log(1 - D_X(F(y))]$$

It turns out that the above adversarial losses are not always strenuous enough to map the input x_i to an output y_i. Therefore, another set of loss functions are added; the forward and backward cycle-consistency loss functions, shown under this paragraph. The goal of these functions is to make sure that an image from X mapped to Y and mapped back to X again will closely resemble the original image. That is, $x \rightarrow G(x) \rightarrow F(G(x)) \approx x$.This concept is called forward cycle-consistency loss, while the same is true for the other direction, backward cycle-consistency loss: $y \rightarrow F(y) \rightarrow G(F(y)) \approx y$. As can be seen in the two equations below, the L1-loss is taken between the original image x and the double-generated image $F(G(x))$. The generators G and F then aim to minimize these terms, such that the two images are as closely resembling as possible. This loss is focused on the generators specifically, since the discriminators only play a roll in predicting if the image is real.

$$\mathcal{L}_{\text{cyc}}(G, F) = \mathbb{E}_{x \sim p_{\text{data}}(x)}[||F(G(x)) - x||_1] + \mathbb{E}_{y \sim p_{\text{dau}}(y)}[||G(F(y)) - y||_1]$$

The identity loss formulated below punishes the model if the generator changes an image that is already part of the domain it transforms to. That is, $y_{new} = G(y)$ gets zero identity loss if y_{new} and y are completely identical and high loss if there are vast differences. The identity loss usually keeps the generators from changing the colors of the input image to get lower errors.

$$\mathcal{L}_{\text{iden}}(G, F) = \mathbb{E}_{x \sim p_{\text{data}}(x)}[||F(y) - x||_1] + \mathbb{E}_{y \sim p_{\text{data}}(y)}[||G(x) - y||_1]$$

4.2.5 Optimizer, Cross Entropy, Etc.

A table of the used hyperparameters and their values in this project in given below. Note that these hyperparameters are applied to different models. Kernel size, stride and padding for each of the convolutional layers of the discriminator and generator are shown in tables 1, 2, 3 and 4.

Hyperparameter	Value
General	
Batch size	1
λ_{iden}	1
λ_{cyc}	10
Optimizer	Adam
Betas	(0.5, 0.999)
Discriminator	
input dimension	$1x256x256$
Conv depths	[3, 64, 128, 256, 512]
Output dimension	$1x30x30$
Leaky-ReLU activation	0.2
Generator	
Input dimension	$1x256x256$
Conv depths	[64, 128, 256, 128, 64]
Output dimension	$1x256x256$
XY Model	
Discriminator learning rate	$5e-8$
Generator learning rate	$1e-7$
Number of epochs	1.5
Number of steps	240000
XZ Model	
Discriminator learning rate	$5e-8$
Generator learning rate	$1e-7$
Number of epochs	2
Number of steps	265000
YZ Model	
Discriminator learning rate	$1e-7$
Generator learning rate	$1e-7$
Number of epochs	1.4
Number of steps	176000
ALL Model	
Discriminator learning rate	$7e-8$
Generator learning rate	$1e-7$
Number of epochs	0.5
Number of steps	245000

The optimizer, batch size, betas, convolution depths and input/output were set to the same value as in the original CycleGAN paper, as this turned out to be efficacious on ADNI images too. Most notably, the number of epochs has been drastically reduced (from 200 in the original paper to under two in most of the models). This is due to the fact that the training data used in this project is much more plentiful than originally used to train the first CycleGAN in the paper. This means that fewer epochs can be used, since the same number of overall steps is achieved. The weight λ_{iden} was set to 1 in this project, since it gave better performance than the 0 value from the CycleGAN paper.

In relation to the discriminator and generator convolution depths, it is self-explanatory that using other images (RGB, non-square, different dimension) would require changing the architecture of input and output dimensions of all four models while also ensuring that the kernel size, stride and padding of the convolutional layers are setup correctly. This can be double checked by using $output\_dim = n - k + p + 1$, where n is the image dimension, k is kernel size and p is padding.

4.2.6 Troubleshooting

Since implementing a CycleGAN requires implementing four different neural networks, the hyperparameter search and minutia of the model can be difficult to get right. Many issues like non-convergence, mode collapse, diminished gradients, vanishing gradients and more are well documented [41]. A known problem is CycleGAN only training until they reach a local minimum [43]. Another is the issue of getting noisy squares in images, sometimes referred to as checkerboard noise. One of the first models that was trained on the MRI ADNI images turned out to contain a version of this checkerboard noise. The source of noise turned out to be an uneven sampling of pixels in earlier layers to newer layers when doing the upsampling (also known as transpose convolution or deconvolution). Mismatches between the kernel size and the stride as well as bad upscaling can cause some areas of the upsampled image to contain information from more pixels than other areas. This is a critical cause of the recognizable checkerboard noise [44]. An illustration of how the flaw gets made can be seen below. images generated from the flawed CycleGAN can also be seen in illustration 16a. Here, it is clear that white squares are surrounded by black squares all over the image, worsening the general image quality.

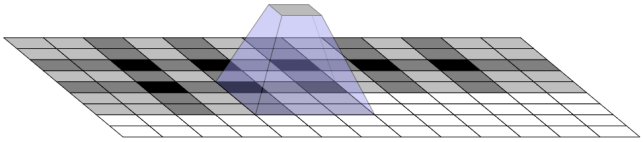
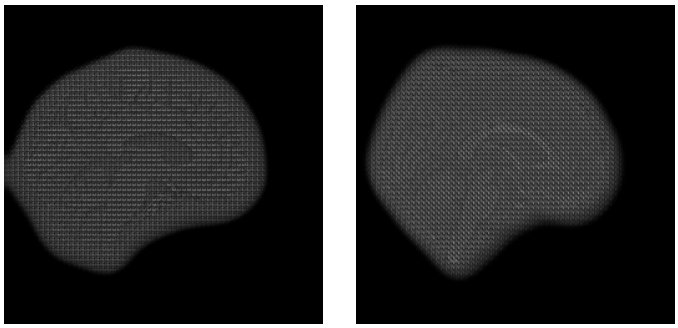


Figure 15: When the kernel size is not divisible by the stride, the image gets higher intensity in some areas than others leading to a checkerboard pattern. Shown here is a deconvolution using kernel size of 3 and stride of 2. Figure courtesy of [44].

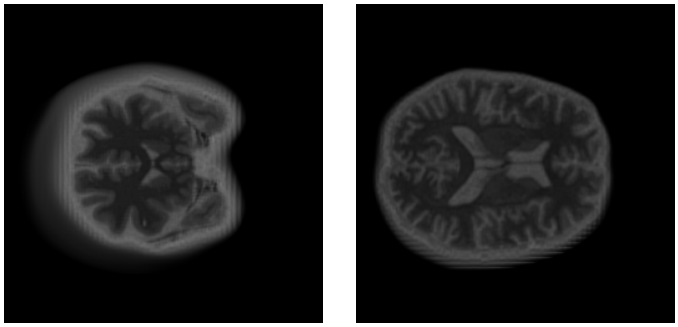
Solving this problem in practice first required separating upsampling from convolution. This works since the more intensity between pixels is split between multiple pixels. Doing this in practice involved switching every deconvolution layer except the last one from `nn.ConvTranspose2d(in_channels, out_channels, **kwargs)` to first `nn.Upsample(scale_factor=2, mode='bilinear', align_corners=True)` and then `nn.Conv2d(in_channels, out_channels, **kwargs)`. In a nutshell, instead of running a transpose convolution, an upsampling is perform to double the size of the image using bilinear interpolation and then a standard convolution is performed. This not only solves most of the checkerboard issues but also ensures that the model is able to learn from the decoding, since (`nn.Conv2d` and `nn.ConvTranspose2d` have trainable kernels while `nn.Upsample` does not) [18]. After implementing this functionality, the generated images now look like shown in illustration 16b. We now see that most of the checkerboard artifacts are gone, though some vertical and horizontal lines still persist. A weird white void has also been generated around about half of the images for some reason.

The slight lines in the generated images after using bilinear interpolation for upsampling are not nearly as problematic but are still unfortunate as a final output. To solve this, we implemented a kernel size that was divisible by the stride in the deconvolution layers, as was mentioned as a solution in the previous paragraph. This should make the kernel spread information from each pixel more evenly between new pixels and supply a less grainy final image [44]. In illustration 17a, most of the noise has finally been removed and no lines persist in the image. The white voids around the brain seems to have become even more prevalent though, which also needs to be fixed. This was taken care of by replacing bilinear interpolation with nearest-neighbor interpolation. The result of replacing the interpolation

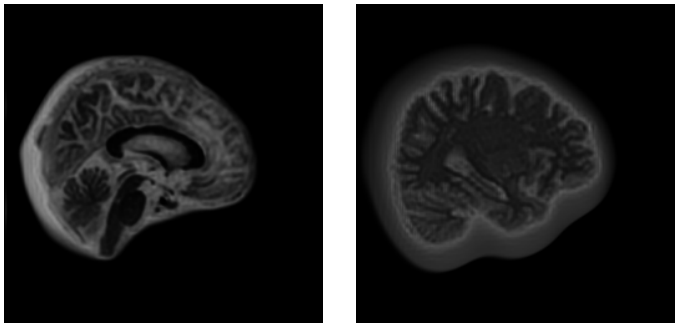
can be seen in 17b. It is now evident that basically no noise is left in the image, and most of the troubleshooting relating to image quality has thus subsided. The full comparison between results after each improvement has been implemented, can be seen in figure 17.



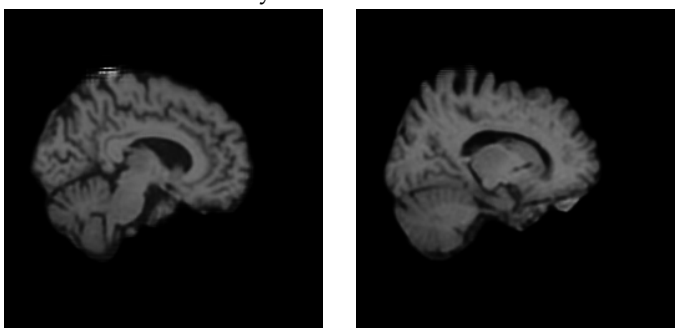
(a) 1.5T and 3T respectively YZ MRI images from the preliminary run.



(b) 1.5T and 3T respectively YZ MRI images after bilinear interpolation has been implemented in upsampling.



(a) 1.5T and 3T respectively YZ MRI images after making kernel size divisible by stride.



(b) 1.5T and 3T respectively YZ MRI images after bilinear interpolation has been changed to nearest neighbor interpolation.

Figure 17: Comparison of troubleshooting efforts to solve the visual issues in the generated images from the generators.

Even after the struggle of implementing all of the these changes, some challenges are still present when generating images, though at this point the severity of the problems has been significantly reduced. It turns out that very rarely, images that are converted from $3T$ to $1.5T$ get a coupe horizontal white lines in the top of the image. This artifact can be seen in figure 18 for a random subject for each field-strength, where the above explained case manifested itself. This artifact seemed to be so rare that it was deemed unimportant to fix in this project.

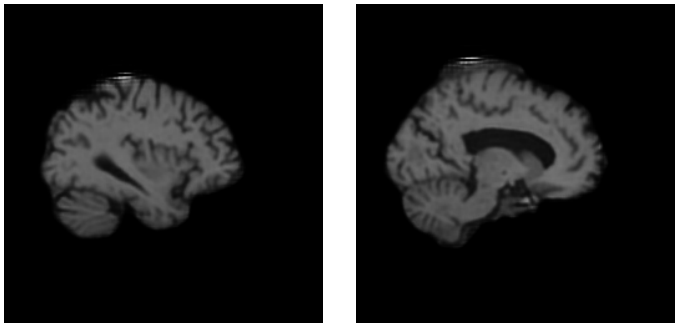


Figure 18: Two generated images before training was completed (half an epoch and one and a half epochs respectively). The left image is 1.5T and the right is 3T. The image artifact occurs only on the top of the 1.5*T* images, where a trading pattern of low-intensity and high-intensity pixels can be seen forming small horizontal lines.

Another issue entirely turned out to be that the vast number of input images all mapped to the same output image. This problem is well-known and is commonly called mode collapse. Figuring out that the models suffered from mode collapse normally required either plotting loss curves and seeing oscillations in the generator loss or sampling a number of images using different input images. In the generator models from this paper, no signs of mode collapse could be seen in the loss curves except for the discriminator constantly having very low loss. It was very evident that the generator only created one image when given multiple input images though [38].

Many solution to the problem of mode collapse exist, some being much easier to implement than other. Easier solutions include reducing the learning rate, making separate learning rates for the discriminators and generators and only updating the discriminator if the accuracy of the discriminator in the current step is under 50%. Harder solutions include implementing instance noise, making the generator better or making the discriminator worse. For the discriminators in this project, it turned out that it was enough to implement the accuracy check before updating the discriminators, reducing the learning rate and making separate learning rates for generators and discriminators. The others methods have been left in this section for future reference [39, 40]. After implementing these fixes, the model was able to generate output samples from a much larger distribution, which indicates that the mode collapse has been fixed. A number of generated images can be seen in section 5.

4.3 Reconstruction of 3D images

The reconstruction of 3D images was done by predicting each 2D slice by a CycleGAN model. Thus, the 3D image was reconstructed from generating ~ 768 2D images, and then re-stitching these together[3]. The ALL model will predict on all three planes, which in practice means that the output slice is an average of 3 generated slices. The reconstruction is done in python and the code can be seen here, GitHub.

4.4 CycleGAN Evaluation Metrics

To be able to properly evaluate whether the CycleGAN models have learned to effectively move between 1.5T and 3T images, a number of evaluation metrics have been applied. This section will explain the background behind these as well as what is expected for bad and good performance when using each metric. The data used for this project contains 40 subject with both a 1.5T scan and 3T scan. This enabled a perceived ground truth to be calculated, that could be used to quantify the CycleGAN model.

4.4.1 Visual Inspection

Visual inspection was done by showing the generated images in a figure next to the ground truth. Visual inspections was used to evaluate if the generated image looked like a brain, and secondly if the generated image looked like the brain it was supposed to be mapping towards. This metric was important, since the first step towards generating convincing brains was to make them look convincing to humans.

4.4.2 Image Subtraction

Another way to visually inspect the images is to subtract images from each other and look at the resulting image. In this report, image subtraction had been used between the 1.5T ground truth and 3T ground truth images as well as between the 1.5T ground truth and 1.5T generated images. This allowed for comparison between how well the 1.5T generated image and the 3T ground truth images fit onto the 1.5T ground truth image. Images that were closer to matching would have fewer high-intensity pixels and a

[3]Images which contain nothing but black pixels around the edges are not predicted by the model since this is unnecessary. Thus, less than 768 generations for each image happens at time of reconstruction.

generally darker image. To enhance the visibility, histogram stretching and gamma transformation was applied.

4.4.3 Root-Mean-Square Error

Another way to show how close the generated image was to the real image was to calculate the root-mean-square error between every pixel in the two images. This is done by applying equation (2).

$$RMSE = \sqrt{\frac{1}{n}\Sigma_{i=1}^{n}\left(y_i - \hat{y}_i\right)^2} \tag{2}$$

where n is the total number of pixels, y_i is the real image and $\hat{y}_i$ is the fake image. RMSE was found for two three of images: 1.5T ground truth & 3T ground truth images, between the 1.5T ground truth & 1.5T generated images and between the 3T ground truth & 3T generated images. A smaller difference between the generated and ground truth image than between the ground truth set would mean that the model had, to some extent, learned to replicate the patterns seen in 1.5T images.

4.4.4 Geometric Accuracy

The geometric accuracy is another metric for determining how close two image are to matching. As an accuracy measure, this method calculates how high a percentage of two images contain the same pixel values by doing a pixel-wise comparison.

$$\text{Geometric accuracy} = \frac{\Sigma_{i=1}^{n}\mathbb{1}}{n}, \qquad \mathbb{1} = \begin{cases} 1, & \text{if } y_i = \hat{y}_i. \\ 0, & \text{otherwise.} \end{cases}$$

Here, n is the total number of pixels that will be compared in the image. As such, a higher accuracy means that the model was more successful in creating the real 1.5T image. It is only possible to calculate this metric on the image data, because values have been rounded to `uint8` integer values.

4.5 Classifier

In order to investigate whether a CycleGAN could be used to remove bias from a classifier, this project introduced a Convolutional Neural Network (CNN), inspired heavily by Basai et al. and Camilla Kergel [20, 25]. This

section will present the architecture of the CNN, the applied hyperparameters as well as some highlighted differences from the other papers. The project implemented two different classifiers, referred to as $\mathcal{M}_1$ and $\mathcal{M}_2$. The model, $\mathcal{M}_1$ was trained on synthetic data from the CycleGAN model and original data[4]. Whereas, the model $\mathcal{M}_2$ was trained purely on original data (both 3T and 1.5T), to recreate the field strength bias.

4.5.1 Neural Network for Alzheimer's Disease Classification

In recent work with Alzheimer's disease classification by Camilla, a neural network was trained to predict whether patients had AD and the largest biases were found. Since the largest bias found in the model was between 1.5T and 3T images, a similar model was used to compare and contrast if the above described methods succeeded in removing the most prominent biases from the model, namely, the difference between 1.5T and 3T images. The model was implemented as a PyTorch neural network with 6 convolution layers using ReLU as activation function between each layer. This was followed by a fully connected layer with sigmoid activation to output the probability of each class (sick or healthy). [20].

4.5.2 Data

The data used for training the classifier is a combination of generated images from a CycleGAN[5] model as well as the original data from ADNI, which has been described in section 3.2. By training a classifier on the data, it was tested whether the model would inherit a bias towards the generated images. The goal was to prove that this bias does not exist by using dimensionality reduction algorithms on the penultimate linear layer from the model. There were 939 images in the training data and 235 in the test data. However, augmentation was used to create a more robust model. Three types of augmentations happened randomly with a probability, $p = 0.8$. The augmentations used were a random rotation, random flip or a random elastic deformation. The goal of using augmentation was to both create more training data and ensure a robust and generalization model after training.

[4]The training data then consisted of 1.5T from ADNI & and synthetic 1.5T images constructed from 3T images.

[5]Important to note that only the CycleGAN trained on xy plane is used to generate the images used for training the classifier. This will be touched upon again in the discussion, see section 7.2.

4.5.3 Deep Neural Network Model

The implemented network is a convolutional neural network (CNN) which is a special type of deep neural network. The name stems from the most important operation in the network, which is convolutions. In short, a convolution is the process of convolving a kernel with the image. An example of convolutions and a simple step in the convolution between an image named input and a horizontal Sobel filter can be seen in figure 19.

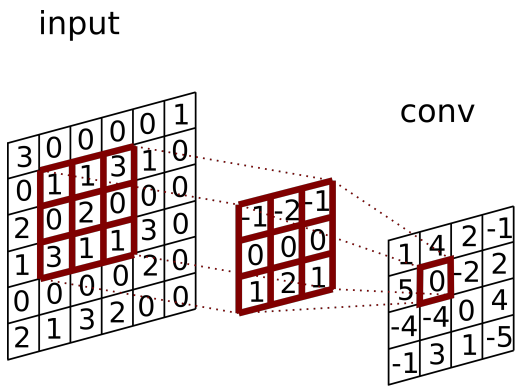


Figure 19: The input is an image with dimensions 6x6, corresponding to 36 pixels. The kernel is a famous filter called the Sobel filter and is used for edge detection.

In figure 19, the kernel is simply a 3x3 matrix:

$$\text{Kernel} = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

One convolution corresponds to multiplying the matrix with every possible space in the 6x6 input image. The stride in this example is one, which refers to how the kernel is moved across the image. If no padding is applied to the input image the dimension of the output image after the convolution will be smaller. In this example the image will shrink from 6x6 to 4x4. To put it simply, the kernel is slid across the width and height of the input and the dot products between the input and kernel are computed at every spatial position [67].

The kernels in a convolutional neural network are learnable parameters.

The dimension of the kernels are smaller than the input, which in turn will reduce the dimension of the input image. All padding are set to zero, which can also be seen in the source code. In this project, the three main type of layers in the network model were:

- Convolutional Layers
- ReLU to ensure non-linearity
- Classification in terms of Fully Connected Layers

The classifier in this project was trained on 80% of the available data and was tested on the last 20%. As for hyperparameters consult following table.

Hyperparameter	**Value**
Train-test split	0.8/0.2
Dropout [p]	0.15
Learning rate [α]	0.0002
Epochs	230

The optimization algorithm used in the classifier was Adaptive Moment Estimation (Adam). This optimizer utilized adaptive learning rate and step size based on the momentum. The activation function used was ReLU, which is defined as $y = \max(0,x)$. ReLU ensured non-linearity in the model, making the parameters more expressible and making the model more generalizable and robust. Sigmoid was used on the last layer to ensure that the output value was a vector between zero and one. Cross-entropy loss was used as the cost function and punished the model, if the prediction was far from the label [68]. The loss was calculated using:

$$L\left(\boldsymbol{y}_i, \hat{\boldsymbol{y}}_i\right) = -(\mathbf{y}\log(\hat{\mathbf{y}}) + (1-\mathbf{y})\log(1-\hat{\mathbf{y}}))$$

The following table will outline the architecture of the model. From the table, each layer with its respective size and trainable parameters will be shown.

Layer	Activation Size	Value	# Parameters
Input	3x256x256		0
32x3x3x3 Conv3d, stride 1, pad 0	32x254x143x119		896
ReLU	32x254x143x119	inplace=False	0
64x3x3x3 Conv3d, stride 2, pad 0	64x126x71x59		55,360
ReLU	64x126x71x59	inplace=False	0
128x3x3x3 Conv3d, stride 1, pad 0	128x124x69x57		221,312
ReLU	128x124x69x57	inplace=False	0
256x3x3x3 Conv3d, stride 2, pad 0	256x61x34x28		884,992
ReLU	256x61x34x28	inplace=False	0
512x5x5x5 Conv3d, stride 1, pad 0	512x57x30x24		16,384,512
ReLU	512x57x30x24	inplace=False	0
1024x3x3x3 Conv3d, stride 2, pad 0	1024x28x14x11		14,156,800
ReLU	1024x28x14x11	inplace=False	0
Linear	128		565,182,592
ReLU	128	inplace=False	0
LayerNorm	128		256
Dropout	128	$p = 0.15$	0
Linear	2		258
Total Parameters			596,886,978
Trainable Parameters			596,886,978

Table 5: The convolutions, stride and padding of each layer in the classification model. The values used in the activation functions and instance normalization (LayerNorm) as well as when they were used in the network have also been shown. Other table details are given in the table 1 and 3 captions.

4.5.4 Memory Handling

Due to the pure size of the 3D images, which was 256x256x256 for each 3D image, an enormous amount of memory was required on the GPU to run the network outlined in table 5. For this project, 32 GB GPUs were available, which made it feasible to run and train the model with a batch size of 1 and images cropped to size 256x145x121. The reason why cropping was used was due to the amount of black pixels around the brain itself. Other implementations to extend the amount of available memory on the GPU have been implemented in the source code, such as: `torch.cuda.empty_-`

`cache()`, which empties the cache of the cuda and in turn enables a bit more of memory.

4.6 Dimensionality Reduction: t-SNE & PCA

The amount of input features or shape of an input for a model is often referred as the dimensionality of the dataset or variable. The end goal of dimensionality reduction is to reduce the number of features (dimensions) in a given dataset, while keeping as much of the variation in the dataset as possible. This is very useful to visualize high dimensional features in a human interpretable space such as 2D or 3D. Two different dimensionality algorithms were utilized in this project to investigate if a CNN trained on generated images and original images inherit a bias towards either of the image types. The two algorithms are Principal Component Analysis (PCA) and t-Distributed Stochastic Neighbor Embedding (T-SNE).

Principle Component Analysis is a widely used algorithm within the field of machine learning for dimensionality reduction. With large datasets containing many features PCA offers a method that can minimize information loss while increasing the interpretability. This is done by calculating the principle components which is a new set of de-correlated variables that maximizes the variance. A way to calculate the principle component is by using Singular Value Decomposition (SVD), to find the eigenvalues and eigenvectors. Then, the original data can be projected into a space with fewer dimensions. Furthermore, PCA can be used to see how much variance is explained by each of the principle components in a dataset. Thus, PCA is useful for both visualizing and intepreting high dimensional features, but also as a way to reduce the dimension of high dimensional datasets without losing more information than strictly necessary. In this project, PCA was used to project the features of the penultimate linear layer in the CNN to understand and see if the model interpreted 1.5T ground truth and 1.5T generated images differently [69, 70].

t-Distributed Stochastic Neighbor Embedding is another type of dimensionality reduction algorithm, which, much like PCA, can be used to visualize and understand high dimensionality data. The main difference from PCA is that t-SNE creates a space where points that are close are similar in the original space. Thus, t-SNE will attempt to cluster data with local similarities, but de-prioritize keeping large pair-wise distances between samples (where PCA will prioritize separating dissimilar data). From a

high-level overview, t-SNE will compare the similarity of two points in both the high-dimensional and low-dimensional space. Then, it will iteratively optimize these measures by using Kullback-Liebler (KL) divergence as a cost function.

A more detailed description of how t-SNE was calculated is outlined below:

$$p_{i,j} = \frac{\exp(\frac{-||\mathbf{x}_i - \mathbf{x}_j||^2}{2\sigma^2})}{\sum_k \sum_{l \neq k} \exp(\frac{-||\mathbf{x}_k - \mathbf{x}_l||^2}{2\sigma^2})} \tag{3}$$

Each sample was made the center of a gaussion distribution. Then the distance to all other points within this distribution was measured, using the euclidean distance. The enumerator of equation (3) calculated the density of the given point whereas the denominator ensured re-normalization. The variable $p_{i,j}$ is a probability measure of distance between two samples in a high dimensional space. The probability of a point, j within the distribution given the point i, is in practice calculated with the following expression.

$$p_{i|j} = \frac{\exp(\frac{-||\mathbf{x}_i - \mathbf{x}_j||^2}{2\sigma_i^2})}{\sum_{j' \neq i} \exp(\frac{-\left\|\mathbf{x}_i - \mathbf{x}_{j'}\right\|^2}{2\sigma_i^2})}$$

Of most importance here are $p_{i|i} = 1$ and $\sum_j p_{j|i=1}$. As the author of "Visualizing Data using t-SNE", Laurens van der Maaten, wrote in her original paper: The similarity of datapoint x_j to datapoint x_i is the conditional probability, $p_{j|i}$ that x_i would pick x_j as its neighbor if neighbors were picked in proportion to their probability density under a Gaussian centered at x_i [71].

$$p_{ij} = \frac{p_{i|j} + p_{j|i}}{2N} \tag{4}$$

By using equation (4), the conditional probabilities were averaged. This was important because the distance from point x to y might not be equal to the distance from point y to x.

After the above similarities were calculated, each sample in the data was represented in a low dimensional space. The whole goal from here was to learn a dimensional mapping from the similarities in the high dimensional space to the low dimensional space.

$$q_{i,j} = \frac{(1 + \left\|\mathbf{y}_i - \mathbf{y}_j\right\|^2)^{-1}}{\sum_k \sum_{l \neq k} (1 + \left\|\mathbf{y}_i - \mathbf{y}_j\right\|^2)^{-1}} \tag{5}$$

From equation (5), a student-t distribution with 1 as degree of freedom was used to measure the similarities between the low-dimensional points. The location of the samples in the mapping process was optimized using the KL divergence as a cost function. The KL divergence measures the similarity of two distributions, in this case: P and Q.

$$KL(Q||P) = \sum_{i \neq j} p_{ij} \log \frac{p_{ij}}{q_{ij}}$$

The minimization process was done using sklearn, which utilizes gradient decent. The output is a mapping that can visualize the similarities from high dimensional inputs. For this project the input will be a massive linear layer from the CNN. The goal was to map this high-dimensional layer into a 2D-space for each sample in the validation data to detect if a difference between the protected classes, such as the field strength of the input image or if the input is synthetically generated by a CycleGAN, could be seen.

4.7 Reproducibility

This section aims to give the reader a precise insight into how the methods of this project were implemented and how it would be possible to reproduce these results in the future. Hopefully, this will simplify the methods. Reproducibility is an important part of writing any project or performing any research. It shows significance of the given result if it is possible for others at a later time to replicate or reproduce the results, since it validates the result of the original study and shows that it was not a fluke. This section was originally adapted from our project work in 2020, so some similarities in layout and wording might occur [32]. By far the most used programming language in this project was Python Version 3.9 and Torch version 1.10.0 [1, 3]. Furthermore, the versions of the packages used in the project are given in the table below. If running any code from this project, please make sure that your packages either as new or newer than the ones specified below [6]. All code for the project is accessible at `https://github.com/oskarwiese/AlzPred/tree/main/`. Note, that the data preprocessing have been described in section 3.2.2 and the code for this can be seen in appendix[9], consult figure 46 & 47.

[6] Be aware that newer version of packages might have changes that can affect the results of this report. If the goal is to reproduce the results 1:1, please ensure that every package is the exact version as specified in table 6.

Package [2]	Version
albumentations [8]	1.1.0
ipython [9]	7.30.1
matplotlib [6]	3.5.0
nipy [10]	0.5.0
pandas [11]	1.3.5
pelutils [4]	0.6.9
pip-chill [12]	1.0.1
torchio [13]	0.18.71
torchsummary [14]	1.5.1
torch [3]	1.10.0 + *cu*113
torchvision [15]	0.11.1 + *cu*113

Table 6: The packages used for all the scrips in this project as well as their versions. Note that `pip-chill` was only used to generate this table and `ipython` is not strictly necessary, but nice for debugging & testing simple pytorch operations.

Since randomness plays a large roll in implementing deep learning architectures, random seeds have been set in order to ensure equivalent results for every run of every code block. The seed number was set to 42 and was used for the python seed `random.seed`, the numpy seed `np.random.seed` and the pytorch seeds `torch.manual_seed`, `torch.cuda.manual_seed` and `torch.cuda.manual_seed_all`. These seeds were set before any processes run in any of the scrips used.

In order to make the results as reproducible as possible, the data as well as the way the data was obtained also plays a big role. The data originally comes from the ADNI studies[7], but the specific preprocessed data for this project are located in the following folders on the HPC / dtu server: `/dtu-compute/ADNIbias/freesurfer_ADNI1` and `/dtu-compute/ADNIbias/freesurfer_ADNI2`. Furthermore, 3T images of validation subjects are located at: `/dtu-compute/ADNIbias/freesurfer` [8].

There are two main code structures which will be explained in more detail

[7] A more detailed outline of the data is given in section 3.2

[8] All the folders can be found on the DTU HPC server with the given absolute paths.

in the following. These are the scripts that were used to construct and train the CycleGAN and CNN classifier models used in this project.

Below, a thorough walkthrough of the most essential parts of the code for the CycleGAN will be made. The code is split between six different files, the names and short descriptions of which can be seen in table 7.

File	Description
train.py	Main script which runs the training loop and saves the final models
discriminator_model.py	Definition of the discriminator architecture
generator_model.py	Definition of the generator architecture
utils.py	Utility functions like plotting, save/load checkpoint and seeding
data_load.py	Preprocessing of the dataset
plot.py	Script for plotting loss
config.py	Paths and hyperparameter definition

Table 7: CycleGAN code description

The primary file is `train.py`, which is responsible for putting all of the functionality together. This script starts by defining each of the four models (generators/discriminators) as well as the optimizers and losses. Afterwards, it loads in the dataset and starts the training loop. This loop calls the function `train_cykel` where all model interactions and singular losses are immediately calculated. Afterwards, adversarial loss, cycle-consistency loss and identity loss are found and added to get the final loss. This loss is then backpropagated through the model and the next batch is run.

The next two files, `discriminator_model.py` and `generator_model.py` are related in the sense that they both define the general functionality of models used in `train.py`. In `discriminator_model.py`, the `Block` class defines the architecture of using one convolution and applying instance normalization and leaky-ReLU, which becomes useful often. The class `Discriminator` is the workhorse of the script. This class initially defines a layer applying a 2D convolution and leaky-ReLU, after which a number of blocks are added. At the end of the architecture, a last convolutional layer is added and sigmoid is applied to output probabilities. In the script `generator_model.py`, the `ConvBlock` class defines a convolution consisting of a 2D convolution, in-

stance normalization and applying ReLU. The `ResidualBlock` class consists of two `ConvBlock` calls. Both of these classes help to implement the general architecture in the `Generator` class. The `Generator` class has an initializer made up of an initial layer, down-blocks, up-blocks, residual blocks and a last layer. These are called in the forward method, where an initial layer is first added, then the down-blocks, the residual blocks, the up-blocks and at last the final layer. On top of this, tan-h is applied to the model output.

The script `config.py` is very short but extremely important for the general workflow and successful use of the code. It defines the directories of the data and almost all hyperparameters that might need to be changed at any point for any reason when running the code.

The script `utils.py` and `data_load.py` are not as important to the general workflow and are much easier to understand. `utils.py` simply implements functions that save and load a checkpoint (current model and optimizer states) such that training can easily be stopped or the best model can be saved. Functions are also saved to plot the loss curves and add seeds to everything, so randomness does not provide difficulty. `data_load.py` does all the data preprocessing.

Below, a thorough walkthrough of the most essential parts of the code for the CNN classifier will be made. The code is split between six different files, the names and short descriptions of which can be seen in table 7.

File	**Description**
train_classifier.py	Main script which runs the training loop and saves the final models
classifier_model.py	Definition of the classifier architecture
utils.py	Utility functions for augmentation, save/load etc.
data_prep.py	Preprocessing of the dataset
feature_visualization.py	Script for visualizing feature representation.

Table 8: CycleGAN code description

The primary file is `train_classifer.py`, which is responsible for putting all of the functionality together. This script starts by reading the data and turning it into a PyTorch data-loader object. Then, the model is initialized along with the optimizer. Then the training loop starts, which ensures training

of the model interactions and singular losses are immediately calculated after prediction on the training set. Afterwards the loss is backpropagated through the entire network. Once this is done, evaluation mode is turned on and the model is tested on the validation set. Afterwards, loss is calculated. The losses are then appended to a list for visualization of the model performance.

The next file, `classifer_model.py`, defines the general functionality of the CNN classifier models used in `train_classifier.py`. In the script, a class is defined, namely, `ConvNet`, which includes the architecture of the classifier. Six 3D convolutional layers and two linear layers. Furthermore, it includes the forward function, which is used to forward the input through the network.

The script `utils.py` and `data_prep.py` are not as important to the general workflow and are much easier to understand. `utils.py` simply implements functions that save the model (current model and optimizer states) such that training can easily be stopped or the best model can be saved. Functions are also saved to plot the loss curves and augmentation. `data_prep.py` does all the data preparation.

Lastly, `feature_visualization.py` is used to construct 2D-plots of the high dimensional features from the second last linear layer in the CNN model. It creates both PCA and t-SNE plots.

5 Results

This section will focus on relevant outputs of the final, trained models. The first four sections will focus on the various losses. The sections 5.1, 5.2, 5.3 and 5.4 will show respectively the discriminator losses, generator losses, cycle losses and identity losses from each of the four aforementioned models. The training progress of the CycleGAN as well as quality comparisons of generated images can be seen in section 5.5, 5.7, 5.8, 5.9 & 5.10. In section 5.11, 5.12, 5.13 and 5.14 examples of differences between generated images contrasted to real images are shown. Afterwards, section 5.15 will quantify the differences by two evaluation metrics. Following this, section 5.16 will illustrate the losses of training two CNN classifiers on CycleGAN-generated data & original data. Lastly, section 5.17 will show t-SNE and PCA plots from both of the trained classifiers.

5.1 Discriminator Losses

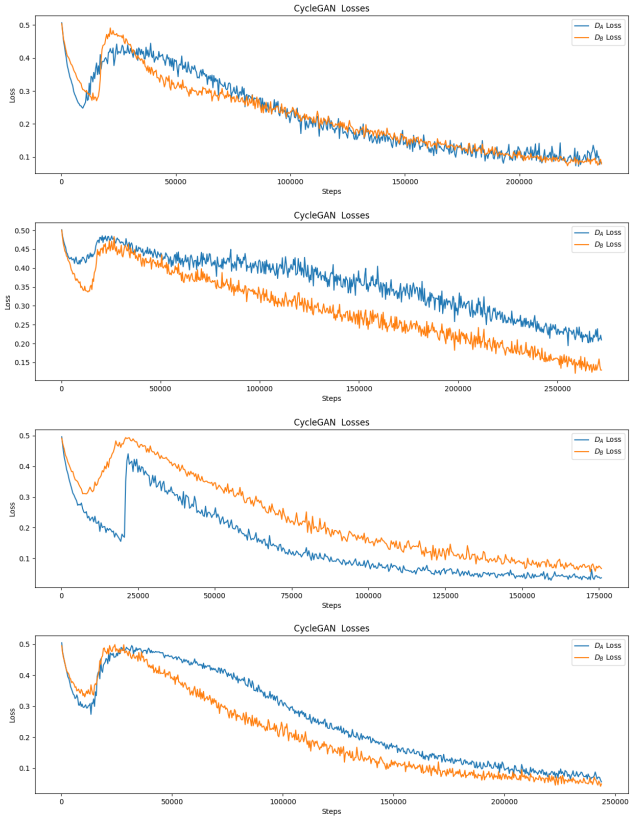


Figure 20: Discriminator losses for respectively the XY, XZ, YZ and the ALL models. D_A tries to predict the validity of real and fake 1.5T images and D_B predicts 3T images. The models have been trained with different parameters and for different periods of time, so this loss is not necessarily useful for gauging model performance. Nevertheless, the figures can be used to determine convergence and signs of learning.

5.2 Generator Losses

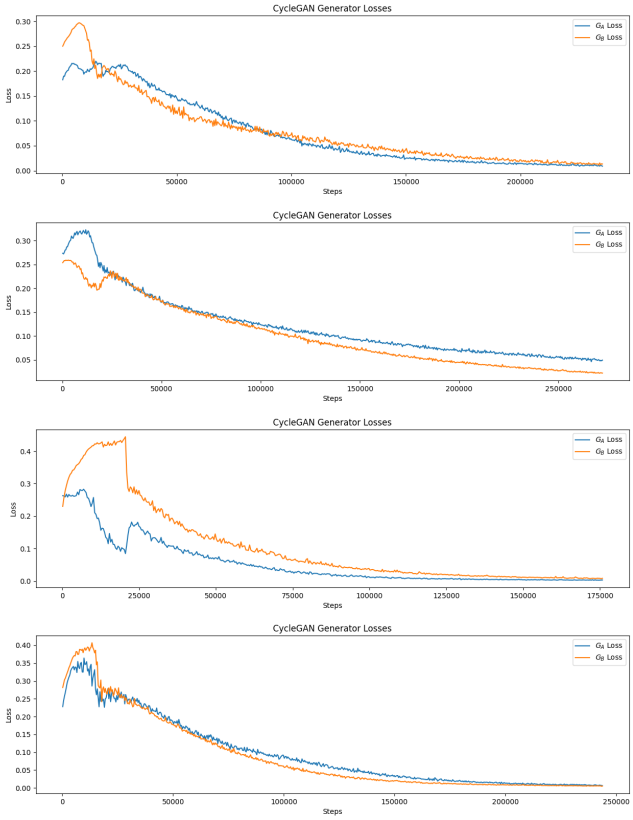


Figure 21: Generator losses for the XY, XZ, YZ and ALL models. G_A converts images from 3T to 1.5T and G_B converts from 1.5T to 3T. Same interpretation applies as described in figure 20.

5.3 Cycle-Consistency Losses

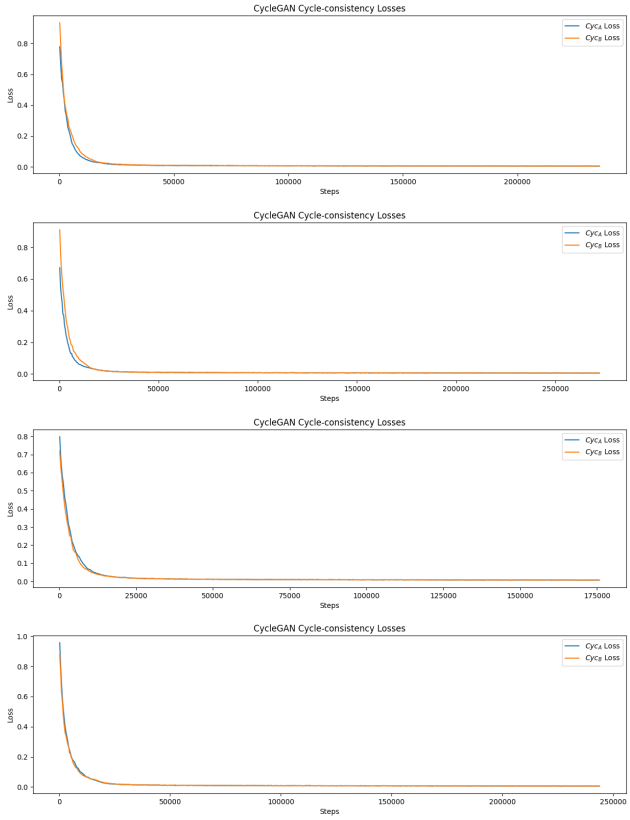


Figure 22: Cycle-consistency losses for the XY, XZ, YZ and ALL models. Same interpretation applies as described in figure 20.

5.4 Identity Losses

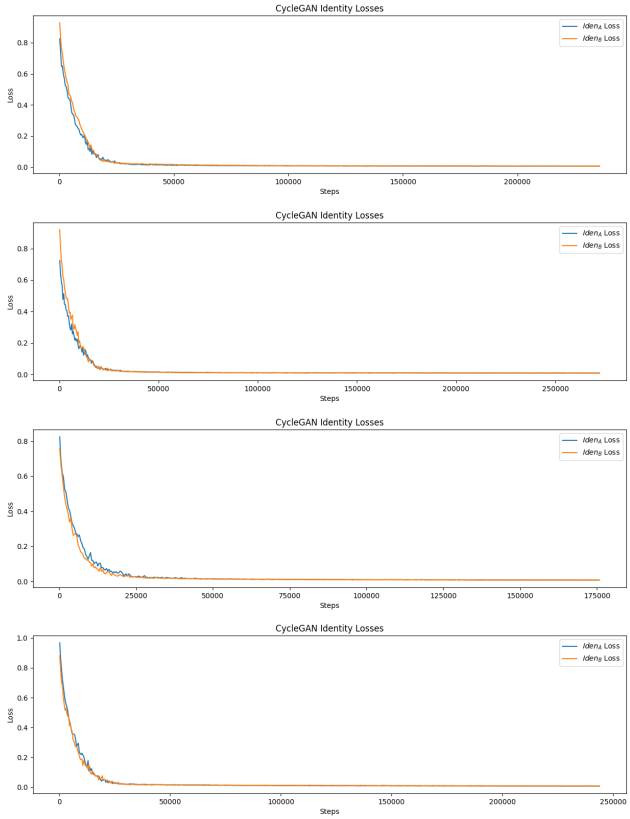


Figure 23: Identity losses for the XY, XZ, YZ and ALL models. Same interpretation applies as described in figure 20.

5.5 Training Progress

The following figure will outline what the training process of the CycleGAN models in the project look like.

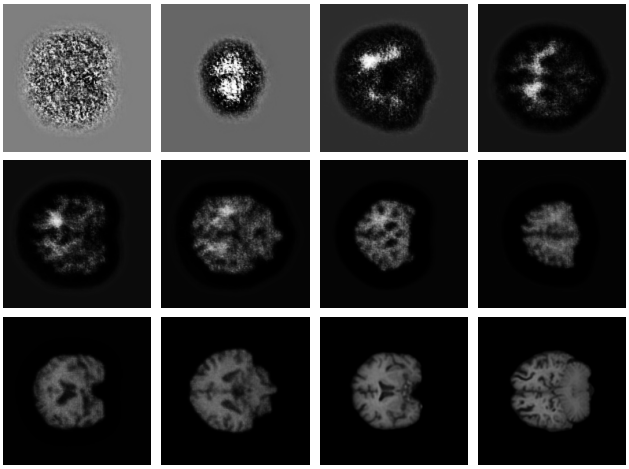


Figure 24: Examples of different slices from the XY model after 0, 1000, 5000, 9000, 11000, 13000, 17000, 19000, 22000, 27000 and 240000 steps respectively from left-to-right, top-to-bottom. The slices where the most interesting developments happen have been chosen. The bottom-left image is generated from the finished model. Note that the subject used to generate each image is random.

5.6 Example of 3D reconstruction from CycleGAN

The following shows what the reconstructed images look like in FreeView. The most import conclusion here is the fact that the reconstructed images are in the mni305 space.

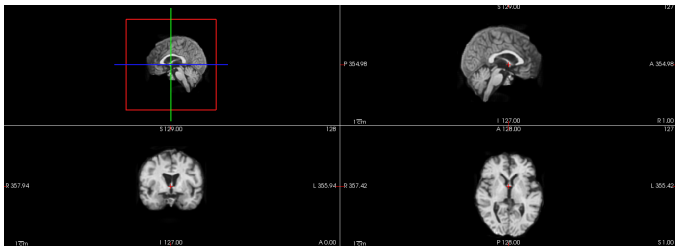


Figure 25: The figure shows the reconstructed 1.5T 3D image of the original 3T image from subject `002_S_0413`.

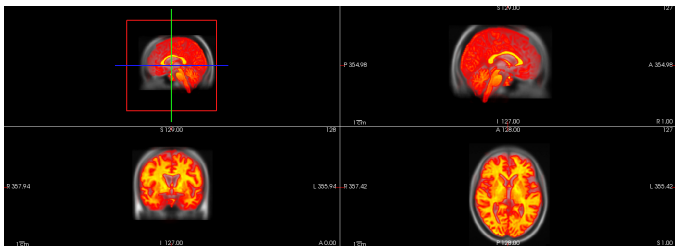


Figure 26: The figure shows the reconstructed 1.5T 3D image of the original 3T image from subject `002_S_0413` as a heatmap (The orange brain). Furthermore, this is put on top of the mni305 atlas showing that the reconstructed brain is in fact in the correct CRS.

5.7 XY Quality Comparison

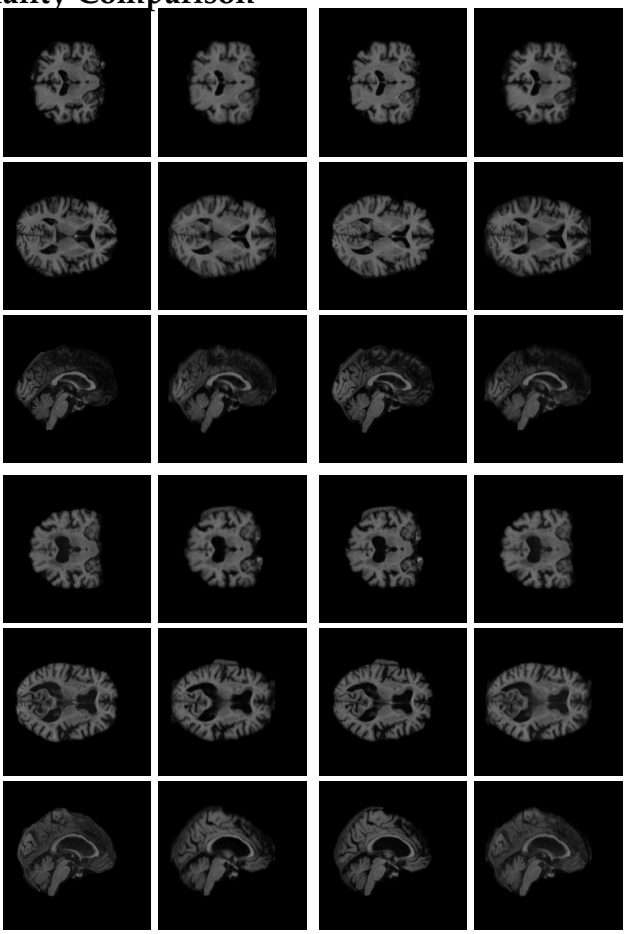


Figure 27: Generated images from the XY model are compared to real ground truth images. The columns, from left to right, contain respectively 1.5T ground truth images, 1.5T generated, 3T ground truth and 3T generated images. As such, the left part of the figure contains only 1.5T images and the right side only contains 3T images. **Top:** images for subject `136_S_0196`. **Bottom:** images for subject `082_S_0469`.

5.8 XZ Quality Comparison

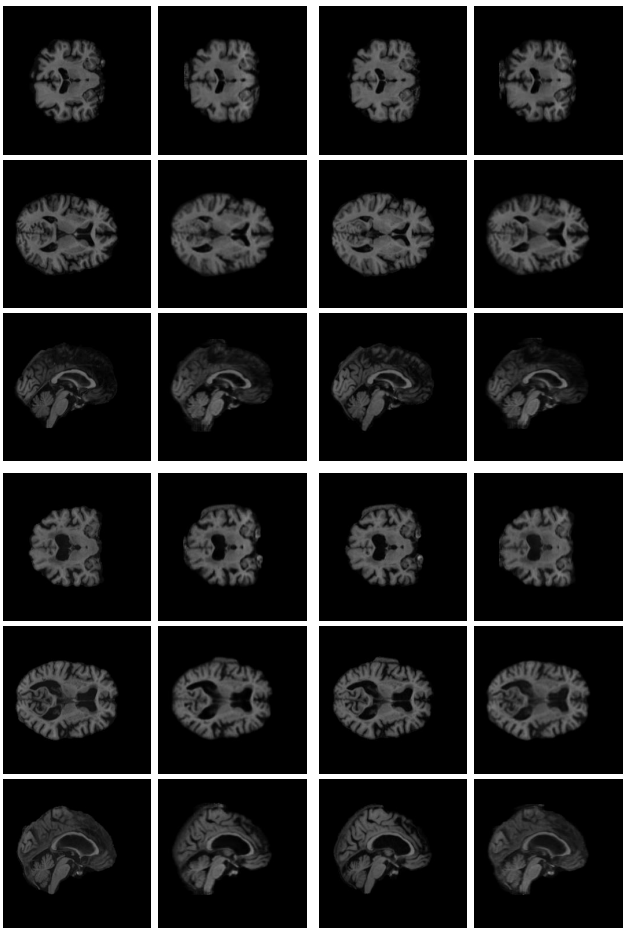


Figure 28: Generated images from the XZ model compared to real ground truth images. See the explanation in figure 27.

5.9 YZ Quality Comparison

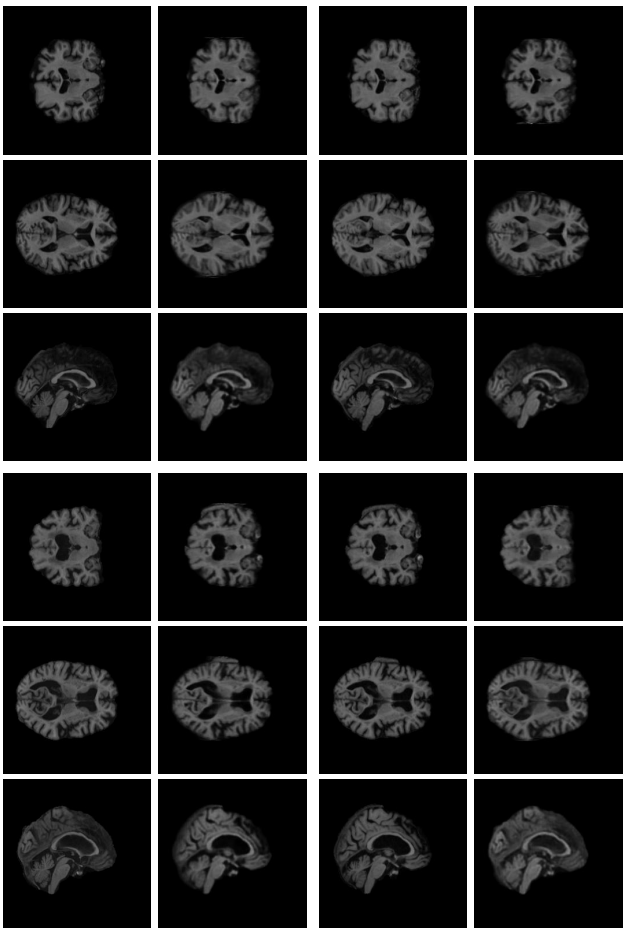


Figure 29: images from the YZ model compared to real ground truth images. See the explanation in figure 27 for a more detailed explanation.

5.10 All Planes Quality Comparison

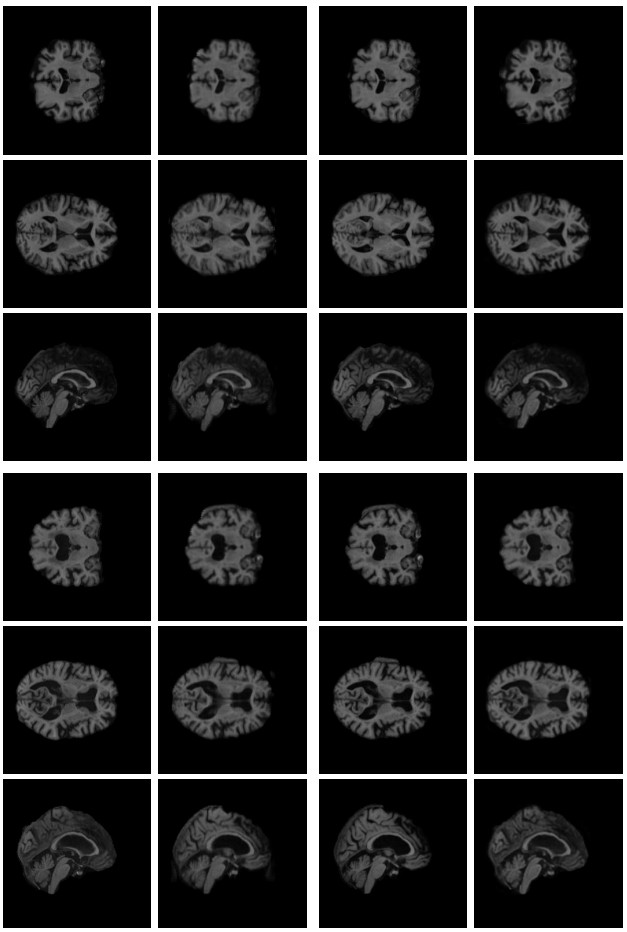


Figure 30: Comparing generated images from the YZ model to real ground truth images. See figure 27 for details.

5.11 XY Subtracted Images

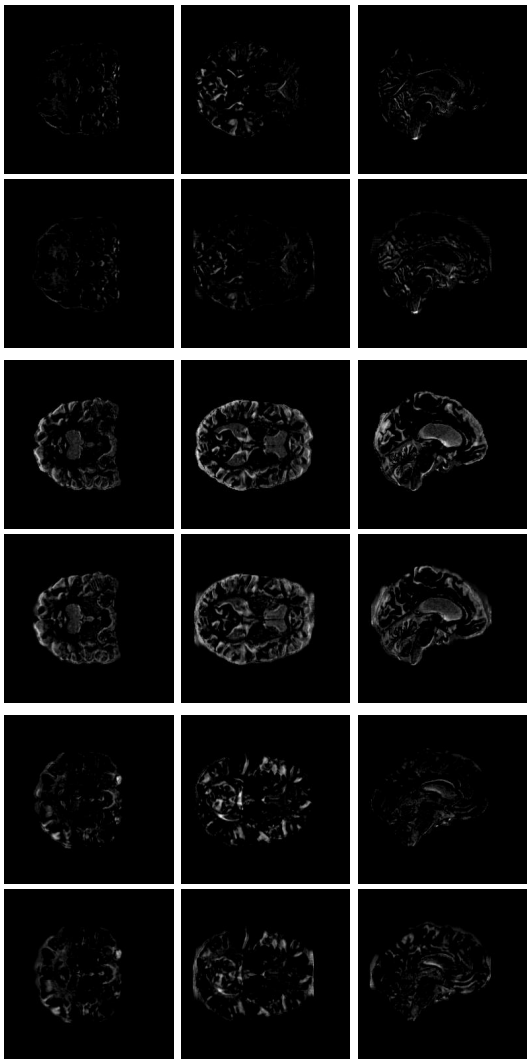


Figure 31: The top, middle and bottom blocks of six contain slices from respectively subject `002_S_0559`, `082_S_0469` and `136_S_0196`. The XY, XZ and YZ plane images are shown from left to right. The top row within each block of six shows the 3T image subtracted from the 1.5T image. The bottom row within each block of six shows the 1.5T perceived ground truth image subtracted from the XY model generated 1.5T images. As such, a darker image represents positive model performance. The images have been histogram stretched for each subject, so that their lowest-intensity pixel have value 0 and highest-intensity pixels value 255 to make the result more visible.

5.12 XZ Subtracted Images

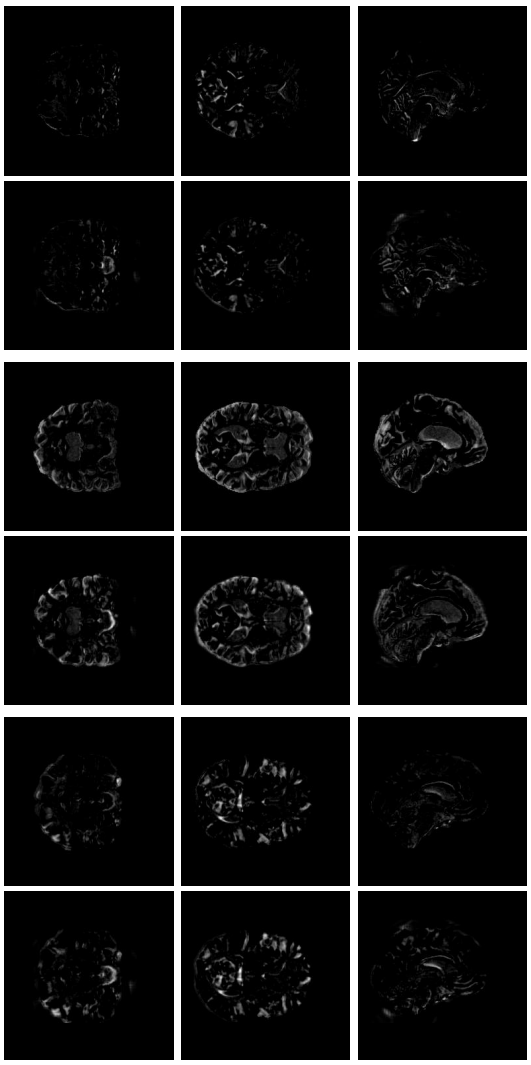


Figure 32: Same image subtraction images as figure 31, but with 1.5T ground truth and XZ model generated 1.5T images in the bottom rows of the blocks of six.

5.13 YZ Subtracted Images

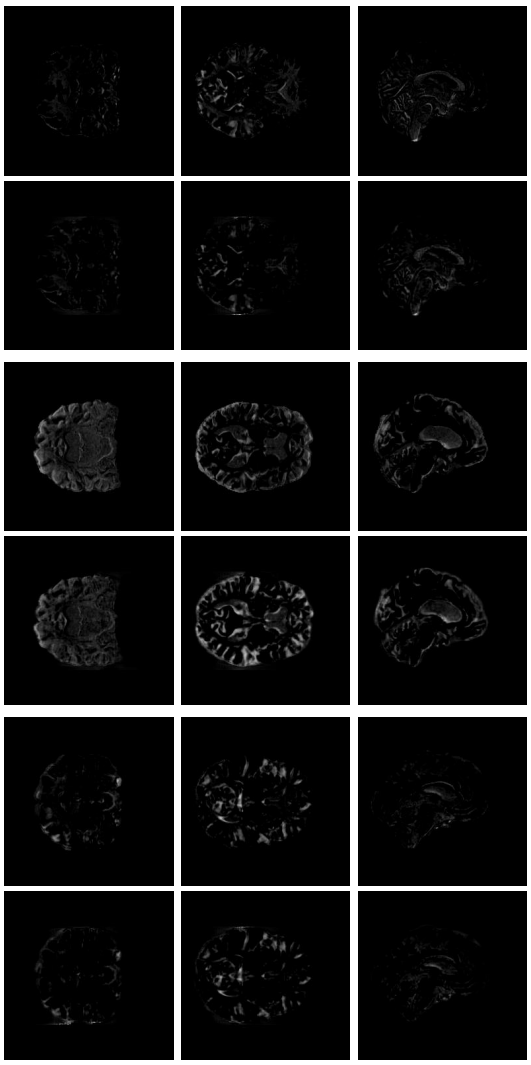


Figure 33: Same image subtraction images as figure 31, but with 1.5T ground truth and YZ model generated 1.5T images in the bottom rows of the blocks of six.

5.14 ALL Subtracted Images

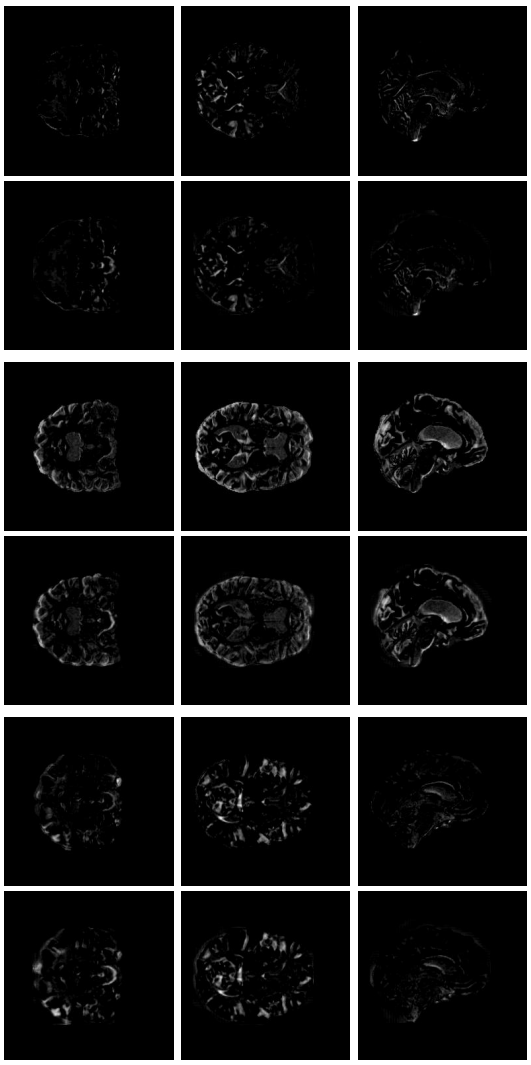


Figure 34: Same image subtraction images as figure 31, but with 1.5T ground truth and ALL model generated 1.5T images in the bottom rows of the blocks of six.

5.15 Evaluation Metrics

RMSE Scores

Model	Avg. Orig.	Avg. Gen.	Avg. Diff.
		3T → 1.5T	
XY	1.5344	1.0253	0.5090
XZ	1.5344	1.1691	0.3653
YZ	1.5344	1.0311	0.5033
ALL	**1.5344**	**0.8505**	**0.6839**
		1.5T → 3T	
XY	1.5344	1.0159	0.5185
XZ	1.5344	1.2517	0.2827
YZ	1.5344	0.9805	0.5539
ALL	**1.5344**	**0.9243**	**0.6100**

Table 9: The validation data contains 40 sets of images of subjects who have been scanned with both the 1.5T and the 3T MRI scanners. This makes it possible to determine a RMSE score for each of the four models. The table shows RMSE scores from the comparison of the images from the four models XY, XZ, YZ and ALL as well as the perceived ground truth image when generating from 3T to 1.5T and from 1.5T to 3T respectively. The avg. orig. column contains the average RMSE score between the original 1.5T and the original 3T images. The avg.gen. column shows the RMSE score between images generated by the models and the original target images. The avg.diff. column shows the average difference between the RMSE scores of the generated images and the baseline (ground truth). The rows highlighted with bold are the best performing models.

Model	Avg. Pixels	Avg. Norm [%]
	Geometric Accuracy 3T → 1.5T	
Baseline	**14745335.0**	**0.8788**
XY	14678174.659	0.8749
XZ	14603704.886	0.8704
YZ	14593826.818	0.8699
ALL	14599432.409	0.8702
	Geometric Accuracy 1.5T → 3T	
Baseline	14706659.0	0.8765
XY	14778896.955	0.8809
XZ	14776926.409	0.8808
YZ	**14793707.02**	**0.8818**
ALL	14694007.18	0.8758

Table 10: **Top:** This table shows geometric accuracy scores for comparisons between 1.5 ground truth images and images generated from each of the four models. **Bottom:** Geometric accuracy between 3T ground truth images and generated 3T images. The rows contain the four trained MRI CycleGAN models and the column contains respectively the number of pixels where the 1.5T ground truth and 1.5T generated images agree and the percentage of agreed upon pixels. The best model for each field strength is highlighted in bold. The baseline is calculated as the average percentage of black pixels in the relevant field strength image.

5.16 Classifier Loss & Accuracy - $\mathcal{M}_1$ & $\mathcal{M}_2$

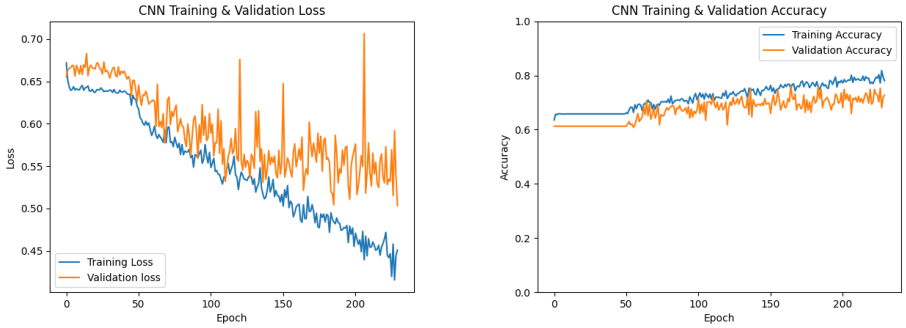


Figure 35: The figure shows training progress of a CNN classifier trained on synthetic CycleGAN images and original images. The classifier will be referred to as $\mathcal{M}_1$. The training & validation loss is on the left and training & validation accuracy on the right. The classifier tries to predict whether the subject has Alzheimer's disease or not. The figures can be used to determine convergence and signs of learning.

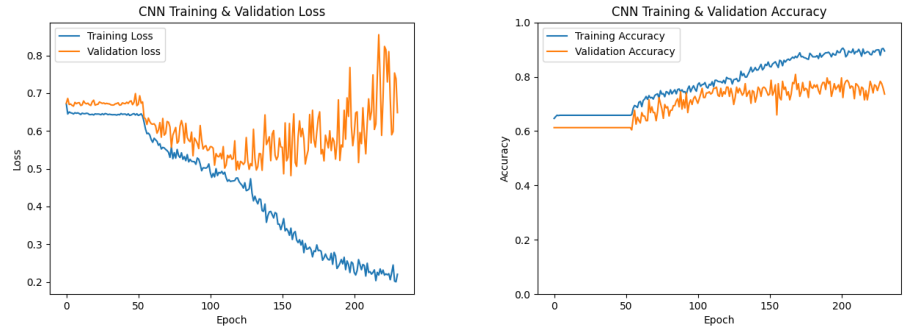


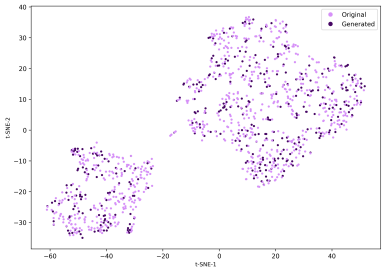
Figure 36: The figure shows training progress of a CNN classifier trained on original 1.5T and 3T images. The classifier will be referred to as $\mathcal{M}_2$. The training & validation loss is on the left and training & validation accuracy is on the right. The classifier tries to predict whether the subject has Alzheimer's disease or not. The figures can be used to determine convergence and signs of learning.

5.17 t-SNE & PCA

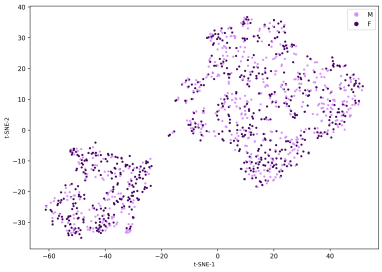
The plots in the following subsections are generated by forward passing images through the model and looking at the weight vector in the second

last linear layer of the model $\mathcal{M}$. The titles of the following subsections refers to the data used to generate the plots.

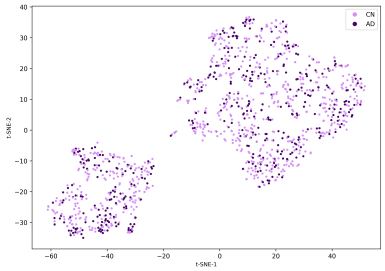
5.17.1 CycleGAN generated 1.5T images & original 1.5T - $\mathcal{M}_1$



(a) t-SNE plot colored by type of image. Generated refers to the subject image being reconstructed from 3T to 1.5T with the CycleGAN model.

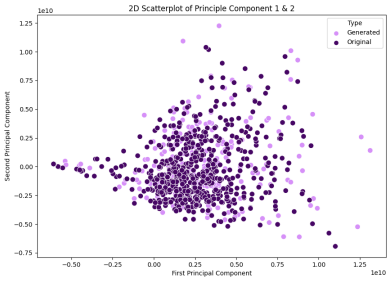


(b) t-SNE plot colored by subject sex.

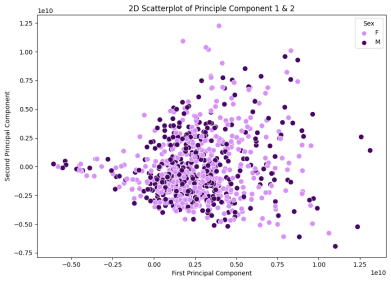


(c) t-SNE plot colored by group. Group refer to the label of the patient indicating whether the person belongs to Alzheimer's or control normal group.

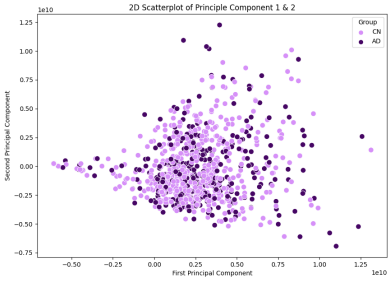
Figure 37: t-SNE plots of model $\mathcal{M}_1$. The plots resemble the features from the second last linear layer when forward passing data through the model. Each of the plots are colored by different labels. Figure (a) shows generated versus original images, (b) shows male versus female and (c) shows the Alzheimer's patients versus control group.



(a) PCA plot colored by type of image. Generated refers to the subject image being reconstructed from 3T to 1.5T with CycleGAN model.



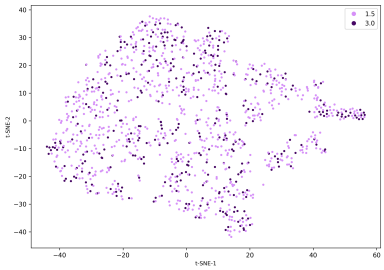
(b) PCA plot colored by subject sex.



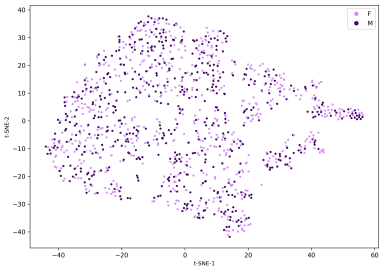
(c) PCA plot colored by group. Group refer to the label of the patient indicating whether the person belongs to Alzheimer's or control normal group.

Figure 38: PCA plots of model $\mathcal{M}_1$. Each of the plots are colored by different labels. Figure (a) shows generated versus original images, (b) shows male versus female and (c) shows the Alzheimer's patients versus control group.

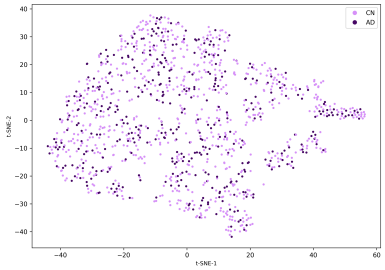
5.17.2 Original 3T & original 1.5T images - $\mathcal{M}_1$



(a) t-SNE plot colored by type of image. The label refers to the subject image being either 3T to 1.5T.

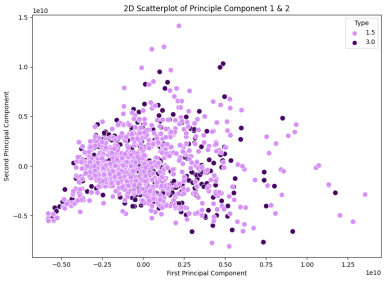


(b) t-SNE plot colored by subject sex.

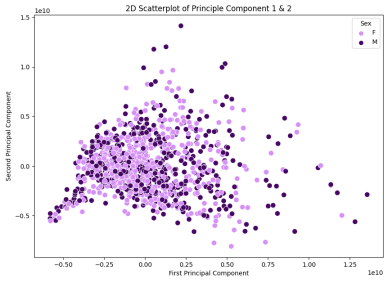


(c) t-SNE plot colored by group. Group refer to the label of the patient indicating whether the person belongs to Alzheimer's or control normal group.

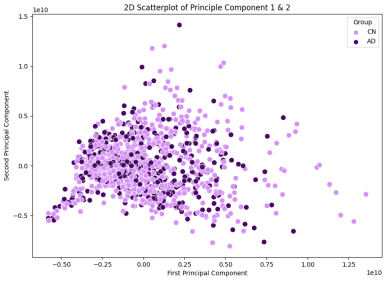
Figure 39: t-SNE plots of model $\mathcal{M}_1$. Each of the plots are colored by different labels. Figure (a) shows generated versus original images, (b) shows male versus female and (c) shows the Alzheimer's patients versus control group.



(a) PCA plot colored by type of image. Generated refers to the subject image being reconstructed from 3T to 1.5T with CycleGAN model.



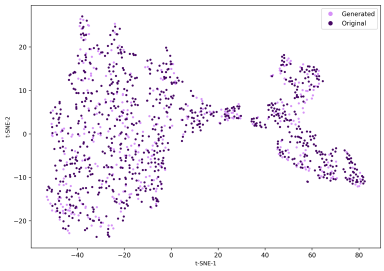
(b) PCA plot colored by subject sex.



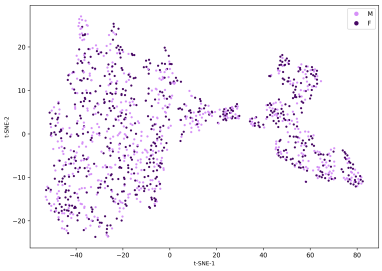
(c) PCA plot colored by group. Group refer to the label of the patient indicating whether the person belongs to Alzheimer's or control normal group.

Figure 40: PCA plots of model $\mathcal{M}_1$. Each of the plots are colored by different labels. Figure (a) shows generated versus original images, (b) shows male versus female and (c) shows the Alzheimer's patients versus control group.

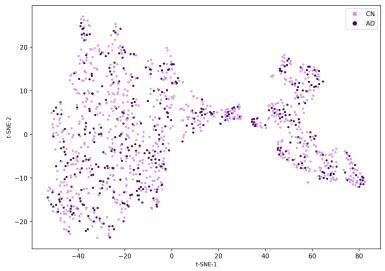
5.17.3 CycleGAN generated 1.5T images & original 1.5T - $\mathcal{M}_2$



(a) t-SNE plot colored by type of image. Generated refers to the subject image being reconstructed from 3T to 1.5T with CycleGAN model.

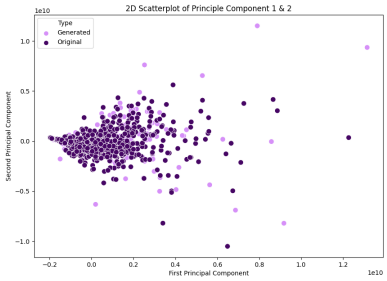


(b) t-SNE plot colored by subject sex.

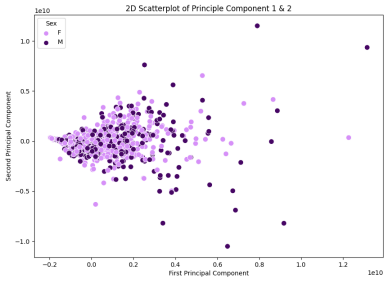


(c) t-SNE plot colored by group. Group refer to the label of the patient indicating whether the person belongs to Alzheimer's or control normal group.

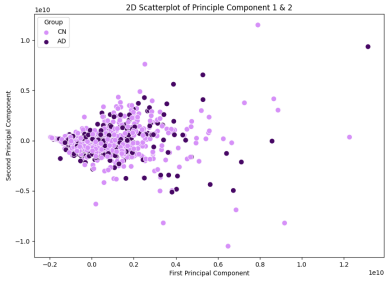
Figure 41: t-SNE plots of model $\mathcal{M}_2$. Each of the plots are colored by different labels. Figure (a) shows generated versus original images, (b) shows male versus female and (c) shows the Alzheimer's patients versus control group.



(a) PCA plot colored by type of image. Generated refers to the subject image being reconstructed from 3T to 1.5T with CycleGAN model.



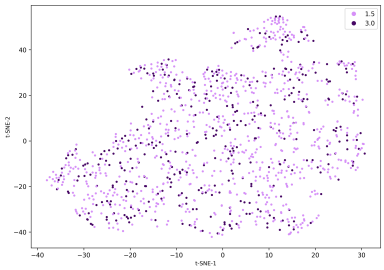
(b) PCA plot colored by subject sex.



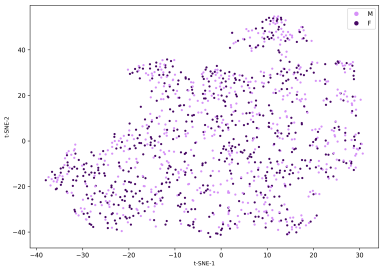
(c) PCA plot colored by group. Group refer to the label of the patient indicating whether the person belongs to Alzheimer's or control normal group.

Figure 42: PCA plots of model $\mathcal{M}_2$. Each of the plots are colored by different labels. Figure (a) shows generated versus original images, (b) shows male versus female and (c) shows the Alzheimer's patients versus control group.

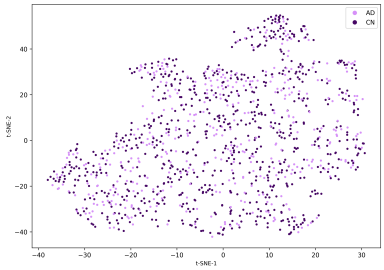
5.17.4 Original 3T & original 1.5T images - $\mathcal{M}_2$



(a) t-SNE plot colored by type of image. The label refers to the subject image being either 3T to 1.5T.

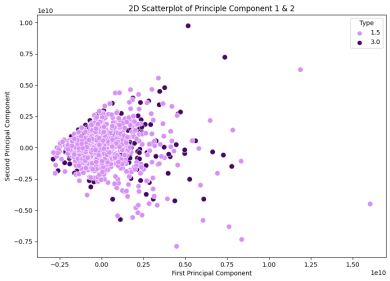


(b) t-SNE plot colored by subject sex.

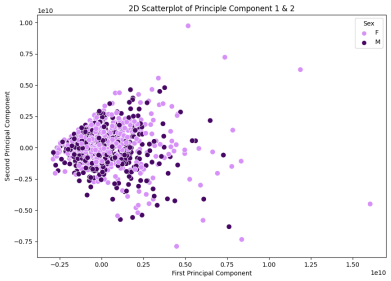


(c) t-SNE plot colored by group. Group refer to the label of the patient indicating whether the person belongs to Alzheimer's or control normal group.

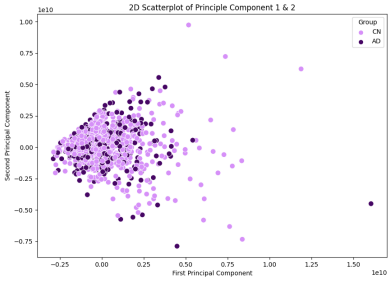
Figure 43: t-SNE plots of model $\mathcal{M}_2$. Each of the plots are colored by different labels. Figure (a) shows generated versus original images, (b) shows male versus female and (c) shows the Alzheimer's patients versus control group.



(a) PCA plot colored by type of image. Generated refers to the subject image being reconstructed from 3T to 1.5T with CycleGAN model.



(b) PCA plot colored by subject sex.



(c) PCA plot colored by group. Group refer to the label of the patient indicating whether the person belongs to Alzheimer's or control normal group.

Figure 44: PCA plots of model $\mathcal{M}_2$. Each of the plots are colored by different labels. Figure (a) shows generated versus original images, (b) shows male versus female and (c) shows the Alzheimer's patients versus control group.

6 Discussion

6.1 What the Results Show

This section aim to elaborate on the results and to discuss what the convenience of mapping between the 1.5T and 3T domain in practice implies, as well as the economic incentives, general principles of safe AI and regulations.

6.1.1 Difficulties and lessons from converging a CycleGAN

In section 4.2.6, issues in the CycleGAN outputs were remarked upon and, to a certain extent, fixed. This section showed that training these powerful models can be cumbersome and difficult. This section will focus more on what might be done to further improve the model and why training CycleGANs and GANs is so difficult. One of the biggest issues of trying the implement a successful CycleGAN is the sheer number of pitfalls. Mode collapse, checkerboard noise and especially lack of convergence have all been experienced in this project and these only scratch the surface of possible issues [41, 38]. Thus, this project proposes the importance of having fail-safes when training. Loss curves can give a false sense of security [53, 54]. Therefore, in addition to losses, examples of generated images from both models as well as the original image and potentially the cycle-generated image should be shown at regular intervals during training to ensure proper convergence.

Another issue entirely is the question of how to find the optimal hyperparameters. Throughout this project, a number of different parameters were ”brute-forced” until the best models were found, though this is not a viable option for everyone. Other possible options could include starting out with hyperparameters from a well-performing paper on a relating subject, using bayesian optimization[57] to find fitting hyperparameters or even changing the structure of the CycleGAN network with newer methods like the Wasserstein GAN[55] or the improved Wasserstein GAN[56], which promise to improve convergence and make the generator loss curves more relevant to generated image quality. In general, finding the best hyperparameters is difficult, requires practice and could become easier if it was easily possible to gauge the performance of the models both during and after training. The next section will focus more on this topic.

6.1.2 How to verify accuracy of the present CycleGAN

This section will focus on how well the MRI CycleGAN models have done in trying to replicate the complex structures of real brains. Looking at the generated images of 27, 28, 29 and 30, a couple of points become clear. All the generated images from all of the models definitely look like real brains. Furthermore, the images look convincingly like the ground truth image matching the same field strength, though some of the details from the old field strength of 3T seem to have seeped through into the new field strength of 1.5T. This is probably due to the fact that the CycleGAN models retain a large focus on structural similarity between input and output because of the cycle-consistency loss [54].

The subtracted images from figure 31, 32, 33 and 34 can be analyzed with a visual inspection. In these figures, it is worth noting that some of the images are definitely darker than the baseline images. This means that the brains have been made to look more like the targeted field strength than the baseline does. The rest of the subtracted images either look sort of the same or a bit worse. This definitely indicates that the models have at least learned to make brains look as convincing as the baseline brains. Comparing models is difficult, as is concluding on the true performance of models based only on images and subjective analysis. Other than these rather subjective notions, how else can the CycleGANs be evaluated?

A number of quantitative evaluations have been performed to get a better view of true performance. One of these, RMSE, can be viewed in table 9. Every model has achieved an RMSE better than the baseline of 1.5344. The XZ model performed the worst and the ALL model the best. This makes sense since the XZ model trained for the fewest epochs and was farthest from reaching a satisfactory convergence, which is shown in the second loss curve of figure 20 and 21. The ALL model might have been the best performing model due to learning from all three planes as input, thus making the model more robust. The fact that the xy, xz and yz planes are averaged might have improved performance as well.

When comparing geometric accuracies of the models to the baseline, table 10, the results seem less substantial. When converting images from 3T to 1.5T, the baseline model has a higher accuracy score than all models. Of the models, the geometric accuracy score is highest for the YZ model. The geometric accuracy scores suggest that the trained models have learned the geometric structure of a brain. Since the RMSE loss is low while the geometric accuracy is lower than baseline, it might be the case that the trained models get closer to the ground truth values without actually hitting

specifically the correct intensity value. This is discussed further in section 7. Reaching close to baseline shows that the models have learned to synthesize real brains to some extent.

6.1.3 Can synthetic CycleGAN Data be Used for Bias Removal?

One of the more prominent research questions is whether generated data from a CycleGAN will be able to remove the bias between the field strength of images when training a classifier on both synthetic and original data. From figure 35 it is evident that the classifier is learning, since both the training and validation loss are trending downwards. Moreover, the accuracy rises as the number of training epochs increase. Due to time constraints, the network is not fully converged. Interestingly, inspecting figure 37, the generated t-SNE plots do not seem to cluster any of the given labels differently. In other words, the internal feature representation of generated images compared to original images in the classifier model are similar, implying the the bias previously demonstrated has been removed. In addition, PCA cannot separate the two types of images in the second last layer of the classifier model, figure 38. Moreover, looking at figure 39 & 40, which illustrates a t-SNE plot by forward passing original 3T images and orignal 1.5T images, no similarity differences in the high dimensional feature space seems to be demonstrated.

Thus, the study suggests that it is beneficial to use CycleGANs for removing the field strength bias between 1.5T and 3T images in a deep learning, computer-vision model. Furthermore, it is suggested that using a CycleGAN to generate synthetic data to extend large datasets or perform advanced augmentation seems very promising.

Efforts in this project was put into trying to recreate the bias found and presented by Kergel[9]. A model identical to the CNN classifier trained on generated data was trained purely on the original data[10]. From figure 36 it is apparent that the model trained on the data is prone to overfitting. Thus, one would suspect a similar bias towards the magnetic field strength of the images. However, from figure 43 & 44, the bias towards the field strength is not present.

The reason why the model does not inherit the field strength bias can be ascribed to differences in data used by Kergel and data used in this project. Kergel has used the same raw images, but the data was preprocessed using

[9] See, section 1 and consult reference: [20]

[10] The preprocessed versions of the 1.5T & 3T images. The files named `norm_mni305.mgz`

spm12 and DARTEL[11]. In this project, the preprocessing has been quite different using FreeSurfer Software. It might be the case that a different preprocessing has removed the bias in the trained classifier. However, further investigation needs to be done to establish this.

Another question that might come to light is if there was a field strength bias between the types of images to start with. As suggested by the results in this project, no bias can be found. Looking at the Baysia et al. model implemented by Kergel, the bias furthermore seems to be less apparent. For completeness, this figure has been included below, see figure 45.

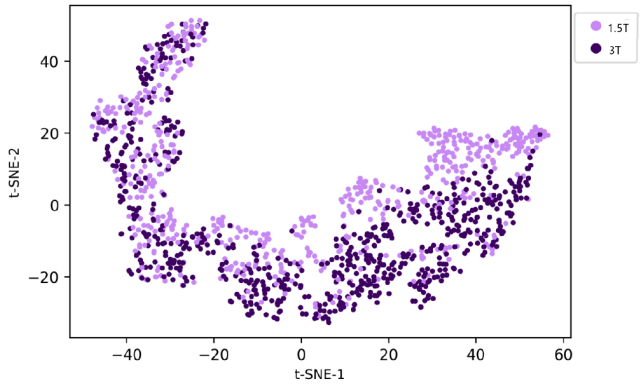


Figure 45: The t-SNE plot from second last linear layer from the Baysia model from thesis by Kergel. Figure courtesy of [20].

Despite the bias not being demonstrated in any of the classifiers trained in this project, another interesting insight into CycleGAN generated data can be gained. No bias towards the generated images can be seen when inspecting figures 41 & 42, which shows the t-SNE and PCA plot for model $\mathcal{M}_2$, the classifier trained on original images. Considering that this model has never seen any type nor resemblance of CycleGAN generated images, the t-SNE and PCA plots imply that the generated data lies very close to the original MRI data. By visual inspection it almost looks as if they come from the same distribution. Thus, the analysis confirms the RMSE-score results from section 5.15, and implies that the generated data can be useful for learning and can potentially increase the robustness of machine learning

[11]DARTEL is another toolbox and is based on "A Fast Diffeomorphic Registration Algorithm" paper (Ashburner, 2007) [73].

algorithms trained on MRI data.

6.2 Convenience of mapping: $G : X \rightarrow Y$ & $F : Y \rightarrow X$

The reason why mapping from one domain to another for deep learning on MRI data is due to the fact that, in a lot of cases around the world, accessing MRI data can be complex. For some studies only one type of field strength measurement is available. If a model can map between two domains accurately enough all MRI data that exist, regardless of the field strength, can be used to train a CNN classifier model. This, implies that a model can generate 3T images from 1.5T and vice versa.

Advantages of the domain translation will be to train a deep learning model with data where the demographics are widely different. It would enable the training of models with subjects from all around the world in contrast to limiting datasets where only one type of field strength image is used. Potentially, this can remove demographic bias if such a bias exist in the model while making the trained classifier more generalizanble and useful for societal implementation. Moreover, the price of 3T MRI scanners are double of what you pay for a 1.5T scanner. Not only are the machinery more expensive, but maintenance and costs of having 3T scanners is also more expensive [72]. If a perfect mapping exists another implication of this study is that 3T scanners are no longer required.

6.3 Economic Incentive of a Good Classification Model

One could ask the question of what societal and economic implications a good classification model within this field of study could imply. This section aims to estimate the outcome of implementing such a classifier into the Danish society. This will give an insight into whether continuing to study in this field would make sense for the sake of societal impact. The danish 2025 dementia plan has the following goals:

- Denmark needs 98 dementia friendly communes
- 80 % of people with dementia need a specific diagnosis
- Reduce the amount of anti-psychotic medicine with 50 % in the year 2025

One of the relevant key focus areas include early detection of dementia which can be aided by deep learning models such as a classifier described

in section 1.2, also consult [29, 26, 27, 28, 25]. An average patient with dementia costs approximately 51000 `DKK / year`:

Practice sector	3900 DKK
Hospital sector somatics	33500 DKK
Hospital psychiatry	4000 DKK
Medicine	9700 DKK

In Denmark around 85000 – 90000 people live with a dementia disease. Around 50000 of these cases are Alzheimer's. Every year, around 8000 new cases are detected, and in 2040 the expected amount of people with dementia above 60 years old is 125000 to 150000. This amounts to Denmark spending 4.59 to 7.65 Billion DKK / year on Alzheimer's Patients [12]. These numbers do not take healthcare personnel, researchers and other variable costs into consideration. Moreover, 65% to 85% of all inhabitants in eldercare has a dementia disease. The cost of informal care time ranges between 5.0 to 6.9 hours daily. Furthermore, the daily cost was estimated to range between 1190 – 1658 DKK / day for primary caregivers[13] and 171 – 253 DKK / day for secondary caregivers. The cost for Denmark amounts to 18 Billion DKK / year on eldercare alone. The total cost of dementia related diseases in Denmark is estimated to be 20 Billion DKK / year [30] [31]. Thus, a screening to detect Alzheimer's before a patient is showing symptoms can be very beneficial to the danish economy. It would furthermore optimize life expectancy and quality of life for citizens in Denmark.

6.4 Safe AI

The principles of safe AI build on a set of rules that the constructed models will have to obey in order to be safe to use on a societal scale. When dealing with models such as the classifier or CycleGANs presented in this project the ethical concerns for potential misuse or hidden biases are very important due to the effects an implementation can have on society. Trust between the prediction of a model and the predicted individual is, thus, key for success. The very first steps of achieving this trust is both by delivering a good service from the machine learning industry and bringing the topic to life in a public discussion in society. The Ministry of Foreign Affairs of Denmark has contributed to just that in collaboration with the Technical University of Denmark (DTU). The outcome of safe AI is machine learning models with

[12] !NB Not taking inflation into consideration for the outlined numbers
[13] A primary caregiver can range from everything from a nurse, family member etc.

democratic values and control mechanisms. Trust from the user point of view is very important for the whole research of AI. If individuals in society do not trust the developers behind them, their data will not be available to further research and enhance the models. This will result in big companies, such as Amazon, Facebook, Google etc., not being able to further develop the technology [65]. The safe AI principles are presented below:

Safe AI [14]	**Explanation**
is secure	Test of verified software and hardware, adversarials
is open-source	Methods, code, hardware, check and evolution
is self-conscious	Understands own role
is secretive	Privacy by design
has calibrated values	Debug for stereotypes, biases
is accountable	Transparent, communicating, "right to explanation"
understands social relations	Understands user's knowledge graph
understands power	Digital self-defense
generates trust	Use in public relations

As can be seen from the safe AI principles, most of them do not apply to the current technology. Is self-conscious, is accountable, understands social relations, understands power and generates trust are all principles for the perhaps distant future. However, principles such as secure, open-source and has calibrated values are more relevant, both for the models presented in this project, but also for the present time and the current state om machine learning. Calibrated values include debugging for biases and stereotypes. A bias seen from previous studies was the varying effect of 3T images and 1.5T images, but biases could also be more harmful. This include gender bias, age bias, racial bias and a plethora of potentially dangerous biases. If a model such as the one presented in this project will classify Alzheimer's patients before they develop symptoms, then the patients need to trust the prediction, and the developers need to ensure fairness and calibrated values.

6.5 Regulations

Due to the rapid advancements of AI in recent years, laws and regulations have not been implemented to avoid potential abuse. A technology which

[14] Table was also presented in previous work, see [32]

is often referred to as a black box tool, where even the developers of AI are not able to explain, why certain predictions are made. With models being trained on historical data and undoubtedly inheriting bias, this study calls for the need for regulations of models with the intend of being implemented into society. To illustrate why there is a need for AI regulations a hypothetical example will be presented.

Imagine a classifier which can predict lung cancer in very early stages from screening of patients. This is already achieved by Raponi et al. [66] with a prediction accuracy of 78%. If the classifier could achieve a prediction accuracy of 90% it would have the possibility to be an advanced tool for doctors or even replacing doctors in the screening. However, despite the 90% prediction, developers with a malicious intent could have programmed the model to always classify healthy lungs for certain ages, minorities or genders, thus discriminating certain groups. Furthermore, this would leave individuals with severe damage or even death in worst cases. In practice, this would have little to no consequences, since there are no preventative regulations or laws to avoid it. This does not only apply to the field of medicine, but to recidivism prediction [32] and potentially all other machine learning models being implemented for use on large groups of people.

By default, this is not an easy problem or area of technology to create laws and regulations for, since many demonstrated models are being used presently, e.g. facial recognition on our phone, email filtering algorithms, Facebook adds, Google adds and much more. In conclusion, a look at both the current models in place and the potential future machine learning models is necessary. A stepping stone could be to create a third party organ, which works as a confidential code reviewer handing out certificates to AI models, which comply with the decided regulations and laws for the country it is implemented in. The third party organ would, moreover, have to track back through the already implemented models to ensure the regulations.

The regulations should be build upon the safe AI principles, but should also include notions of fairness to ensure no groups are being discriminated. Therefore, a council consisting of experts within the domains of machine learning propose , and the industries in which machine learning is used, as well as civil individuals should decide on key parameters, which can be used to build the regulations themselves to inspire trust, transparency and safety for the future.

7 Future work

7.1 Method to measure performance of a CycleGAN

An important thing to ensure in future work within this field is a standardized way to measure the performance of a CycleGAN and its mapping between two domains. For this project, a 1.5T and 3T scan of the subject was available. Despite this, when registering the subjects to the MNI305 atlas, the two original images will not align completely due to the images stemming from two different measurements, which means that a patient was in two different MRI scanners. Hence, the patient was not in the exact same position resulting in minor changes as to where the brain is in the subject space. Thus, when registering the image to MNI305, the images are not completely aligned. If the two original images are compared to each other (the 1.5T and 3T image), a given pixel from one image is not in the same position in the other image. Moreover, when a synthetic 1.5T version generated from a 3T images is compared to the original 1.5T image, one cannot simply subtract the images.

Proposed in this study is the RMSE between the generated 1.5T and original 1.5T and a geometric accuracy, which does a pixel-wise comparison. The geometric accuracy will give an overview of whether the model has learned the geometric structure of the brain, where the RMSE will tell us how far the generated images is from the original image. However, even these measures might be biased by the two different scans of the subject. In conclusion, this report calls for new methods to quantitatively evaluate if the synthetically generated images from the CycleGAN model are similar to the 3T equivalents that do not exist.

7.2 Training multiple classifiers

One of the main things for future work is to train more classifiers than the simple one implemented in this study and getting the large networks to converge before perform the t-SNE analysis. From figure 35, it is very apparent that the network is not fully converged, which may cause spurious conclusions as to whether the bias is completely removed or not. The CycleGAN model used to generate data for the showcased CNN classifier was trained on the xy planes. Therefore, future work needs to investigate some of the other CycleGAN models, epecially the `ALL_model`, which has the lowest RMSE of the ones trained in this project.. To fully investigate whether the bias between field strength images is successfully removed

from the model, an identical classifier trained purely on original data also needs to be trained and converged for comparison.

7.3 Investigating possible accuracy improvements using synthetic data

Exploring if synthetic data can be used to make models more robust and generalizable, has been a fruitful idea since the beginning of GANs. However, the data being generated is often noisy and does not seem to resemble the training dataset distribution accurately enough to be useful for generation of training data. Despite this, using synthetic data generation from a CycleGAN model has not yet been explored or used as an advanced augmentation in any state of the art medical imaging models. Future work should include the possibility of using synthetic data from a CycleGAN as part of the training data, to ensure slower convergence of classifier networks such as the Baysia et al. or the one presented in this project, which in turn result in more robust and generalizable models. This include classifiers in the X-ray domain, MRI, fMRI etc.

To test this hypothesis, experiments should be carried out to train two identical classifiers. One classifier on a dataset with synthetic data in it and another on a dataset without synthetic data.

8 Conclusion

The proposed CycleGAN models from this project can, according to the accuracy measurements presented, map successfully between the 1.5T and 3T MRI image domains. The best performing model is the `ALL`-model, which has a RMSE score of 0.8505 when mapping from 3T to 1.5T and 0.9243 when mapping from 1.5T to 3T compared to the perceived ground truth image. However, in order to confidently conclude that a CycleGAN can construct 3D MRI images, a proper and agreed upon methodology of how to measure the precision of the generated data is needed.

The CycleGANs' generated images seems promising as training data for a CNN classifier for Alzheimer's disease. It is evident that the classifier models are indeed learning, since the overall trend of the loss is downward. The models are not fully converged due to the time it takes to train a model with ~560 million parameters. Furthermore, the bias between 3T and 1.5T images has been removed to some extent.

If synthetic generated data from a CycleGAN proves successful in extending current datasets and generating even more data, it could potentially result in more robust models with better accuracy performances. This could potentially improve the economy of healthcare significantly, which enables governments around the world to allocate money elsewhere. Thus, investing in Alzheimer's research not only contributes to ensuring unbiased prediction models, which confronts the limits & possibilities within deep learning, but additionally contributes to drive down costs in society and increases the quality of life for individuals within.

9 Appendix

This section shows the parts of the report that were not important enough to be included in the general architecture of the project but still played an important role in explaining parts of the project. Here, the shell scripts used for preprocessing will be shown and the preliminary GAN and CycleGAN results have been showcased.

9.1 Shell scripts

Listing 1: FreeSurfer Preprocessing

```
#!/bin/bash
# Script to process FreeSurfer for Bachelor Project:
# AlzPred by Anders Henriksen & Oskar Wiese
source FSstable71

cd /dtu-compute/ADNIbias/freesurfer/

filename='freesurfer_log.txt'

rm -f $filename

touch $filename

count=1

SUBJECT_NAMES=$(ls /dtu-compute/ADNIbias \
/ADNI1_baseline3T_collection/ | grep '_S_')

echo 'Starting FreeSurfer by Mr FreeSurfer'

for f in $SUBJECT_NAMES; do
echo 'Workin on:' >> $filename
echo $f >> $filename
echo $count >> $filename
count=`expr $count + 1`
recon-all -motioncor -s $f -i /dtu-compute/ADNIbias/ \
ADNI1_baseline3T_collection/$f/MPR*2/20*/S*/*.nii
echo 'Motion Cor done' >> $filename
recon-all -nuintensitycor -s $f
echo 'Nu intensity cor done' >> $filename
recon-all -talairach -s $f
echo 'Tailarach done' >> $filename
recon-all -normalization -s $f
echo 'Normalization done' >> $filename
recon-all -skullstrip -s $f
echo 'Skullstrip done' >> $filename
recon-all -gcareg -s $f
echo 'gcareg done' >> $filename
recon-all -canorm -s $f
echo 'canorm done' >> $filename
echo '' >> $filename
done
```

Figure 46

Listing 2: FreeSurfer mri_vol2vol

```
#!/bin/bash
# A script to create a normalized MNI305
cd /dtu-compute/ADNIbias/freesurfer_ADNI1/

source FSstable71

SUBJECT_NAMES=$(ls /dtu-compute/ADNIbias/ \
freesurfer_ADNI1/ | grep '_S_')

for f in $SUBJECT_NAMES; do
echo "Processing subject: $f"
mri_vol2vol --mov /dtu-compute/ADNIbias/freesurfer_ADNI1 \
/$f/mri/norm.mgz --targ $FREESURFER_HOME/average \
/mni305.cor.mgz --xfm /dtu-compute/ADNIbias \
/freesurfer_ADNI1/$f/mri/transforms/talairach.xfm --o \
/dtu-compute/ADNIbias/freesurfer_ADNI1/$f/norm_mni305.mgz
done
```

Figure 47

9.2 MNIST GAN

9.2.1 Losses

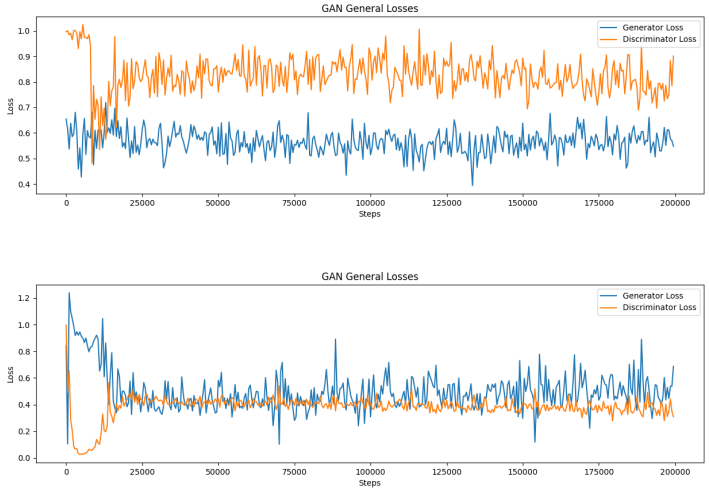
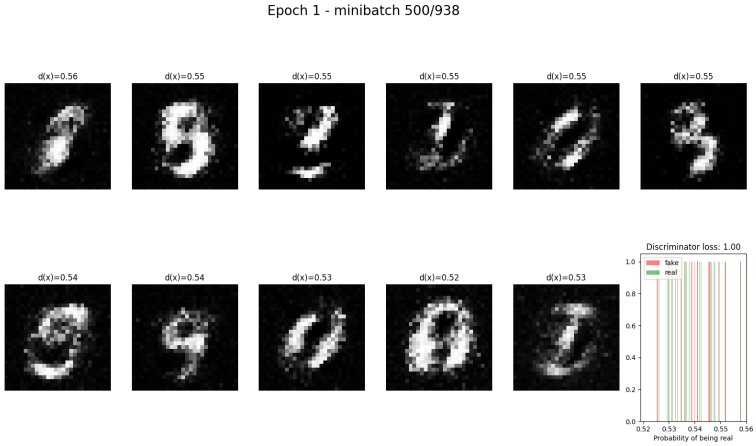
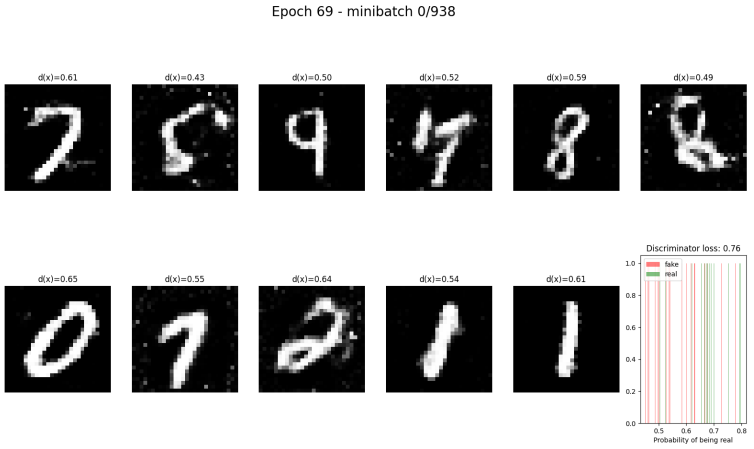


Figure 48: The losses measured two times each epoch for 200 epochs during training of the GAN on the MNIST dataset. **Top:** Training the model using L1 loss for the discriminator and BCE with logits loss for the generator. **Bottom:** Training the model using MSE loss for both models. MSE has a tendency to generally work well when training neural networks.

9.2.2 Progress of Training





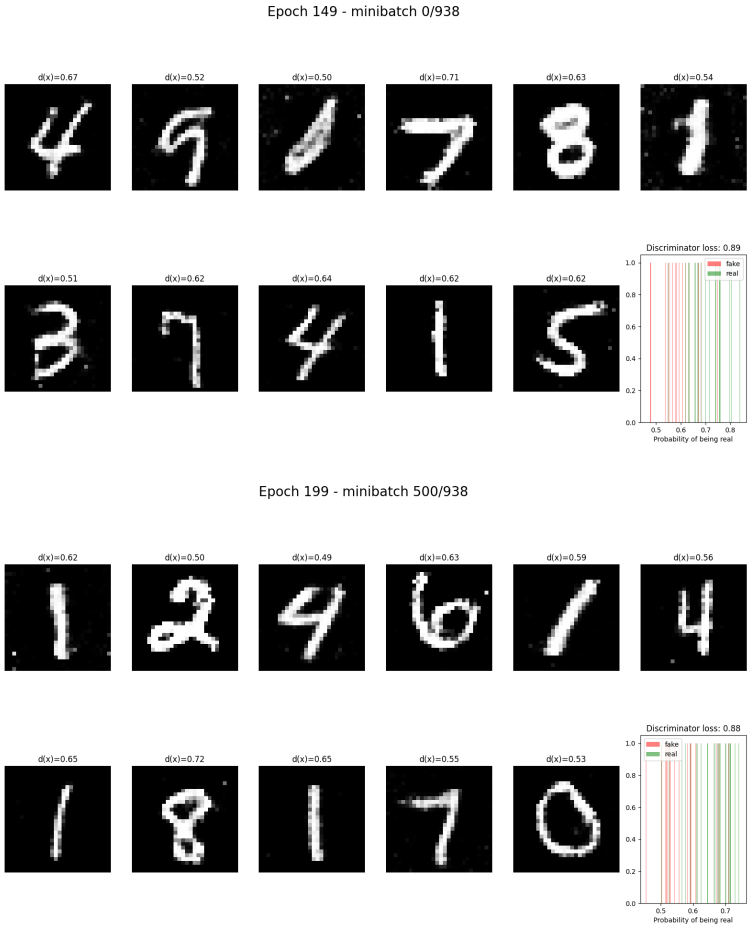
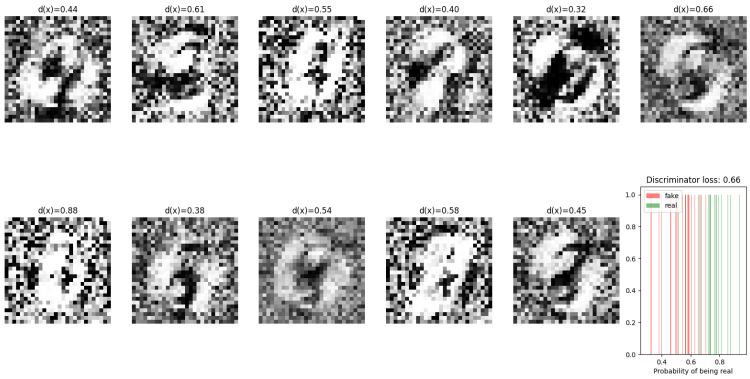
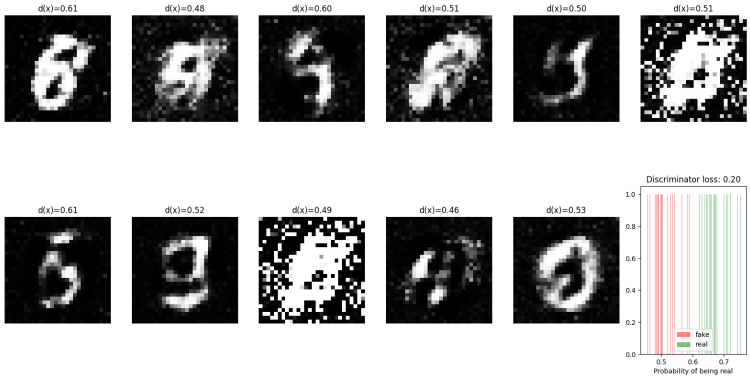


Figure 49: During training, 11 random generated images have been shown at different points along the training process (epoch 2, epoch 70, epoch 150 and epoch 200). The epochs were chosen to show the most interesting behavior or the achieved best performance of the model. The last of the 12 plots depicts the discriminator prediction for the current minibatch. The red columns are generated images and green columns are real images. as such, a perfect discriminator would place every red column at 0 and every green column at 1.

Epoch 2 - minibatch 0/938



Epoch 14 - minibatch 0/938



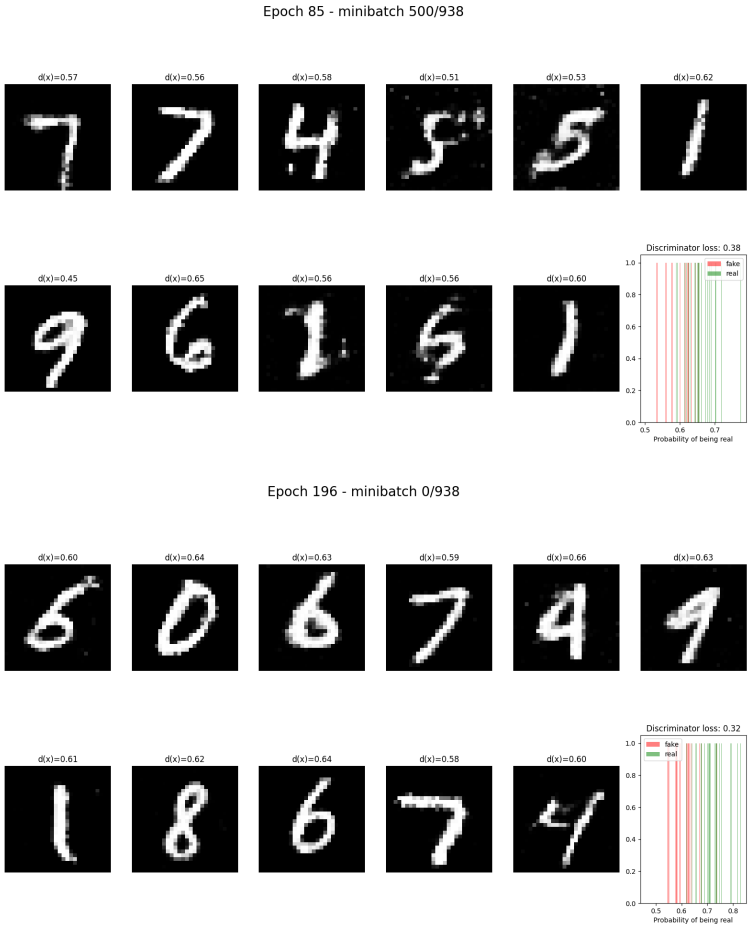
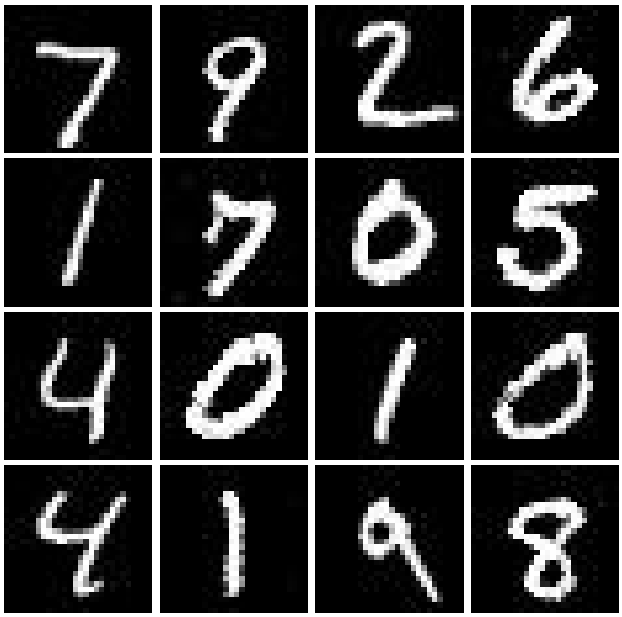
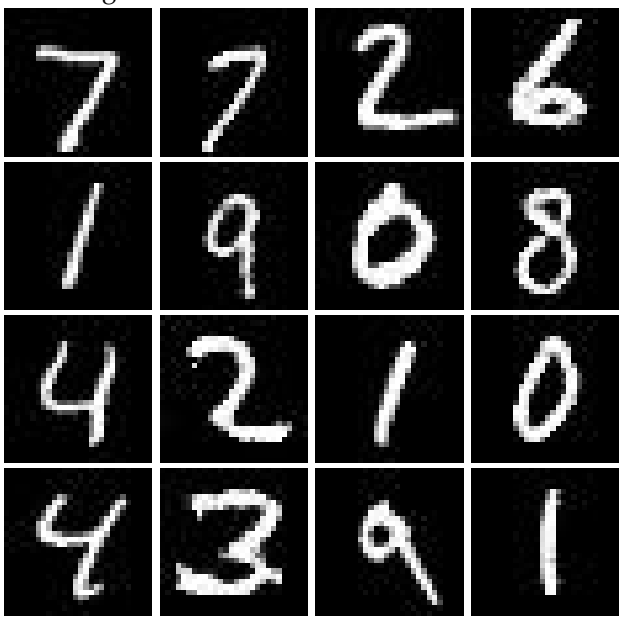


Figure 50: During training, 11 random generated images have been shown at different points along the training process (epoch 3, epoch 15, epoch 86 and epoch 197). The epochs were chosen to show the most interesting behavior or the achieved best performance of the model. The last of the 12 plots depicts the discriminator prediction for the current minibatch. The red columns are generated images and green columns are real images. as such, a perfect discriminator would place every red column at 0 and every green column at 1.

9.2.3 Generated Images



(a) Samples of real images from the MNIST dataset of handwritten digits and generated images from the generator in the MNIST GAN using L1 loss and BCE. Column 1 and 3 show real images and column 2 and 4 show generated images.



(b) Samples of real images from the MNIST dataset of handwritten digits and generated images from the generator in the MNIST GAN using MSE loss. Column 1 and 3 show real images and column 2 and 4 show generated images.

9.2.4 Discriminator Predictions

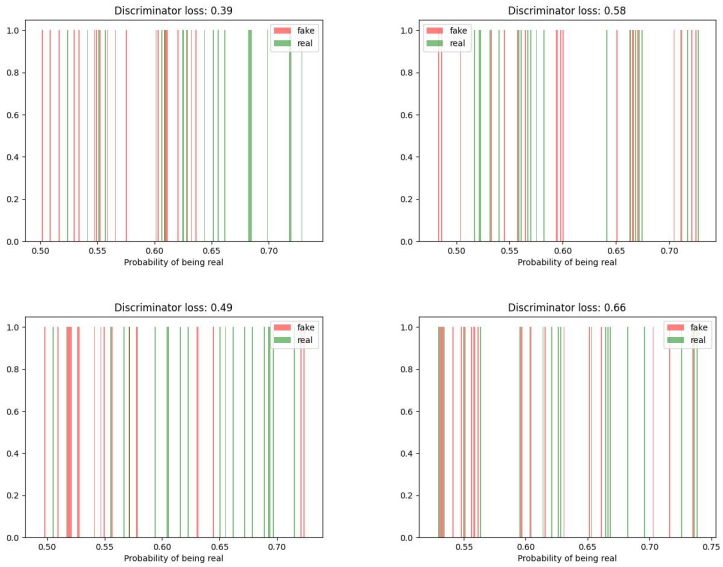


Figure 52: The discriminator predictions on 20 real images from the MNIST data and 20 fake images generated by the L1 loss and BCE loss generator. The four discriminator predictions are chosen at random.

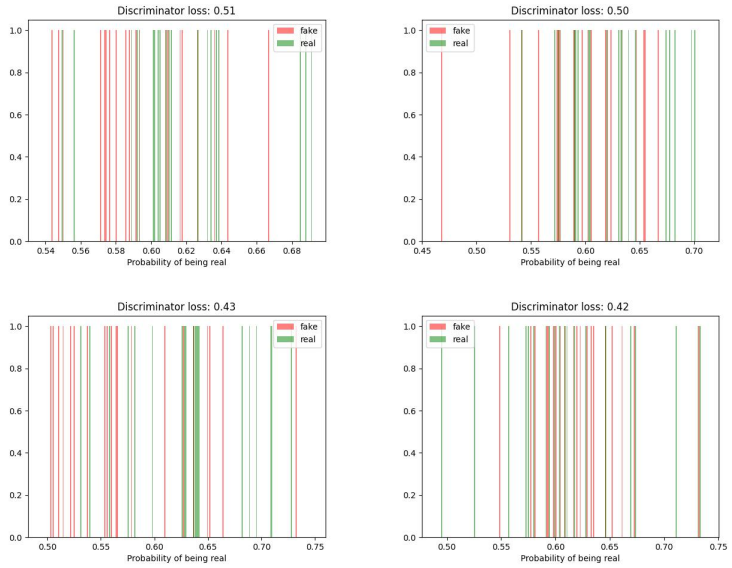


Figure 53: The discriminator predictions on 20 real images from the MNIST data and 20 fake images generated by the MSE loss generator. The four discriminator predictions are chosen at random.

9.2.5 Discussion of Results

This sections aims to interpret the results of training a GAN to generate images from the MNIST dataset shown in section 9.2. In the loss function plot for BCE and L1 loss GANs in figure 48a, the losses do not seem to change much over time, except for the large downwards spike of the discriminator in the beginning of training. This would normally be a negative sign, signifying that the models are not learning much of anything. In the case of GANs though, the discriminator and generator are always fighting betwixt each other for the lowest possible loss. This means that a theoretical increase in one performance leads to a decrease in the performance of the other. This means that both models are actually still learning although their losses are not an indication of such. Much the same can be said for figure 48b, though this figure seems to generally be more interesting. As expressed ad nauseam in the literature, a successful GAN should have losses that slowly converge towards some constant over time. This seems especially true here, since the discriminator gets higher loss while the generator gets lower loss until they arrive at pretty much the same loss. The generator generally looks to have slightly higher loss and a much larger variance of loss for each step.

Some hyperparameters could thus be tuned further to allow the generator to reach a more stable point of convergence.

In the next section, it is discovered how the generator learns using the two different loss functions for training. In the first row of figure 49 is shown how the Binary Cross Entropy (BCE) loss very quickly enforces background pixels to be black and the generators starts learning the general outline that is shared by multiple digits after only 1.5 epochs. Many images look to be a mix of 5 and 9 and 1 and 2 especially. In row two after 70 epochs, many of the digits have become sharp and now look only like one digit. Some small artifacts like floating pixels and disconnected parts of digits are still present, especially in the 8's, 5's and 7's. After epoch 150 in row three, the generator has now learned to connect most parts of the digits, although it still generated random small pixels at incorrect parts of the image. Finally, in row four after 200 epochs, most of the grains now seem to have been removed and only some artifacts are left in the images. This shows that even though the generator loss in figure 48a was stable, the generator wass definitely still learning. Using Mean Squared Error (MSE) loss instead of BCE, it seems that the generators learns in a completely different manner based off of figure 50. Here, row one after training 3 epochs shows that the general shape of a digit has been found, though the salt-and-pepper noise from the original gaussian-distributed noise is still present in all the images. After 15 epochs, in row two, most of the images now get generated with little to no salt-and-pepper noise, and the shape of some digits is already starting to become visible. After 86 epochs in row three, the generator seems to be in pretty much the same spot as the BCE generator was at epoch 150. The shape of singular digits is learned and only some parts of digits are disconnected. In the last epoch of the last row, virtually no noise is present and the digits look really convincing. This whole experiment seems to show that the generator using MSE seems to train faster and learn better features than the generator using BCE.

Since the images have started looking so convincing after generating them with BCE and MSE, the BCE images have been shown with original images in the figure 51b to compare and contrast. Notably, the generated images in this figure are sampled by manually picking out the best 8 from a sample of 20 generated images. Comparing the real and fake images, it seems that the largest difference between the two is a number of sharp pixels at random points of the 0's, the 6 and the 5. Some of the digits also look a bit too thick compared to the real images, though this is probably due to the large number of 4's and 1's in the randomly sampled real images. Figure 51a shows MSE generated images compared to real images. Here,

digits like the 3, 2 and 6 have pretty random-looking pixels poking out the side of them. These pixels are most likely more difficult to remove than the salt-and-pepper noise since real digits have a high variance, and as such, some of the floating pixels would have been placed correctly in another iteration of the same digit. Comparing the best examples of generated MSE and BCE images, the difference are very vague. The BCE model seems to have more, smaller blobs of noisy pixels at the perimeter, while the MSE generator creates images with few large noisy concentrations. Overall, the results are closely comparable. The recommendation from this paper is to train on MSE if under time-constraint. Otherwise, one model does not seem to out-perform the other.

Even though the discriminators are not nearly as important after having trained the generators, the discriminator predictions on a number of images have been shown for completeness. The BCE loss model is shown in figure 52. From this figure, a couple points quickly become apparent. It seems the discriminator acts in a risk-averse fashion, never predicting any probability under 0.45 or over 0.75, which is probably negative for the overall performance. This could be due to the generator becoming stronger than the generator and being more able to fool it. Meanwhile, the discriminator performance has high variance. The second figure has almost completely switched the predictions between real and false, while the first and third figure shows the discriminator experiencing more expected behavior. This variability might be a result of very low changes in loss, which do not penalize the discriminator enough for wrong predictions. A solution for this could be to scale the loss function, though this is left as future work. As for the MSE loss models, figure 52 shows much the same patterns as mentioned in the first part of the paragraph. This means that the discriminator was not benefit nor hindered from the different use of loss function. This also might indicate that a future model trained on the MNIST dataset would need to take into action a weaker generator or stronger discriminator.

9.3 Horse2Zebra CycleGAN

9.3.1 Losses

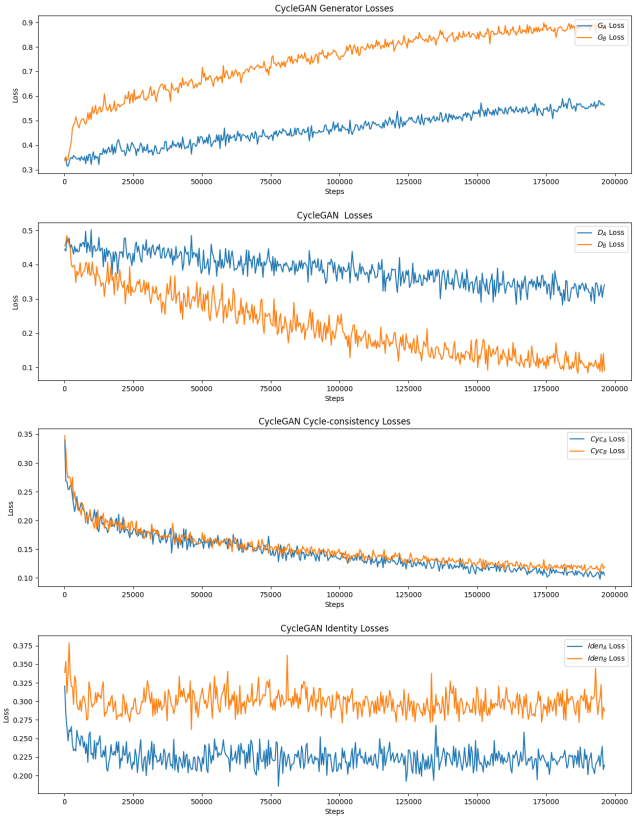


Figure 54: Horse2zebra CycleGAN loss curves. The curves are order to be respectively generator losses, discriminator losses, cycle-consistency losses and identity losses. G_A converts zebras to horse and G_B converts horses to zebras. Meanwhile, D_A predicts validity of horse images and D_B predicts zebra validity.

9.3.2 Progress of Training

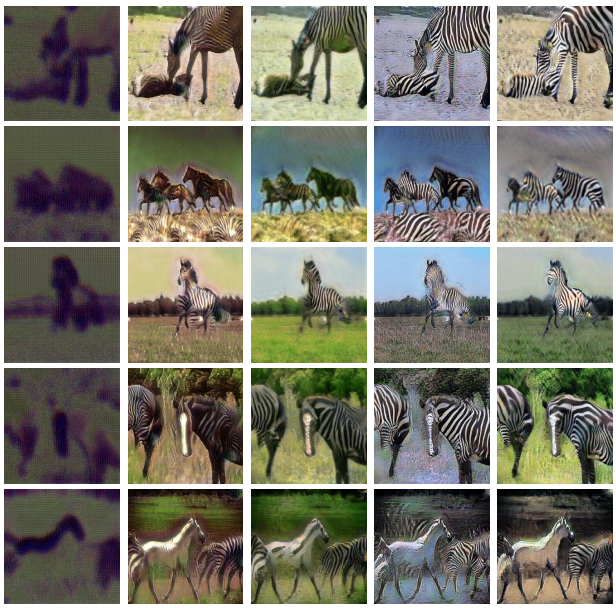


Figure 55: Progress of training the horse2zebra cyclegan. The rows represent unique images and the columns show the generator output after 0, 20, 40, 60, and 140 epochs. Since training converges over time, it has been deemed more interesting to look at the first epochs than the ones later in training. The subjects in this figure are used throughout the rest of this section, and have been chosen as the best generated images in a set of 40 images.

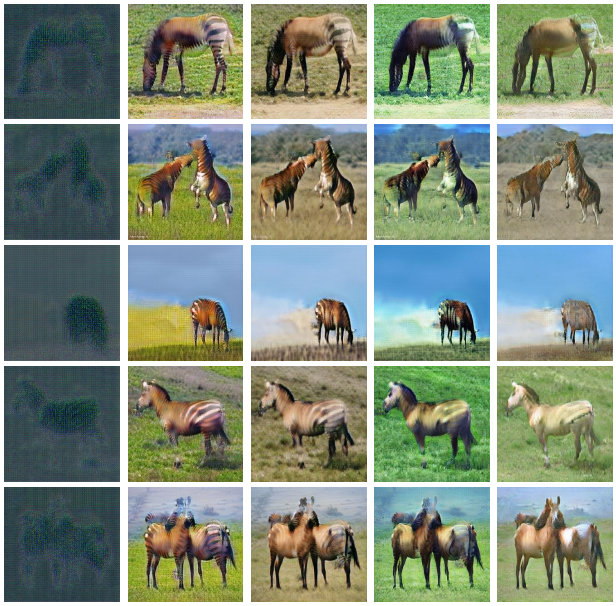


Figure 56: Like in the above figure, this figure shows the progress of training the horse2zebra cyclegan after 0, 20, 40, 60, and 140 epochs. The subjects in this figure are used throughout the rest of this section, and have been chosen as the best generated images in a set of 40 images.

9.3.3 Generated Images and Testing Cycle-Consistency

Figure 57: This figure shows the results of using G_A and G_B to generate zebras and horses respectively. It also shows how well the models fare at restoring the input image when given the generated image. For each block of images, the first column shows the original input image, the second column shows one generator turning the image into the other class and the third column shows the cycle-consistency when the opposite generator turns the image into the other class again. Each row contains one unique subject for each of the classes. Keep in mind that this task is unpaired, so the horse and zebra input images have no relation.

9.3.4 Testing Identity

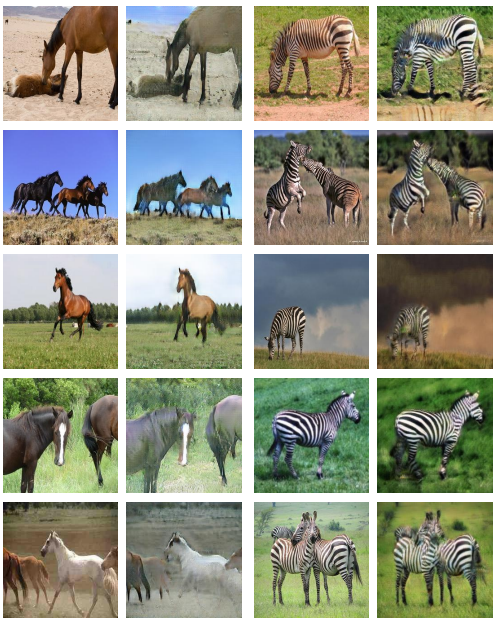


Figure 58: Analyzing how well the model obeys applying the identity transformation when the input image is already part of the output class. The left block of images applies G_A to samples of the horse class and on the right, G_B is applied to zebra class images. In each block, the first column contains the input horse/zebra and the second column shows the generator output. Each column is a subject for each class.

9.3.5 Discussion of Results

In the loss curves of figure 9.3.1, a number of points become apparent. Figure 54a and 54b, it seems that the losses do not experience the same convergence as is commonplace in a model with well-optimized hyperparameters. This could of course happen because the hyperparameters are bad, since a short amount of time has been used to optimize these. Another possibility is that losses for adversarial networks simply look like this sometimes, which is also supported in machine learning circles [38]. This is also not an issue, as will be covered later. The cycle loss in figure 54c does experience this curve though, which might mean that the generator is overly focused on reducing the cycle-consistency loss. Since λ_{iden} was set to 0 for this run, the identity loss does not converge, as expected.

In regards to the progress of training in figures 56 and 55 We notice that the models learn rather quickly to insert stripes/browning and make it more realistic over time. One interesting point is that comparing the epoch show that the images have received a tint of unwanted color. This might be because the model is exploring possible minima, or because the identity loss should have been a value slightly higher than 0.

For the results in figure 57, it is first of all noted that the results are rather impressive for being trained on an unoptimized model in a short amount of time. It is also noted that the model has obviously learned to output horses and zebras even though the generator loss only increased and the discriminator loss only linearly decreased. The zebras are missing many features like eyes and muzzle, although G_B has obviously learned to mostly keep the stripes within the subject horse or horses. As for G_A, this model also learns to transfer the zebras to horses, but some of the stripes are still visible through the browning. The features are also more prevant on these generations, since the browning is a less aggressive pattern. Lastly in regard to the pure generated images, G_B has seemingly also learned to color the background larger and make grass on the beach, while G_A Adds a more blue background and more fresh-looking grass. In regards to the cycle-consistency, almost no images seem to have any major flaws after having been sent through two networks. This shows that λ_{cyc} should probably have been even lower to force the models to learn how to create the other class before learning how to go back again. In general, the models have learned to transfer back to the original image.

In figure 58, it seems that the identity loss might have been a necessary parameter for this network. Many of the images have a completely different saturation and hue. More obvious zebra stripes are added to the zebras and the horses are for some reason made to look more pale. Artifact are als ovisible on a number of the images.

10 References

[1] Van Rossum, Guido et al. "Python 3 Reference Manual". CreateSpace. 2009.

[2] Van Rossum, Guido. "The Python Library Reference, release 3.8.2". Python Software Foundation. 2020.

[3] Paszke, Adam et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". Curran Associates, Inc. 2019. http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[4] Laurits Schultz, Asger, Winkel Holm, Søren. "Pelutils License". Pelutils Developers. 2020. https://github.com/peleiden/pelutils/blob/master/LICENSE.txt

[5] R. Harris, Charles et al. "Array programming with NumPy". Springer Science and Business Media LLC. 2020. https://doi.org/10.1038/s41586-020-2649-2

[6] Hunter, J. D. "Matplotlib: A 2D graphics environment". IEEE COMPUTER SOC. 2007.

[7] Umesh, P. "Image Processing in Python". Citeseer. 2012.

[8] Buslaev, Alexander. "Albumentations: Fast and Flexible Image Augmentations". Information. 2020. https://www.mdpi.com/2078-2489/11/2/125

[9] Pérez, Fernando et al. "IPython: A System for Interactive Scientific Computing, Computing in Science and Engineering", , vol. 9, no. 3, pp. 21-29. 2007. https://ieeexplore.ieee.org/document/4160251

[10] Gorgolewski, Krzysztof et al. "NIPY: an open library and development framework for FMRI data analysis". 2016. https://www.researchgate.net/publication/238805751_NIPY_an_open_library_and_development_framework_for_FMRI_data_analysis.

[11] McKinney, Wes et al. "Data structures for statistical computing in python". Austin TX. 2010.

[12] Bánffy, Ricardo. "PIP Chill - Make requirements with only the packages you need". pypi. 2021. https://pypi.org/project/pip-chill/

[13] Pérez-García, Fernando et al. "TorchIO: A Python library for efficient loading, preprocessing, augmentation and patch-based sampling of medical images in deep learning". Elsevier BV. 2021. https://arxiv.org/abs/2003.04696.

[14] Yep, Tyler. "torch-summary". pypi. 2020. https://pypi.org/project/torch-summary/

[15] Marcel, Sébastien et al. "Torchvision the machine-vision package of torch". Proceedings of the 18th ACM international conference on Multimedia. 2010. https://dl.acm.org/doi/10.1145/1873951.1874254

[16] Pedregosa, F et al. "Scikit-learn: Machine Learning in Python". Journal of Machine Learning Research. 2011. https://scikit-learn.org/stable/modules/model_evaluation.html#mean-squared-error

[17] Paszke, A et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library", Curran Associates, Inc., 2019, https://pytorch.org/docs/stable/generated/torch.nn.BCEWithLogitsLoss.html

[18] Neda et al., "torch.nn.ConvTranspose2d vs torch.nn.Upsample", PyTorch, 2018. https://discuss.pytorch.org/t/torch-nn-convtranspose2d-vs-torch-nn-upsample/30574/7

[19] Burns, Jeffrey M., et al. "Reduced lean mass in early Alzheimer's disease and its association with brain atrophy." Archives of neurology 67.4 (2010): 428-433.

[20] Pedersen, Camilla Kergel "Demographic bias in public neuroimaging databases, and its effect on AI systems for computer-aided diagnosis." University of Copenhagen Faculty of Science. 2021.

[21] Alzheimer's Disease Neuroimaging Initiative homepage: http://adni.loni.usc.edu/, last visited at 28 Nov. 2021

[22] Alzheimer's Disease Neuroimaging Initiative about ADNI1 http://adni.loni.usc.edu/about/adni1/, last visited at 28 Nov. 2021

[23] FreeSurfer Software Suite https://surfer.nmr.mgh.harvard.edu/, last visited at 2nd December, 2021

[24] FreeSurfer: mri_normalize, https://surfer.nmr.mgh.harvard.edu/fswiki/mri_normalize

[25] Basaia, Silvia et al. "Automated classification of alzheimer's disease and mild cognitive impairment using a single mri and deep neural networks." NeuroImage: Clinical, 21:101645, 2019.

[26] Suk, Heung-II and Shen, Dinggang "Deep learning-based feature representation for AD/MCI classification2", 2013

[27] Suk, Heung-II et al. "Latent feature representation with stacked auto-encoder for AD/MCI diagnosis", 2015

[28] Cheng, Danni "CNNs based multi-modality classification for AD diagnosis", 2017 10th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI)

[29] Zhang, Yudong et al. "C lassification of Alzheimer's Disease Based on Structural Magnetic Resonance Imaging by Kernel Support Vector Machine Decision Tree", Progress In Electromagnetics Research, Vol. 144, 171-184, 2014

[30] Alzheimer.dk "Fakta om demens": https://www.alzheimer.dk/viden-om-demens/fakta-om-demens/, last visited at 22nd December, 2021

[31] Jakobsen, Marie et al. "Costs of Informal Care for People Suffering from Dementia: Evidence from a Danish Survey"

[32] Henriksen, Anders, Christensen, Oskar. "Discrimination of Race, Age & Gender in the COMPAS Dataset". Technical University of Denmark. 2020.

[33] He, Kaiming. "Deep Residual Learning for Image Recognition". Cornell University. 2015. https://arxiv.org/abs/1512.03385v1

[34] Persson, Aladdin. "CycleGAN implementation from scratch". YouTube. 2021. https://www.youtube.com/watch?v=4LktBHGCNfw

[35] Zhu, Jun-Yan, et al. "Unpaired image-to-image translation using cycle-consistent adversarial networks." Proceedings of the IEEE international conference on computer vision. 2017.

[36] J. Goodfellow, Ian. "Generative Adversarial Networks". Université de Montréal. 2014. https://arxiv.org/abs/1406.2661

[37] Ayari, Rabeh. "Generative Adversarial Networks". Towards Data Science. 2020. https://towardsdatascience.com/generative-adversarial-networks-gans-2231c5943b11

[38] Brownlee, Jason, "How to Identify and Diagnose GAN Failure Modes", Machine Learning Mastery, 2019, https://machinelearningmastery.com/practical-guide-to-gan-failure-modes/

[39] AnvaMiba, "[Discussion] Discriminator converging to 0 loss in very few steps while training GAN.", Reddit, 2016, https://www.reddit.com/r/MachineLearning/comments/5asl74/discussion_discriminator_converging_to_0_loss_in/

[40] vrao9, "Loss D goes to zero when adding a new loss to generator", Github Issues, 2019, https://github.com/junyanz/pytorch-CycleGAN-and-pix2pix/issues/626

[41] Hui, Jonathan, "GAN — Why it is so hard to train Generative Adversarial Networks!", Medium, 2018, https://jonathan-hui.medium.com/gan-why-it-is-so-hard-to-train-generative-advisory-networks-819a86b3750b

[42] FreeSurfer Software Suite, https://surfer.nmr.mgh.harvard.edu/fswiki/recon-all, last visited 1st of January, 2022.

[43] Bansal, Aayush et al. "Recycle-GAN: Unsupervised Video Retargeting", Carnegie Mellon University, 2018, https://arxiv.org/pdf/1808.05174.pdf

[44] Odena, et al., "Deconvolution and Checkerboard Artifacts", Distill, 2016. http://doi.org/10.23915/distill.00003

[45] Bansal, Hardik. "Understanding and Implementing CycleGAN in TensorFlow". GitHub. https://hardikbansal.github.io/CycleGANBlog/,lastvisited01/01-2022.

[46] Khosla, Ritwek. "Auto-Encoders for Computer Vision: An Endless world of Possibilities". Analytica Vidhya. 2021. https://www.analyticsvidhya.com/blog/2021/01/auto-encoders-for-computer-vision-an-endless-world-of-possibilities/

[47] "The Generator". Google. 2019. https://developers.google.com/machine-learning/gan/generator

[48] "The Discriminator". Google. 2019. https://developers.google.com/machine-learning/gan/discriminator

[49] "GAN Training". Google. 2019. https://developers.google.com/machine-learning/gan/training

[50] Nicholson, Chris. "A Beginner's Guide to Generative Adversarial Networks (GANs)". Pathmind. 2020. https://wiki.pathmind.com/generative-adversarial-network-gan.

[51] Hannemose, Morten. "CycleGAN walktrough". Technical University of Denmark. 2019. http://summer-school-gan.compute.dtu.dk/3CycleGanWalkthrough.pdf.

[52] Ding, Yan. "Calculating Parameters of Convolutional and Fully Connected Layers with Keras". Medium. 2020. https://dingyan89.medium.com/calculating-parameters-of-convolutional-and-fully-connected-layers-with-keras-186590df36c6.

[53] Jayathilaka, Mirantha. "Understanding and optimizing GANs (Going back to first principles)". Towards Data Science. 2018. https://towardsdatascience.com/understanding-and-optimizing-gans-going-back-to-first-principles-e5df8835ae18

[54] Wang, Tongzhou and Lin, Yihan. "CycleGAN with Better Cycles". University of California. https://ssnl.github.io/better_cycles/report.pdf

[55] Arjovsky, Martin et al. "Wasserstein GAN". 1Courant Institute of Mathematical Sciences. 2017. https://arxiv.org/abs/1701.07875

[56] Gulrajani, Ishaan et al. "Improved Training of Wasserstein GANs". Montreal Institute for Learning Algorithms. 2017. https://arxiv.org/abs/1704.00028

[57] Snoek, Jasper et al. "Practical Bayesian Optimization of Machine Learning Algorithms". NeurIPS. https://proceedings.neurips.cc/paper/2012/file/05311655a15b75fab86956663e1819cd-Paper.pdf

[58] FreeSurfer Software Suite, https://surfer.nmr.mgh.harvard.edu/fswiki/recon-all, last visited 1st of January, 2022.

[59] Evans, A.C., Collins, D.L., Milner, B., 1992a. An MRI-based stereotaxic atlas from 250 young normal subjects. Proc 22nd Annual Symposium, Society for Neuroscience, 18, p. 408

[60] Collins, D.L., Neelin, P., Peters, T.M., Evans, A.C., 1994. Automatic 3-D intersubject reg- istration of MR volumetric data in standardized Talairach space. J. Comput. Assist. Tomogr. 18 (2), 192–205.

[61] Keck School of Medicine of USC University of Southern California https://www.loni.usc.edu/, last visited 2nd of January 2022

[62] Callaghan, Paul T "Principles of Nuclear Magnetic Resonance Microscopy", 1994

[63] Callaghan, McRobbie, Donald et al. "MRI from Picture to Proton", Cambridge University Press, 15. feb. 2007

[64] Magnetic resonance imaging https://en.wikipedia.org/wiki/Magnetic_resonance_imaging, last visited 2nd January 2022

[65] Ministry of Foreign Affairs of Denmark, "Denmark Paves the Way for the Implementation of Trust by Design", 25.03.2019, https://investindk.com/insights/denmark-paves-the-way-for-implementation-of-trust-by-design, visited 2nd January 2022

[66] Raponi, Mitch et al. "MicroRNA Classifiers for Predicting Prognosis of Squamous Cell Lung Cancer", Published July 2009

[67] Liu, Jun et al. "Computer Vision for Human–Machine Interaction" Computer Vision and Pattern Recognition 2018, Pages 127-145

[68] Goodfellow-et-al-2016. Deep Learning, MIT Press in 2016.

[69] Cadima, Jorge et al. "Principal component analysis: a review and recent developments", Published:13 April 2016

[70] ScikitLearn `sklearn.decomposition.PCA`, see more at: https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html

[71] Maaten, Laurens van der "Visualizing Data using t-SNE", Journal of Machine Learning Research 9 (2008) 2579-2605

[72] Evans, Erik "MRI Advantages 1.5T Vs. 3T MRI: One Size Does Not Fit All", Jan 27, 2021. Website: https://kingsmedical.com/mri/mri-advantages-1-5t-vs-3t-mri/

[73] Neurometrika DARTEL. Can be visited at, https://www.neurometrika.org/node/34