SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY — INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Robotics, Cognition and Intelligence

Full Waveform Inversion Using Generative Adversial Network Regularizers

Yize Jiang

SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY — INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Robotics, Cognition and Intelligence

Full Waveform Inversion Using Generative Adversial Network Regularizers

Full-Waveform Inversion mit Regularisierung durch GAN

Author:	Yize Jiang
Supervisor:	Dr. Felix Deitrich
Advisor:	Qing Sun
Submission Date:	15.12.2023

I confirm that this master's thesis is my own work and I have documented all sources and material used.

Munich, 15.12.2023 Yize Jiang

Acknowledgments

Sincere thanks to Qing Sun for all and all and all the help and support I have received regarding this paper. I would like to thank Dr. Felix Dietrich for offering the opportunity to work with the project and for all the precious comments and guidance.

Thank you so much for all the friends in Munich, who accompanied me through all the difficulties. Great thanks to my parents, my friends and my therapist in China, who supported me all the time as always.

Finally, many thanks to me myself and I, in past present and future.

Abstract

Full-waveform inverse problem is a significant research field, embedded with problems and solutions from different studies. The boost of deep learning and neural network lead to the occurrence of more and more data-driven solutions, therefore, the study in this field is more specifically divided.

In this paper, the regularization part of the full-waveform inverse problem is researched by using a generative adversarial network(GAN). From building of the scenario, mimicking the defect problem set, to the training process of the GAN and the evaluation of the formed regularizer. To conclude, the paper draws the full methodology to deal with problem and the possibility of the regularization through a GAN trained discriminator is validated. The regularizer is evaluated as valid in distinguishing between real images and counter facts.

Contents

1 Introduction

Inverse problem is a set of ill-conditioned and ill-posed problems, which has been gaining continuous attentions of scholars from different study fields. Full waveform inverse problem as a subset of the problem, which researches on reconstruction of the subsurface properties by seismic data, is also well researched.
Since the fast development of machine learning, deep learning and neural networks, we gain more capacity of solving inverse problem with data-driven method. And more and more techniques and details about the machine learning and deep learning has been raised, and the study field is more and more sophisticated.

This project intends to meet the need of quick structural health monitoring, detecting and reconstructing the defect properties of built structures, using the pressure field data. The current work related to the problem has drawback of difficult interpretation, as most of them are knowledge-based inversions. Also, it would not be possible to be broadly applied due to the specific characteristics of each cases. In turn, they try to seek for data-driven solutions, as for the data-driven solutions, once the model is trained, the interpretation cost would be relatively small and the results can be shown relatively quick.
In this data-driven, machine learning, model fitting scenario, the regularization of the training would be a very important part in regard of the performance. As regularizer would prevent the model from overfitting and also add on reward and punishment degree to better control the training. And the usual regularizer uses fixed-norm, or regularizer that pair with the loss function.
However, in the full waveform inversion problem set, the regularization framework is of great value. It would help improve the reconstruction performance. In the case of structural health monitoring, it is not possible to see the possible structure of the defect directly, therefore, we propose a methodology to mimickthe defect, and the gain the possibility to apply the training scenarios to all different environments according to the requirements. In this way, it would help the main inverse model to be easily fitting different scenarios and requirements.

Therefore, this paper tries to build a regularizier that would distinguish between the real or the optimal structure and the counter facts or unrealistic structures. Such a

regularizer has the same property as a discriminator in a GAN model. Therefore, this paper tries to validate the possibility of building a regularizer using a GAN trained model.
The paper contributes in the methodology, that using a dataset of mimicking defects to train the GAN model, to generate the regularizer ability. The building of the dataset is easily transferrable, making the inverse problem applicable to all kinds of requirements. The trained out generator would be possible to use as further source of counter facts. And the regularizer is validated as possible to distinguish the images and some of the counter facts.
And there is plenty of research space to keep adding on this field. Through a proper regularizer, that is trained from a GAN model. It is possible to improve the performance of the inverse solution is a large scale.

2 Related Work

This research project functions as a sub-project of a defects detection project. The defect detection project uses ultrasonic detect techniques, aiming at predicting a wave velocity distribution given a pressure field. In the project, three different ways of solving inverse problem would be examined for the wave function problem set, including data-driven deep learning, iterative forward solver or physics-informed network.
The defect detection solution is an inverse problem. The specific ultrasonic detection techniques categories the problem into the full-waveform inverse problem. As the project deals with different techniques in this field, the article stands on the regularization for the problem. And it is proposed to use a GAN trained neural network as a discriminator.
Therefore, from the relevant perspectives, this chapter is divided into the inverse problem, full waveform inverse problem, deep learning(data-driven solutions), regularization, GAN relevant and the training techniques.

2.1 inverse problem

An inverse problem is a scientific process of calculating the causal factors that produced a set of observations. Inverse problems are some of the most important mathematical problems in science and mathematics because they tell us about parameters that we cannot directly observe. They have wide application in system identification, optics, radar, acoustics, communication theory, signal processing, medical imaging, computer vision, geophysics, oceanography, astronomy, remote sensing, natural language processing, machine learning, nondestructive testing, slope stability analysis and many other fields.[Ger70] The inverse problem in ultrasonic transmission tomography system is encountered in practical implementation, which consists of reconstructing an image that is an estimation of an unknown object from a finite set of projection data[Ara+18]. Reconstructive algorithms used in transmission tomography are based on linear mathematical models, which makes it necessary to process non-linear data into estimates for a finite number of projections.(Rymarczyk, T. et al, 2021) The application of transformation methods requires building a mathematical model in which the projection data forming the known and unknown quantities are functions with arguments from a continuous set of real numbers, determining the function describing the unknown quantities sought in the form of inverse relation and adapting it to operate on discrete and noisy data.(S. Goudarzi et al, 2022)

Seismic exploration is often used to map the structure of subsurface formations based on the propagation of seismic wave on the Earth.[Ois+] It can estimate the physical properties of the Earth's subsurface mainly from reflected or refracted seismic wave. Since seismic exploration is capable of detecting target features from a large to small scale, it plays an important role in the delineation of near-surface geology for engineering purposes, hydrocarbon exploration, as well as the Earth's crustal structure investigation.[Ara+18][Li+20] Usually, artificial sources of energy are required and a series of receivers are placed on the surface to record seismic waves. One major outcome of processing the recorded data is the reconstruction of the subsurface velocity model, namely seismic velocity inversion, which has a substantial impact on the accuracy of locating and imaging target bodies. Recently, in addition to stochastic inversion strategies, by using full-waveform information of seismic data, full-waveform inversion (FWI) is now one of the most appealing methods to reconstruct the velocity model with high accuracy and resolution.

2.2 Full-Wave Inversion

The inversion of full-waveform equation is very computationally demanding with no guarantees of global convergence. Additionally, the problem is ill-posed, the existing solution may not being unique, and is ill-conditioned, the output being highly sensitive to the input, making the processing of noise from input highly demanding. Therefore, a data-driven deep learning model has been widely applied recently to improve the performance.

Full Waveform Inversion (FWI) was firstly introduced as a method to reconstruct the velocity model by minimizing the difference between seismic data and synthetic data in a least-squares sense. Conventional FWI uses gradient-based solvers to update the model parameters, and the gradient is normally calculated through backward wavefield propagation of data residuals based on adjoint-state methods. However, seismic velocity estimation from observed signals is a highly nonlinear process, so the conventional iterative algorithm usually requires a good starting model to avoid local minimum. Moreover, FWI faces severe nonuniqueness due to inadequate observation or observation data contaminated with noise. To address these issues, geophysicists have proposed many improvements, such as the multiscale strategy, processing of seismic data in other domains, and so on.

2.3 Deep Learning(data driven solutions)

Deep neural networks (DNNs) face several challenges when it comes to seismic data analysis. [Li+20]One of the main challenges is the weak spatial correspondence and uncertain reflection-reception relationship between seismic data and velocity model, which makes it difficult to estimate seismic velocity from observed signals. The conventional iterative algorithm usually requires a good starting model to avoid local minimum. Furthermore, due to the ill-conditioned problem setting, the corrupted data due to existence of noise in experiments would be another obstacle to overcome, which would also worsen the impact of nonlinearity during prediction.

However, recent developments in DNNs have demonstrated remarkable ability to approximate nonlinear mapping functions between various data domains, such as images and label maps, images and text, and different types of images, especially for inverse problems, such as model/image reconstruction, image super-resolution, and real-world photosynthesis[**empty citation**]. These state-of-the-art developments bring new perspectives for seismic inversion and velocity model reconstruction. DNN-based seismic inversion is to learn the mapping function F from seismic data to velocity model.

Some work has already made progress on this task. Moseley et al. achieved 1-D velocity model inversion by WaveNet after depth-to-time conversion of the velocity profile. Araya-Polo et al. used convolutional neural networks (CNNs) to reconstruct velocity model from a semblance cube calculated from raw data. These two approaches may introduce biases because of the human intervention in seismic data processing. Other than data processing, Wu et al. proposed InversionNet to build the mapping from raw seismic data to the corresponding velocity model directly by Autoencoder architecture, which decodes the velocity model from an encoded embedding vector. Since data are extremely condensed in the embedding vector, the decoded velocity model is more robust to noise and less prone to overfitting.

In general, there are three main characteristics that pose a great challenge for deep neural networks (DNNs) when it comes to seismic data inversion. Firstly, the spatial correspondence between raw time-series signals (seismic data) and seismic images (velocity model) is weak, especially for reflected seismic signals. This means that the position on the velocity model that corresponds to a reflected wave on the seismic profile may not contain any interface and vice versa. Secondly, the complexity of subsurface structure in various velocity models makes it difficult to determine the corresponding interfaces that cause the reflections in velocity model for reflected seismic signals received by one receiver from one source (i.e., seismic trace). We refer to this characteristic as the uncertain reflection-reception relationship. Thirdly, seismic data are of time-varying property, which means that the recorded seismic wave gradually weakens as time goes by. This makes seismic pattern hard to capture with fixed kernel. All these characteristics make it challenging for DNNs, especially for convolutional neural networks (CNNs) with spatial correspondence and weight-sharing properties.

2.4 Regularization

Deep learning has already been widely used in the inverse problem solving, especially in image processing procedure. And the structure plays a significant role in the performance of the model. The sub-project is designed to improve the performance of the data-driven deep learning methods in inversion of full-waveform equation through the modification of the regularisation mechanism, trying to support overcoming the challenges resulting from the noise corruption in a structural way.

This would probably in turn improve the overall performance of the model due to the endogenous relation between noise corruption and nonlinearity, time-variety and uncertainty in relation between input and output(seismic data and velocity model).

2.5 GAN-Regularizer

Regularization is a technique that adds some constraints or penalties to the inverse problem to make it more stable and well-posed. In regard of regularization of the the inversion problem, there are two types of techniques to regularize the ill-posed inverse problems.[LSÖ]

2.5.1 Knowledge-driven

Traditional knowledge-driven approaches to inverse problems are based on functional analytic inversion and Bayesian inversion, which rely on physical–analytical models and prior information to regularize the ill-posedness of the problem.

Functional analytic inversion is based on the theory of Fredholm integral equations, which can be solved using methods such as Tikhonov regularization, sparsity-promoting regularization, low- rank regularization, and manifold regularization[Arr+19]. These methods use different norms or functions to measure the smoothness, sparsity, low-rankness, or manifold structure of the model parameter, and add them as penalty terms to the least-squares objective function.

• Tikhonov regularization: This is a classical method that adds a penalty term to the least- squares objective function to stabilize the solution. The penalty term is proportional to the norm of the model parameter, and the proportionality constant is called the regularization parameter. The paper discusses how to learn the regularization parameter from data using cross-validation or Bayesian inference.

• Sparsity-promoting regularization: This is a method that imposes sparsity on the model parameter by using norms or functions that are non-smooth or non-convex, such as the L1- norm or the L0-norm. The paper reviews some of the algorithms for solving sparse inverse problems, such as iterative shrinkage-thresholding algorithm (ISTA), fast iterative shrinkage- thresholding algorithm (FISTA), alternating direction method of multipliers (ADMM), and proximal gradient methods.

• Low-rank regularization: This is a method that exploits the low-rank structure of the model parameter by using matrix or tensor decompositions, such as singular value decomposition (SVD), principal component analysis (PCA), or Tucker decomposition. The paper presents some of the applications of low-rank regularization in image reconstruction, video completion, and hyperspectral imaging.

• Manifold regularization: This is a method that incorporates prior knowledge about the manifold structure of the model parameter by using graph-based or kernel-based techniques, such as graph Laplacian regularization or manifold kernel learning. The paper shows some of the examples of manifold regularization in image denoising, super- resolution, and inpainting.

• Bayesian inversion is a statistical framework that uses prior information and likelihood models to infer the posterior distribution of the unknown model parameter from the noisy observations. By choosing different the prior distribution and the likelihood model in a Bayesian inversion framework for different problem set, the ill-posedness would be able to be regularised.

• prior distribution: Using Gaussian prior distribution with a covariance matrix that incorporates spatial correlation and physical bounds on the slip. A Gaussian likelihood model with a covariance matrix would also be possible to function as account for data noise and model errors.

• posterior distribution: Using Markov chain Monte Carlo (MCMC) methods, which are stochastic algorithms that generate samples from the target distribution. It this way, the problem set can be transformed into a generated a well-posed problem.

2.5.2 Data-driven

The data-driven techniques are methods that use data-driven models, such as deep neural networks, to learn from data and incorporate domain-specific knowledge to deal with ill- posedness and ill-conditionedness.

• Deep neural network regularization: This is a method that uses deep neural networks to learn a nonlinear mapping from the observations to the model parameter or to learn a nonlinear regularization operator that enforces prior information. Some of the architectures and training strategies for deep neural network regularization have already occur in the existing literature, such as convolutional neural networks (CNNs), recurrent neural networks (RNNs), variational autoencoders (VAEs), and generative adversarial networks (GANs).

2.5.3 Adversarial regularizer

In the paper, "Adversarial Regularizers in Inverse Problems," [BKM17] a new framework was proposed for applying data-driven approaches to inverse problems using a neural network as a regularization functional. The network learns to discriminate between the distribution of ground truth images and the distribution of unregularized reconstructions. Once trained, the network is applied to the inverse problem by solving the corresponding variational problem. The algorithm can be applied even if only unsupervised training data is available. The authors demonstrate the potential of the framework for denoising on the BSDS dataset and for computed tomography reconstruction on the LIDC dataset.

In "Learned Regularizers for Inverse Problems", the paper proposes a class of algorithms that builds on the well-established variational framework, training a neural network as a regularization functional. These approaches come with the advantage of a theoretical understanding and stability theory that is built on existing results for variational regularization. In comparison to “Adversarial Regularizers in Inverse Problems,” which also proposes a new framework for applying data-driven approaches to inverse problems using a neural network as a regularization functional, “Learned Regularizers for Inverse Problems” focuses on learning priors from data directly, with the goal of obtaining a more realistic and detailed image representation.

2.5.4 Evaluation

A complete mathematical analysis of a regularization method includes proving convergence rates and stability estimates, in addition to existence, stability, and convergence. Convergence rates provide an estimate of the difference between a regularized solution R θ (g) and the solution without regularization. Stability estimates, on the other hand, provide a bound on the difference between the true data and the experiment collected data, depending on the error term ||e||.

2.5.5 GAN model

A GAN training is a process of learning to generate realistic data using a generative adversarial network (GAN). A GAN is a type of neural network that consists of two parts: a generator and a discriminator. The generator tries to create fake data that look like the real data, while the discriminator tries to distinguish between the real and fake data. The two parts compete with each other in a game-like scenario, where the generator aims to fool the discriminator, and the discriminator aims to catch the generator. The GAN training alternates between updating the parameters of the generator and the discriminator, until they reach a balance where the discriminator cannot tell the difference between the real and fake data.

Some of the benefits of GAN training are that it can generate novel and diverse data, such as images, text, audio, or video, that are not present in the original dataset. It can also learn the underlying distribution of the data, which can be useful for tasks such as data augmentation, anomaly detection, or domain adaptation. However, some of the challenges of GAN training are that it can be unstable, difficult to converge, or suffer from mode collapse, where the generator produces only a limited variety of data. Therefore, many techniques and tricks have been proposed to improve the stability and quality of GAN training, such as using different loss functions, architectures, regularization methods, or evaluation metrics

2.6 training technique

2.6.1 batch normalization

Batch normalization is a technique used in deep learning to improve the performance of neural networks by normalizing the inputs of each layer. It is a type of normalization that is applied to the outputs of a layer, before the activation function is appliedBatch-Norm1d is a PyTorch module that applies batch normalization over a 2D or 3D input as described in the paper "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift" [IS15]
The primary use of batch normalization is to accelerate the training of deep neural networks and to reduce the number of training epochs required to train deep networks 12. It also helps to reduce the internal covariate shift and makes the training process more stable.
This is an essential layer in the linear deep model, the batch normalization layers make the training faster, and allow a wider range of learning rate without compromising the training convergence.

2.6.2 reparameterization

The reparameterization trick is a powerful method that makes the training of Variational Autoencoders (VAEs) possible and efficient. By cleverly separating the random and deterministic elements of the sampling operation in the VAE, it allows us to leverage the power of backpropagation while maintaining the stochastic nature of the model. The trick involves reparameterizing the random variable in the VAE's latent space so that it can be expressed as a deterministic function of the random noise variable and the model parameters. This allows us to compute gradients with respect to the model parameters using backpropagation, which is essential for efficient training.[HS18] [KW22]
When computing the gradient of an expectation, i.e. $\mathbb{E}_{p(z)}[f_\theta(z)]$,where p is a density. it is easily to calculate out, the gradient of the expectation is equal to the expectation of the gradient.

$$\begin{aligned}\nabla_\theta \mathbb{E}_{p(z)}[f_\theta(z)] &= \nabla_\theta \Big[\int_z p(z) f_\theta(z) dz \Big] \\ &= \int_z p(z) \left[\nabla_\theta f_\theta(z)\right] dz \\ &= \mathbb{E}_{p(z)} \left[\nabla_\theta f_\theta(z)\right]\end{aligned}$$

However, if the p is also the function of θ, then the expectation would change to more complicated result:

$$\begin{aligned}
\nabla_\theta \mathbb{E}_{p_\theta(z)}[f_\theta(z)] &= \nabla_\theta \left[\int_z p_\theta(z) f_\theta(z) dz\right] \\
&= \int_z \nabla_\theta [p_\theta(z) f_\theta(z)] dz \\
&= \int_z f_\theta(z) \nabla_\theta p_\theta(z) dz + \int_z p_\theta(z) \nabla_\theta f_\theta(z) dz \\
&= \int_z f_\theta(z) \nabla_\theta p_\theta(z) dz + \mathbb{E}_{p_\theta(z)} \left[\nabla_\theta f_\theta(z)\right]
\end{aligned}$$

The first term of the last equation is not guaranteed to be an expectation. It is possible we can sample from $p_\theta(z)$ but it is not possible to get the gradient. Since$\nabla_\theta p_\theta(z)$ is not analytic.
What is done by reparameterization, is that we set:

$$\begin{aligned}
\boldsymbol{\epsilon} &\sim p(\boldsymbol{\epsilon}) \\
\mathbf{z} &= g_\theta(\mathbf{ffl}, \mathbf{x})
\end{aligned}$$

Then it is clear that:

$$\begin{aligned}
\mathbb{E}_{p_\theta(\mathbf{z})}[f(\mathbf{z}^{(i)})] &= \mathbb{E}_{p(\boldsymbol{\epsilon})}[f(g_\theta(\boldsymbol{\epsilon}, \mathbf{x}^{(i)}))] \\
\nabla_\theta \mathbb{E}_{p_\theta(\mathbf{z})}[f(\mathbf{z}^{(i)})] &= \nabla_\theta \mathbb{E}_{p(\boldsymbol{\epsilon})}[f(g_\theta(\boldsymbol{\epsilon}, \mathbf{x}^{(i)}))] \; (1) \\
&= \mathbb{E}_{p(\boldsymbol{\epsilon})}[\nabla_{\boldsymbol{\theta}} f(g_{\boldsymbol{\theta}}(\boldsymbol{\epsilon}, \mathbf{x}^{(i)}))] \quad (2) \\
&\approx \frac{1}{L} \sum_{l=1}^{L} \nabla_{\boldsymbol{\theta}} f(g_\theta(\epsilon^{(l)}, \mathbf{x}^{(i)})) \quad (3)
\end{aligned}$$

In this way the reparameterization express a gradient of an expectation (1) as an expectation of a gradient (2). Provided $g_\theta$is differentiable, then it is possible to sample and estimate $\nabla_\theta \mathbb{E}_{p_\theta(\mathbf{z})}[f(\mathbf{z}^{(i)})]$.[Doe21] [MDA]
This reparameterizaton technique is essential to the training of VAE and GAN.

To conclude, this chapter lists the project related work. The full-wave inverse problems in regard of defects detection, and GAN-trained models and regularization are all significant research areas and topics, this paper stand on the point to combine the related work and current techniques together, trying to apply them to the new scenarios to solve the essential problems.

3 methodology

This chapter introduces the methodology that would be used in the experiment, i.e. the research question settings of the paper. What kind of problem shall be dealt with or solved during this study, what kind of methods would be used to achieve the goal, and what kind of evaluations would be used to evaluate on the goal.

3.1 problem setting

The project tries to build a regularizer that would help the full-waveform inversion problem. Therefore, the whole task for the regularizer is to distinguish between the real solutions and fake solutions, a model that would give pseudo inversion solutions more punishment than the possible inversion solutions, so that the the learning process of the inverse problem itself would be trained or updated in the right direction. So the solution to the inverse problem would not be overfitted, giving only a databased solution, or being updated around too generated result, that would not form proper solution.
However, it is not possible to see the solutions that would be calculated from the inversion problems in advance, therefore, it is not possible to form a regularizer that would be able to regularize on all kinds of "unrealistic" defects calculated from the data-driven model. Therefore, a regularizer with the possibility of being fitted into different defects scenarios is needed.

From the literature of related work in chapter 2, we can clearly conclude that, it is possible to give the regularizer with knowledge-driven base, build up a more physical/reality-based model, mimicking the defect in the way that should be possible in reality and use the model to rule out the other "fake" defects. However, the knowledge-based regularizer would be in need of great amount of modelling and very complex physical calculation, and the scenarios is not universally applicable. It is highly case-dependent, which would cause difficulties in using the regularizer for different kinds of application scenarios.

Therefore, the data-driven based methods was preferred in our project. It would be possible to achieve better results from the data-driven mind, using the neural networks to gain the knowledge from large amount of cases. The possible advantages with the data driven solutions would be, after the learning process, it would be possible to get a model that is easier calculated or more competent than the knowledge based model, as the model can learn information that may not be modeled by a knowledge based model, or the trivial differences that would be simplified by a universally used knowledge-based model would have case-dependent significant effect. Furthermore, the regularizer would be easier to be applied to other scenarios when equipped with enough data. With the data-driven regularizer, it would be possible to solve the regularization problem with intuitive images of the defects, rather than comprehensive analysis of the situation in advance to fit for the knowledge based data model for hyper-parameters, which requires much higher qualifications than feeding the image data to the model. Therefore, the realization of a data-drivien regularizer would be very promising.

3.2 difficulties

According to problem setting, we choose the data-driven regularizer as the preferred answer. However, the difficulty also comes along with the data advantages. It is difficult to gain the data, therefore, difficult to train. The difficulty comes both from the authentic data side and the counterfactual data side.

For the authentic data, it is not possible to directly gain the defect data that would be enough for training. If modelled from the knowledge-based model, it would also face the problem of large amount of calculation and model simplification.

For the counterfactual data, that should be the solutions calculated from the data-driven model for the whole inverse problem. If trained with solutions, the regularization for the solution itself would be not realiable. Therefore, it is also not possible to have reliable counterfactual "fake" data to train the regularizer.

3.3 entry point

In this experiment, we try to partly solve the problem, finding possible and feasible solutions to the possible way of training such regularizers, so that the regularizer would

be able to distinguish the real defects and the fake defects.
Essentially, this paper tries to validate the possibility of training such regularizers in a relatively more practical sense, since there is no easy direct way of gaining training data and train such regularizers in reality. The paper tries to build up a pseudo scenario that would be easily extended or applied to other scenarios. The training process of the regularizer would be validated in this pseudo scenario, so it would be possible to see the critical points of training, the possible difficulties and corresponding solutions.In this way, it would be possible to further transform the results to other scenarios.

Regarding the substitute of the authentic data, the pseudo scenario, it is important, that the building of the pseudo scenario should be easily not too complex and tedious to build up, unlike direct knowledge-based modelling. At the same time, it shall perform due reconstruction of the reality, with representative characteristics. Additionally, it shall be easily transferable, meaning it is possible to change the specification of the training data structure according to the characteristics of different scenarios. This means there shall be enough extend ability of the building scenarios.
Apart from the requirements for the authentic data, it also important to find solutions for the counter-facts. On one hand, the counter-facts should be reasonable for be trained against the authentic data/pseudo scenario, the counter-facts shall be the "impossible solutions" of the inverse problem. This requires the generalization ability of the counter-facts, i.e. it is not specific to only one type of counter-fact, meaning the regularizer would be stable against all kinds of noised/fake images. On the other hand, it should have the capacity to deal with specific noise and fake "attack".

3.4 solving methods

Based on the explanation in section 3.3, the database would be created on with a easy accessible way, and a GAN-trained regularizier is used for training. The GAN would be trained on the built-up pseudo datas, and the adversarial part would be considered as a good regularizer.
In regard of pseudo scenarios, a set of data would be created for the experiment and the training. It would be easy to change the hyper-parameters of the data structure, and apply it to other scenarios. The details would be explained in the chapter 4
In regard of counter-facts, the training process of the GAN-generator, would be considered as a process trained by "counterfactual generator", during which the regularizer is exposed to all different kinds of counterfacts (unlearned results from the generator), that would be considered as a good learining process for the regularzier to be trained.

3.5 evaluations

In the related work, the evaluation of a regularizer is usually directly bounded with the solving of the inverse solution, observed with the performance, whether it help improve the performance of the solution or not.
In the case of our full-waveform inversion, it would be very time and computation consuming, to simply test the performance of the regularizer. Therefore, there should be some ways of evaluation that is independent from the inversion solving procedure, as a prediction of the regularizer performance.
The regularizer should be able to distinguish between the authentic images and the counterfactual images. It is trained in this way through the adversarial networks. And for the evaluation, the counterfacts from the generators are not available anymore, and have already been feed to the regularizer. Therefore, in the experiments, the evaluation is carried out using the manually noised images, this would be illustrated in detail in chapter 6. The regularizer would go through all the noised images and see the score given by the regularizer, to check if the threshold for authentic and counterfactual images exist.

4 Data creation

This chapter is about the data structure used in the experiments. It would illustrate how the data-sets used in the experiments would be created, and the considerations behind this process.

The data structure is vital to the experiment, as it is related to the performance of the discriminator/regularizer being created, and the evaluation of the results. The data structure would on one hand, affect the training difficulty of the model, as the training process and the model hyper-parameters shall be adjusted due to different dataset characteristics, on the other hand, the characteristics of the data would also impact the generalization ability of the model, and also the evaluation of this ability. To be more specific, the degree of freedom of the dataset, would directly impact the training diffculty: with a large number of degrees of freedom in the dataset, and a sparse sample along the DOF, it would be difficult for the model to gain the information. Furthermore, how identical the distributions of the validation set and test set are to the training set, would directly impact the evaluation of the robustness and generalization ability. Whether those distributions are identical to each other, would be related to characteristics of the data-set, i.e. the DoF and whether the samples are representative enough for the distribution.

The data used in the experiments are virtually created data that are intended to feature the possible characteristics of detected defects. The data set would consider the appearance of a possible spline, and try to mimic the defect with relative simple methods.

In this way, it would be possible to easily extend the data-set into other categories of defects (not included in current experiment work), and test the robustness of the model of this experiments and validate the reliability of the training process when using other types of the data. There is always this trade-off between robustness and generalization ability of machine learning model. The manually-created data-set, would make it easier to test the ability and adjust due to realistic requirements.

4.1 Dataset

The original database contains 128x128 images with a defect in the middle of the picture.

The training datasets contains 128x128 images with a "low DOF" defect. The validation datasets contains 128x128 images with a "low DOF" defect. The test datasets contains 128x128 images with a defect.

4.2 Generating procedure

The generating procedure illustrates, how the images would be generated, especially how the spline of the defect would be generated. Generally, the spline would be represented as a randomly weighted combination of 5 random normal distribution. To simplify the training procedure, and control the DOF, the training spline was generated with a plain combination of two normal distribution.

4.2.1 Sampling

The training dataset contains the combination of 2 normal distributions. With a random mean and variance of the normal distribution, the dataset is featured with four DOF. To avoid sparse sampling, the mean and variance was sampled through a linear space each with 16 samples, creating a pseudo-random combination of the mean and variance of the normal distribution. With the four DOF, the dataset contains 65,536 images.

4.2.2 Normal distribution

The random mean is selected from the linear sample space of (-64,64), and the random variance is selected from the linear sample space of (16,32).

A variance number selected from other interval would cause either no observable variance in the plot or too abrupt turning, as illustrated by figure 4.1 A small variance would lead to abrupt shot in the plot, due to the limited sample capacity within 128 pixels, causing sparse sample results with a large breaking space between sampled points.

With the pair of norm and mean, the normal distribution is sampled with an x-axis between (-64,64) with a sample interval of 1, which would create a 128 samples on the

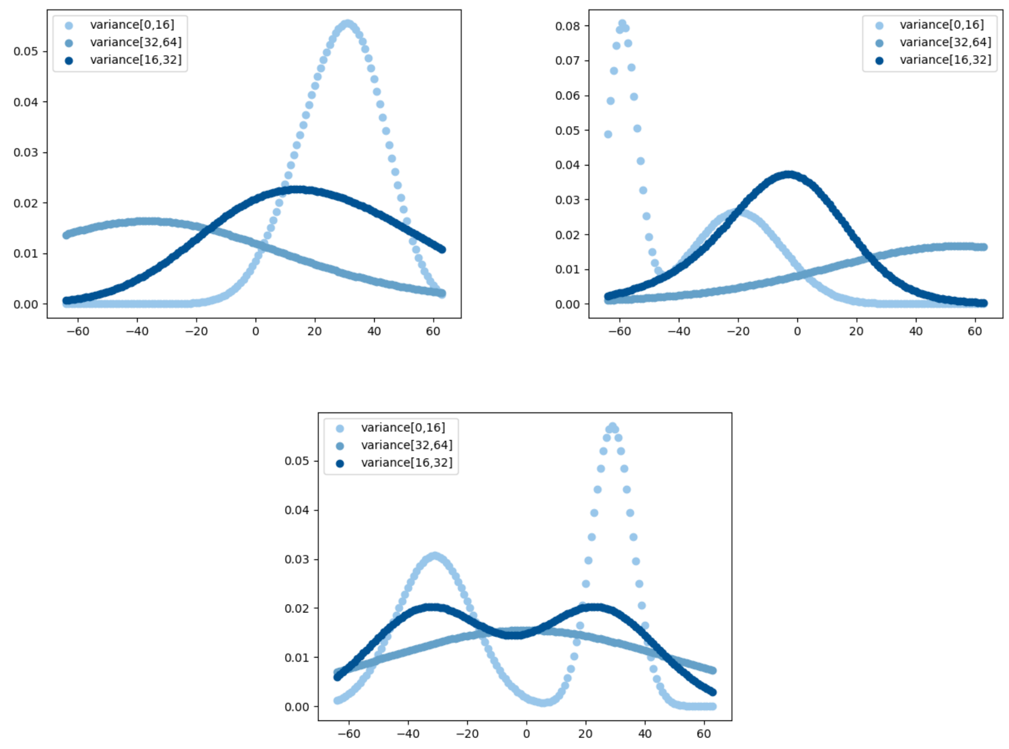


Figure 4.1: Comparison between difference variances sampling

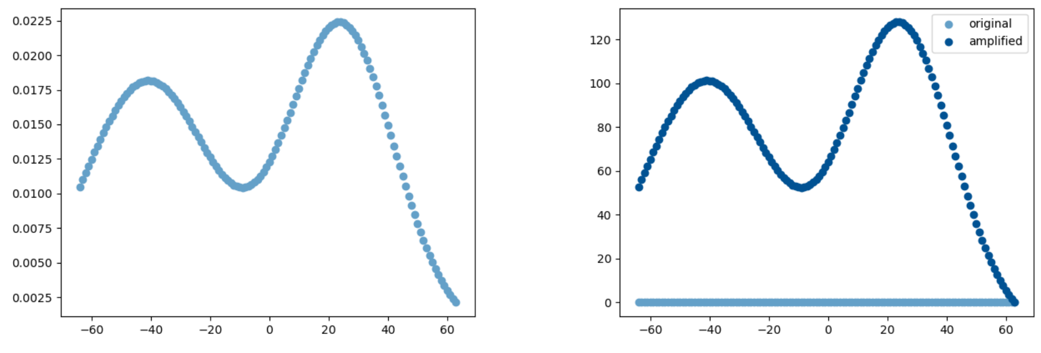
image, corresponding to the final image size.

4.2.3 Amplifying

To make the spline positioned in a proper place, and make the variance of the normal distribution observable for further extension, the normal distribution is amplified with a constant of 128.

The amplification process first uses a min-max normalization, to project the normal distribution to 0-1 range. Then, a amplifier constant is applied to the new distribution, resulting in a "normal-like" distribution between 0-128.

The results is illustrated by the figure 4.2.



(a) original normal distribution (b) amplified distribution

Figure 4.2: Amplification of the distribution

4.2.4 Rotation

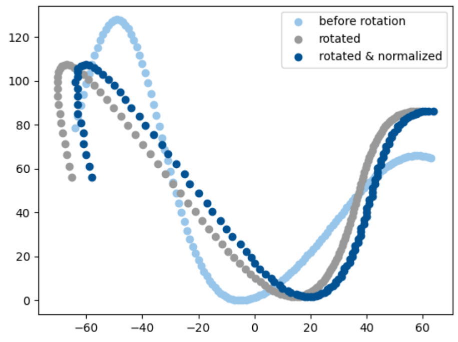
A possible extension of the distribution is to rotate the distribution with certain angles. This is the rotation of the distribution itself, to better mimicking the defect, which is different from a direct rotation-augmentation that can be used when loading training data.

The direct rotation augmentation of the final image would be taken simply as a useful technique to improve the robustness of the model, same as other augmentation techniques like flipping, cropping and translational shifting, while the rotation here is trying to improve the technique of the mimicking the defect in reality.

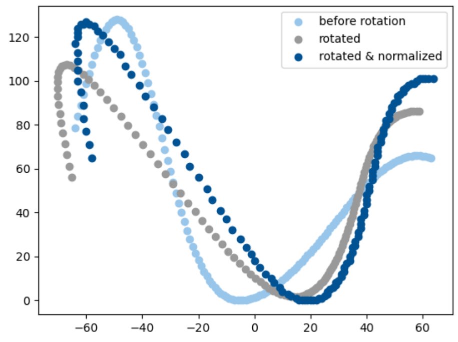
The rotation uses simply the rotation matrix for x and y axis. And the rotation center is set as the center of the sampled image (0,64), as after amplification the distribution is ranged from 0 to 128 along y-axis. The rotation degree is set to 20 degree.

To restrict the rotated image still within the 128x128 image frame. A normalization and integerization is applied on the rotated image. The x-axis is normalized to (-64,64), while the y-axis is normalized to (0,128)

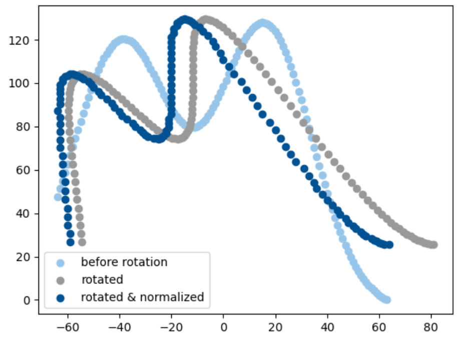
There are different ways to do the normalization. As the spline is mimicked to defect through the image, the x-axis normalization would be always be applied to the rotated distribution. And the y-axis can also be normalized, or just be kept after being rotated. As illustrated in the figure 4.3.



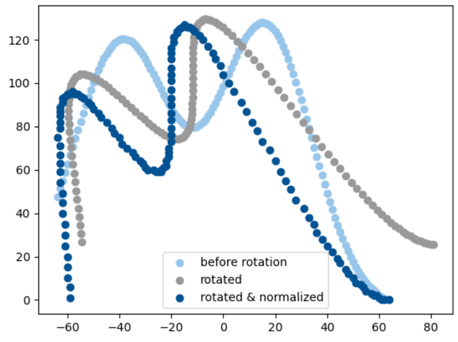
(a) rotated and x-normalized(1)



(b) rotated and xy-normalized(1)



(c) rotated and x-normalized(2)



(d) rotated and xy-normalized(2)

Figure 4.3: Rotation and normalization of the distribution

4.2.5 Normalization/Integerization

In this step of the image creation, the whole plot is normalized into a specific range of the picture. As the final image is created into a 128x128 image, the the distribution is normalized into the range of (21,106), so the plot is centered in the image, occupying 2/3 of the whole image. To achieve this, a min-max normalization is used to project the distribution to specific range.

Furthermore, to create the image with pixel wise value, the value of the normalization is also integerized. So each sampled value would be represented to the corresponding pixel in the image.

In this part, the normalization is just trying to make the upper and lower bound of the spline, making the defect centered in the image, which can be easily changed and augmented for further research.

Noticeably, the normalization and integerization should be aligned with the rotation part. The y-axis normalization can be applied after the rotation, which would make the final result always the case of xy-normalized(the cases of 4.3(b) and 4.3(d)), even if a x-normalized is used in the rotation part(the cases of 4.3(a) and 4.3(c)). To maintain the feature of x-normalization, rotation must be applied after the normalization to the image frame, and another x-axis normalization must be applied, as illustrated in the figure 4.4.

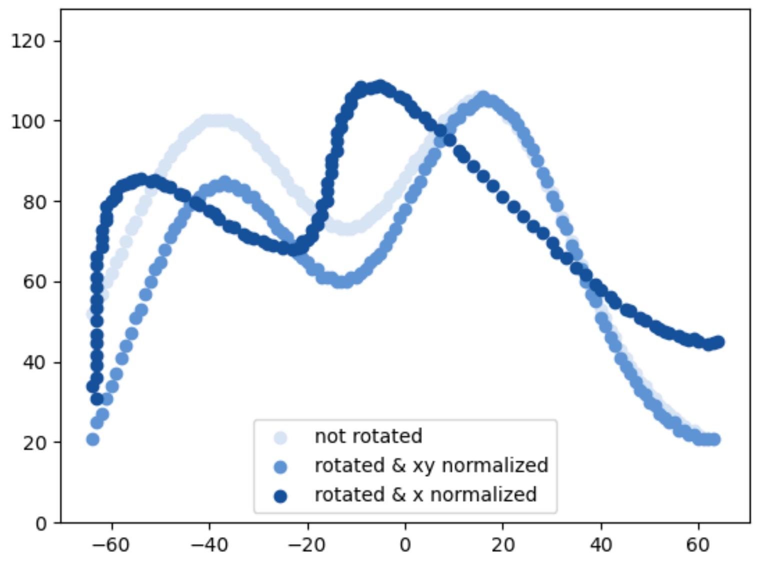


Figure 4.4: Distribution after normalization w/o rotation

4.2.6 Image Matrix

After the normalization, the distribution would be plotted centered in an 128x128 image. A pixel-wise matrix would be created to plot the image.

As the plot is now two one-dimensional value of the x and y. This image matrix creating procedure tries to create the two dimensional matrix by pixel-wise value through the whole 128x128 image.

As the result of samping and integerization, there would be possible breakpoints on the plot. To fill the information between the breakpoints, when sampling along the x-axis, the occurring break space would be filled by giving values of the neighbouring sample, connecting the breakpoints and creating a plot. As illustrated in the figure 4.5, the variable d refers to the distribution value that would be plotted into a matrix.

In this way, the image matrix would be created and the pixels are connected with each other in the plot. The image then is created using this pixel-wise matrix illustrated in figure 4.7(a).

```
def create_Matrix(d):
    gMatrix = np.zeros([128,128])
    for i in range(128):
        if i <127:
            d1 = min(d[i],d[i+1])
            d2 = max(d[i],d[i+1])
            for x in range(d1,min(d2,127)+1): gMatrix[i][x]=1
        else: gMatrix[i][d[i]]=1
    return gMatrix
```

Figure 4.5: source code of matrix creation

However, after rotation of the plot, the x-axis sampling is not as uniformly distributed as before rotation. Therefore, the matrix creating procedure would be more complicated. The spline should be draw pixel by pixel and the space between the breakpoints should be filled in another way, illustrated in 4.6. The missing samples between the pixels are filled by a whole rectangle, i.e. all pixels among the range of the two pixels would be filled with information, i.e. set value as 1.

```python
def create_Matrix(rx,de):
    gMatrix = np.zeros([128,128])
    rx = rx+64 #change x-axis from (-64,64) to (0,128)
    for i in range(128):
        x = rx[i]
        y = de[i]
        if i < 127:
            gMatrix[x][y]=1
            #next pixel with value
            x1 = rx[i+1]
            y1 = de[i+1]

            #find out the matrix of the missing data between two pixels
            if x <= x1: X = np.array(range(x,min(x1,127)+1))
            else: X = np.array(range(min(x,127),x1-1,-1))
            if y <= y1: Y = np.array(range(y,min(y1,127)+1))
            else: Y = np.array(range(min(y,127),y1-1,-1))

            #set the missing information value as 1
            for p in X:
                for q in Y:
                    gMatrix[p][q]=1
        else: gMatrix[x][y]=1
    return gMatrix
```

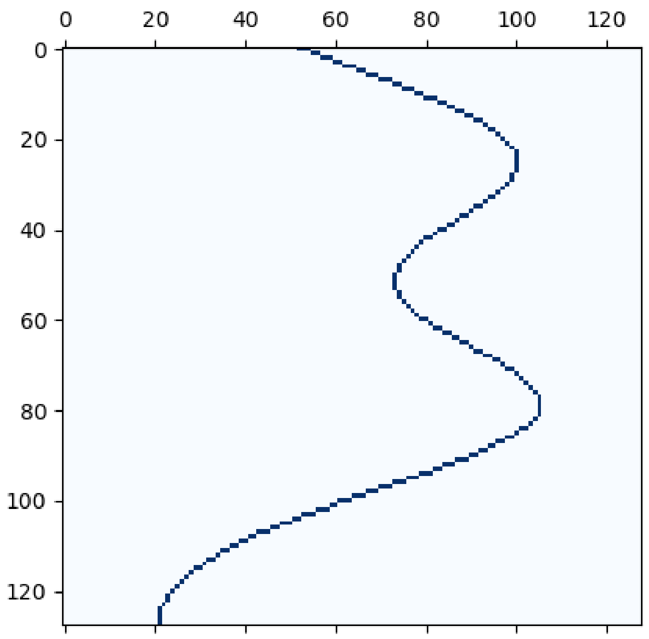
Figure 4.6: source code of matrix creation for rotated image

4.2.7 Extensions

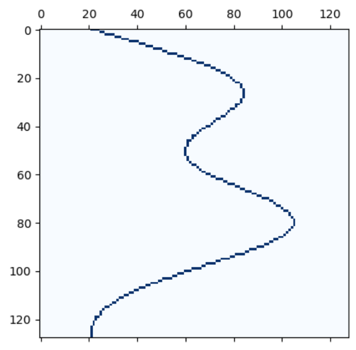
It is also possible to add on some other extensions in the data generating process, as the image is created pixel by pixel. It is easier to extend other augmentations for further research.

Originally, the database would be intended to create with a 5-normal-distribution-based spline, combining 5 randomly weighted normal distributions, as illustrated in figure 4.9 without randomly weighted coefficient, and 4.10 with random weights.

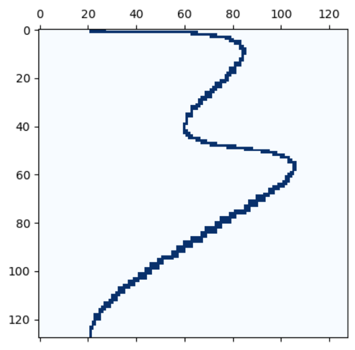
With the random sampled mean and variance of each normal distribution and a random coefficient weight, the image database would have a 15 DOF. The spline is



(a) without rotation



(b) rotated and xy-normalized



(c) rotated and x-normalized

Figure 4.7: pixel-wise matrix

better mimicked, similar to the defects in reality.

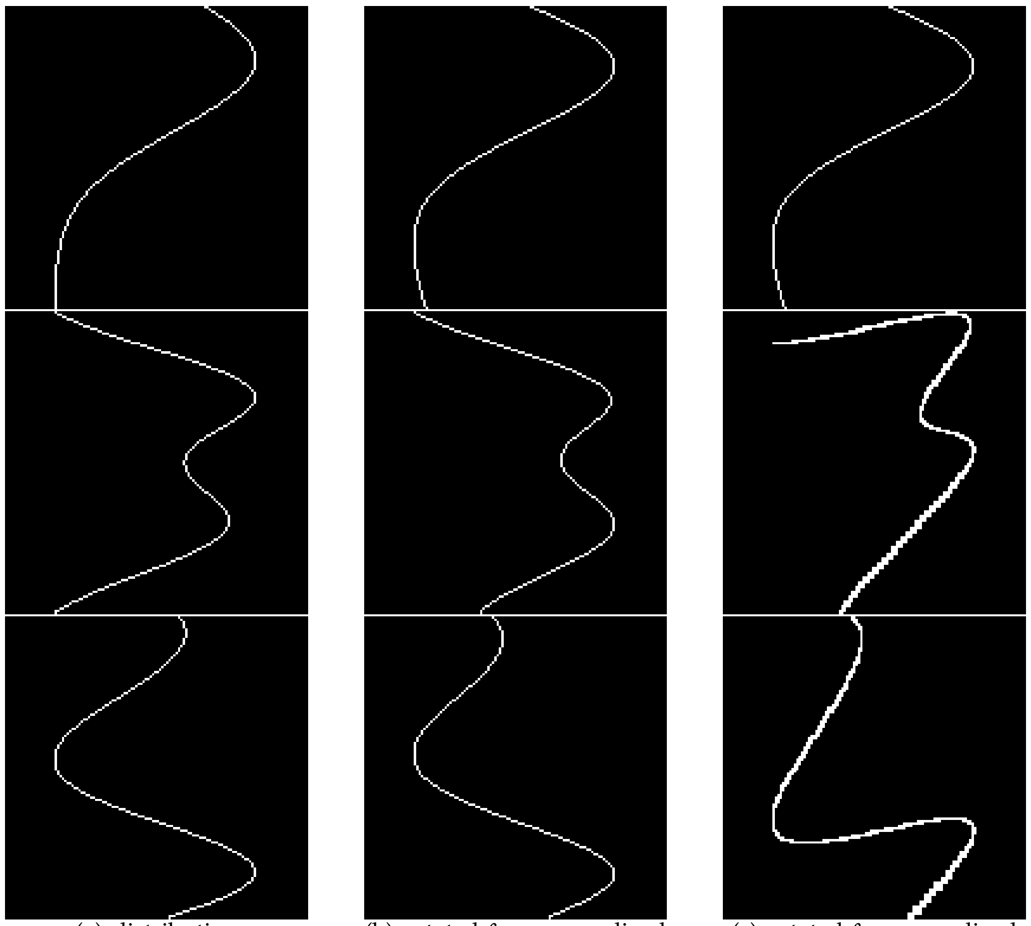

(a) distribution (b) rotated & xy normalized (c) rotated & x normalized

Figure 4.8: 2-norm distribution images

However, as the experiment would have no intense samples of each degree, it would be difficult for the model to learn about the characteristics. Therefore, the 15-DOF spline is only used for test sets, to test the generalization ability of the model.

Furthermore, as explained in the 4.2.4 part, it is possible to add on rotation matrix before the normalization, which would better model the possible defects in reality. However, this would also increase the degrees of freedom. To avoid the problems that would be caused because of sparse sampling, which in turn causing confusion of information. The rotated images, illustrated as in figures 4.8, 4.9,4.10 are also only used in the test sets to evaluate the generalization ability.

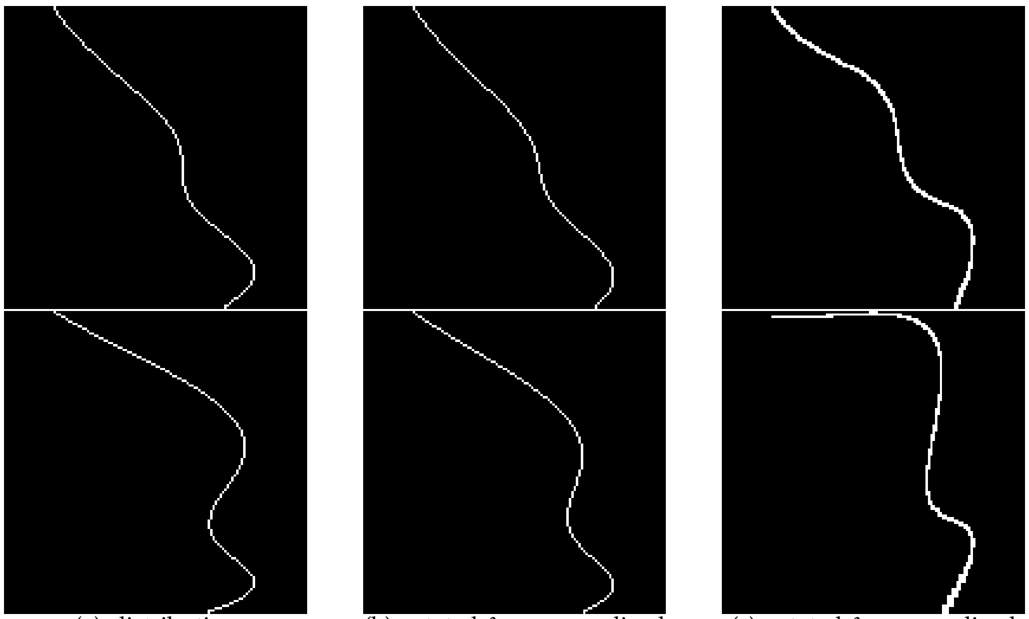

(a) distribution (b) rotated & xy normalized (c) rotated & x normalized

Figure 4.9: 5 norm distribution images

4.2.8 Discussion

Due to the limit of computing capacity and storage, it is not possible to use an intense sample along all degrees of freedom. Therefore, the whole database is always faced with the trade-off between furious information that would improve the generalization ability of the training and sparse sampling without clear representations of any characteristics that would confuse the model.

Therefore, the different datasets, i.e. samples from different data distributions, are allocated into different experiments settings.

4.3 Datasets allocation

After the creation of the data according to the data structure. The data would also be allocated into different datasets to fit with the machine learning requirements.

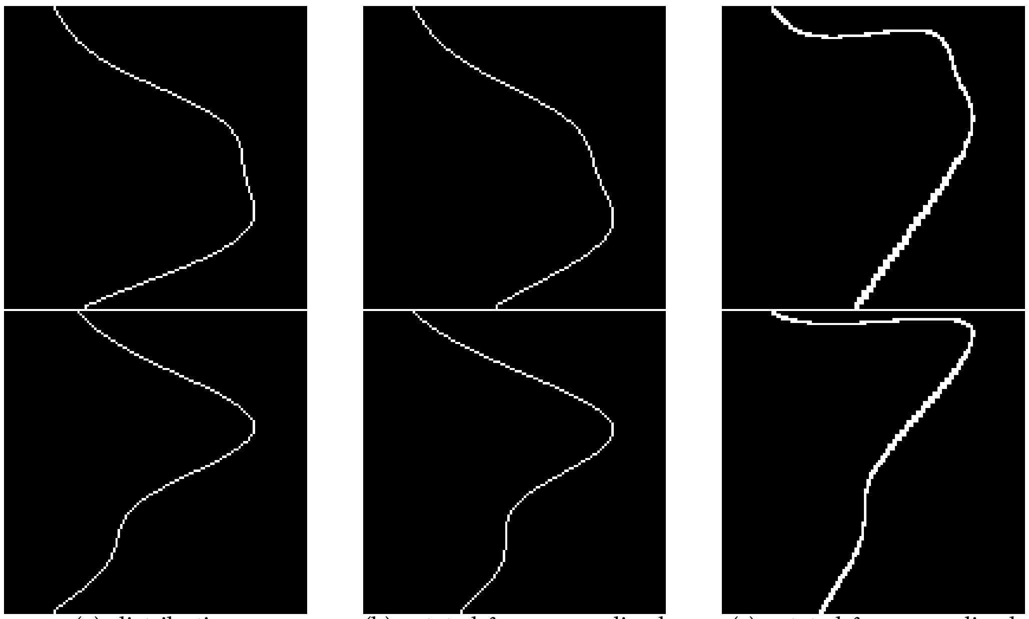

(a) distribution (b) rotated & xy normalized (c) rotated & x normalized

Figure 4.10: randomly weighted 5 norm distribution images

For the 2 normal distribution based images, a intense sampling of the mean and variance of each normal distribution is created. The sample space is therefore structured with 16*16*16*16 images. For the training process, all data would be used, a random selected 32 images are used for validation. The test datasets is combined with random selected 32 images from the training set and 32 images from the 5-norm distribution based dataset and rotated images.

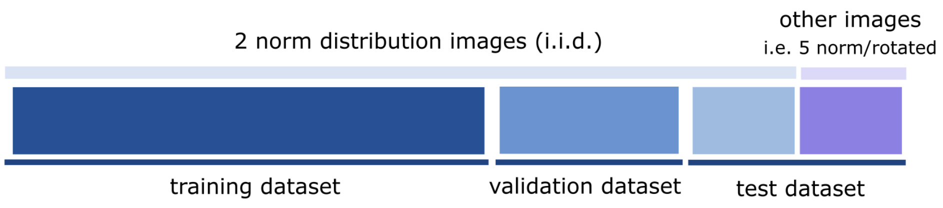


Figure 4.11: Dataset structure illustration

4.4 Conclusion

The data creation part, explained in detail how the datasets used in the experiment have been created and allocated. And the possible further extensions on the datasets would depend on the change of the hyper-parameters of the data structure, which would lead to further possible research, and more thorough study of the training features of the data.

The hyper parameters of the datasets include:

distribution relevant:

- numbers of normal distribution basis, a larger number of distribution would lead to more variance in the image, creating the spline closer to the reality
- random coefficient, whether the distributions are randomly weighted or not, a random weighted distribution would avoid the impact of extreme samples and the corresponding breakpoints due to limited sample capacity
- the sample interval of mean and variance, from which range of mean and variance should the distributions be sampled from, this would directly affect the result of the spline
- amplifier constant, this amplifier is used to prepare for rotation, but would also dinimish the accuracy of the sample, making the sampled result sparse
- rotation degree, how much degree should be rotated, this can also be used to the better mimic the reality.

image creating relevant:

- information range, the spline should be plotted in which part of the image, this would control the information intensity in the image
- information filling strategy, when the break points occur due to sampling, which strategy should be used for connecting the breakpoints and add on information

For further data creation, changes to the hyper-parameters could be considered. On one hand, the hyper parameters that would add on DOF, would lead to difficulties in training, therefore, should be taken more considerations, and the training procedure and model structures should be modified accordingly. While the change of other hyper-parameters that has no relation with the DOF, would lead to different mimicking scenarios, and the relation between these hyper-parameters and the training performance can be further studied.

Table 4.1: The hyper-parameter table.

hyper-parameters	case in experiment	add on DOF
numbers of distribution	2/5	Y
random coefficient	without/with	Y
the sample interval of mean and variance	(16,32)	N
amplifier constant	128	N
rotation degree	20	N
information range	2/3 of image, pixel 21-106	N
information filling strategy	neighbouring pixels	N

5 experiment

5.1 training structure

I based my training process on the model structure illustrated from the paper adversial autoencoder. The model provides the essential GAN framework for the training.
The idea is to create a GAN-trained generator and discriminator with the neural network model, and use the discriminator as the regularizer, testing the performance of the regularizer.

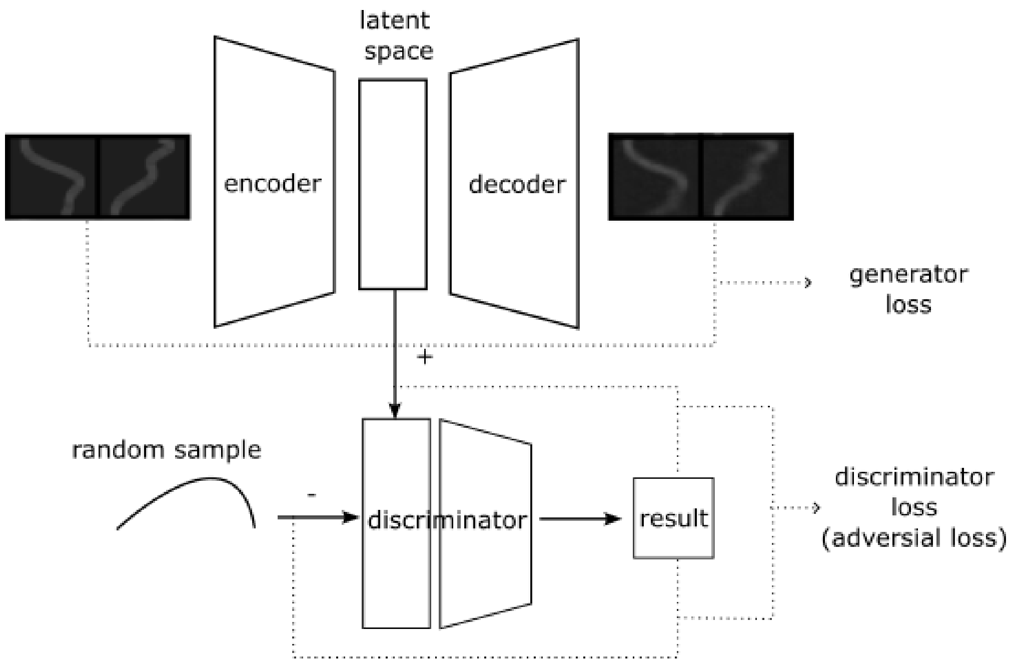


Figure 5.1: training structure

The generator (variational autoencoder) and the discriminator would be trained at the same time, the neural networks would be updated using different loss functions. The two neural networks would be trained against each other to improve performance of each other. The performance of the model is evaluated from both of the loss functions.

5.2 model details

This section the detailed information about the entire network would be explained. The entire training model includes a variational autoencoder and a discriminator. There are three neural networks in the whole model. The encoder, decoder and the discriminator. The model would train both the discriminator and the variational autoencoder, setting the discriminator and the variational autoencoder as the adversarial counterparts in the model.
The performance of the model would be evaluated both from the performance of the variational autoencoder and the performance of the discriminator.

5.2.1 Convolution neural network

The model given by the existing paper, used a fully connected neural network. The training in the original model was trained on MNIST, and the generator reached relatively good result. Due to the similarity (single line based) between the MNIST and our datasets, I first tried on the linear model(illustrated in figure 5.2) for training. However, the result was not able to converge. It is probably due to the sparse sampling, i.e. the samples are not representative enough for the linear model to learn so many weights. Therefore, I finally chose the convolution neural network to do the training.

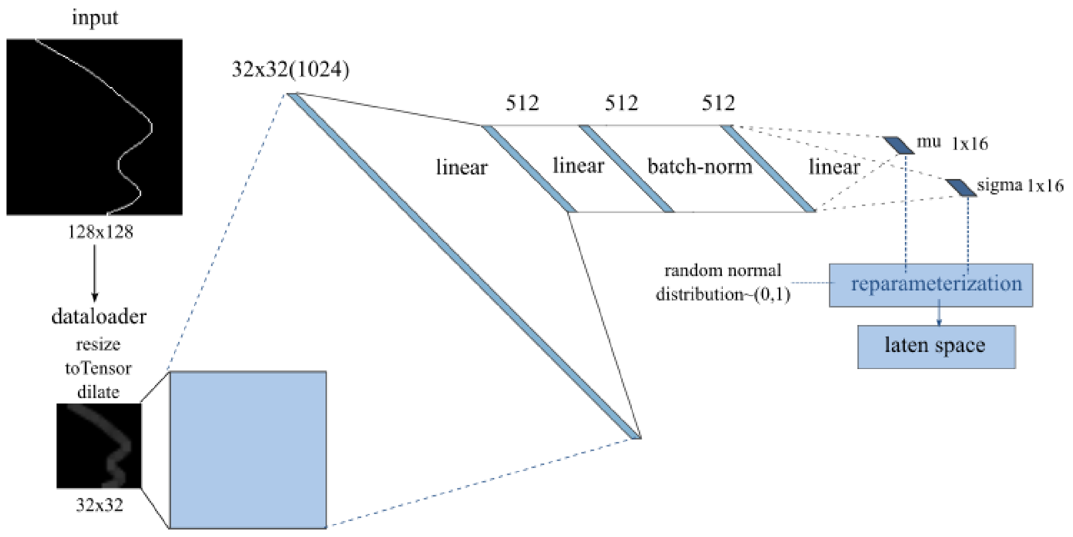


Figure 5.2: linear neural network structure

The CNN was trained step by step from native autoencoder, to variational autoencoder and finally to the adversial autoencoder. The model structure's ability to train a converging neural network was proven by the AE and VAE. Therefore, the model structure for autoencoder part is not changed, only a discriminator instead of a simple BCE loss function is added when training.

5.2.2 Encoder

The structure of the encoder is clearly illustrated in the figure 5.3. There are three convolution layers for the image processing. After each convolution layer, a ReLu() function is followed for non-linearity. The filters are all 4x4 convolution filters. In the first two layers, not much information is reduced, stride was set as 1 and channels increases from 1 over 8 to 16. The third layer reduced the weights dimension, with a stride of 2, decreasing the image size to 12*12 and a decrease in channel number to 8. After the convolution layer, channels are fully flattened into a 8*144 linear combination and then decreases over 144 to latent space dimension 16, after two layers of fully connected neural networks.
For the variational reparameterization, the mean and variance for the normal distribution is separately modeled.
Then, a reparameterization is used to built up the relation between the weights and the samples. So, it would be possible to do the back-propagation for the weights updating. As illustrated in figure 5.3

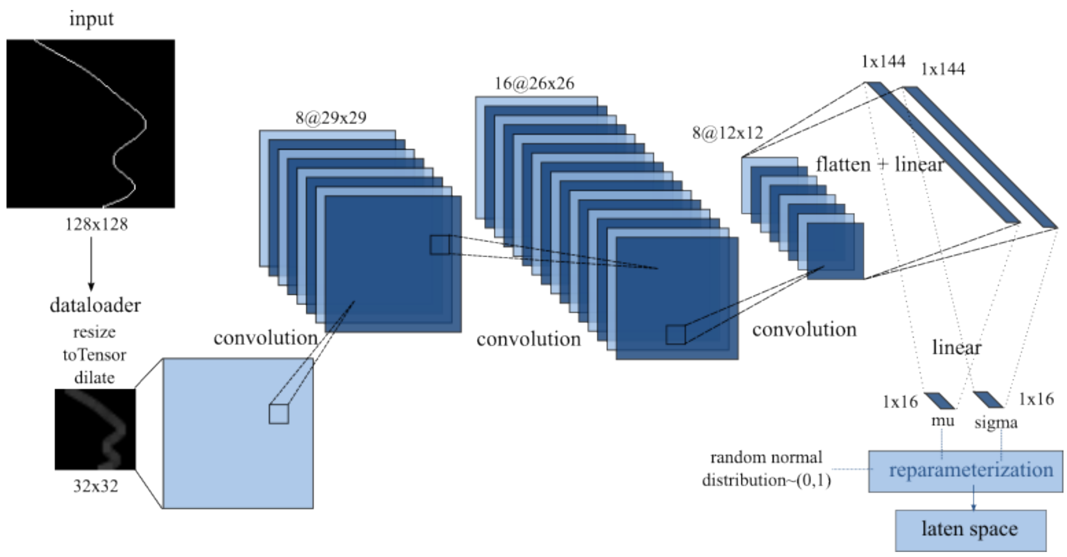


Figure 5.3: encoder neural network structure

5.2.3 Decoder

The decoder here used the normal form of decode in VAE, using the opposite layers of the encoder, as illustrated in figure 5.4.
For the decoder, fist layer is the linear layer, increasing the latent space into 144 neurons, with activation function LeakyReLu(0.2). A further linear layer with the same activation function would prepare the input out as 8*144 neurons, prepared for the unflatten into images. After being unflattened into 8 channels of 12x12 images, three layers of Convolution Transpose would be applied, with non-linearity formed by ReLU(). All filters are 4x4 and the channels after increased to 16 would remain 8 before the final output. In the last layer, the images would be transposed to a 1 channel 32x32 image, which is the final output of the procedure.

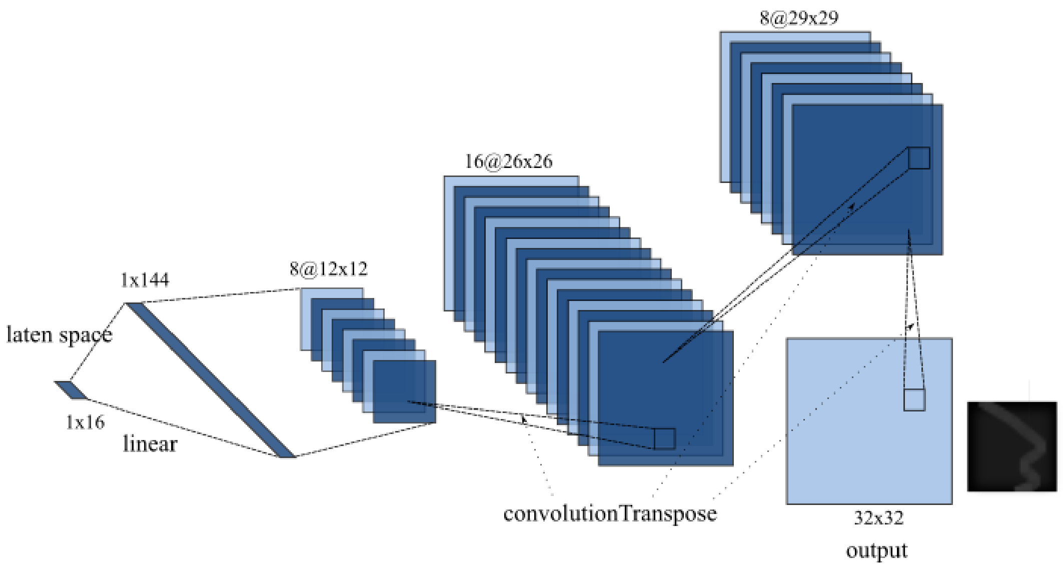


Figure 5.4: decoder neural network structure

5.2.4 Discriminator

As illustrated in the training structure figure 5.1, the discriminator has the input from the latent space and a random normal distribution sample. Therefore, the input are already linear vectors. Therefore, the discriminator uses a fully connected linear neural networks model, as illustrated in figure 5.5.
From the latent space with an input of a vector of 16 neurons, the first layer increases the information into 512 neurons, and second layer decreases to 256 neurons, both activated by LeakyReLU(0.2). The final layer give the output as to one layer and activated by

Sigmoid(), having a useful distirbution between (0,1) which is aligned with the need of discriminator.
The input of the discriminator is either the latent space result from the encoder, or the random sample from a normal distribution. The discriminator is trained to distinguish the difference between the two. The detailed mechanism of the discriminator would be mentioned in the loss function part.

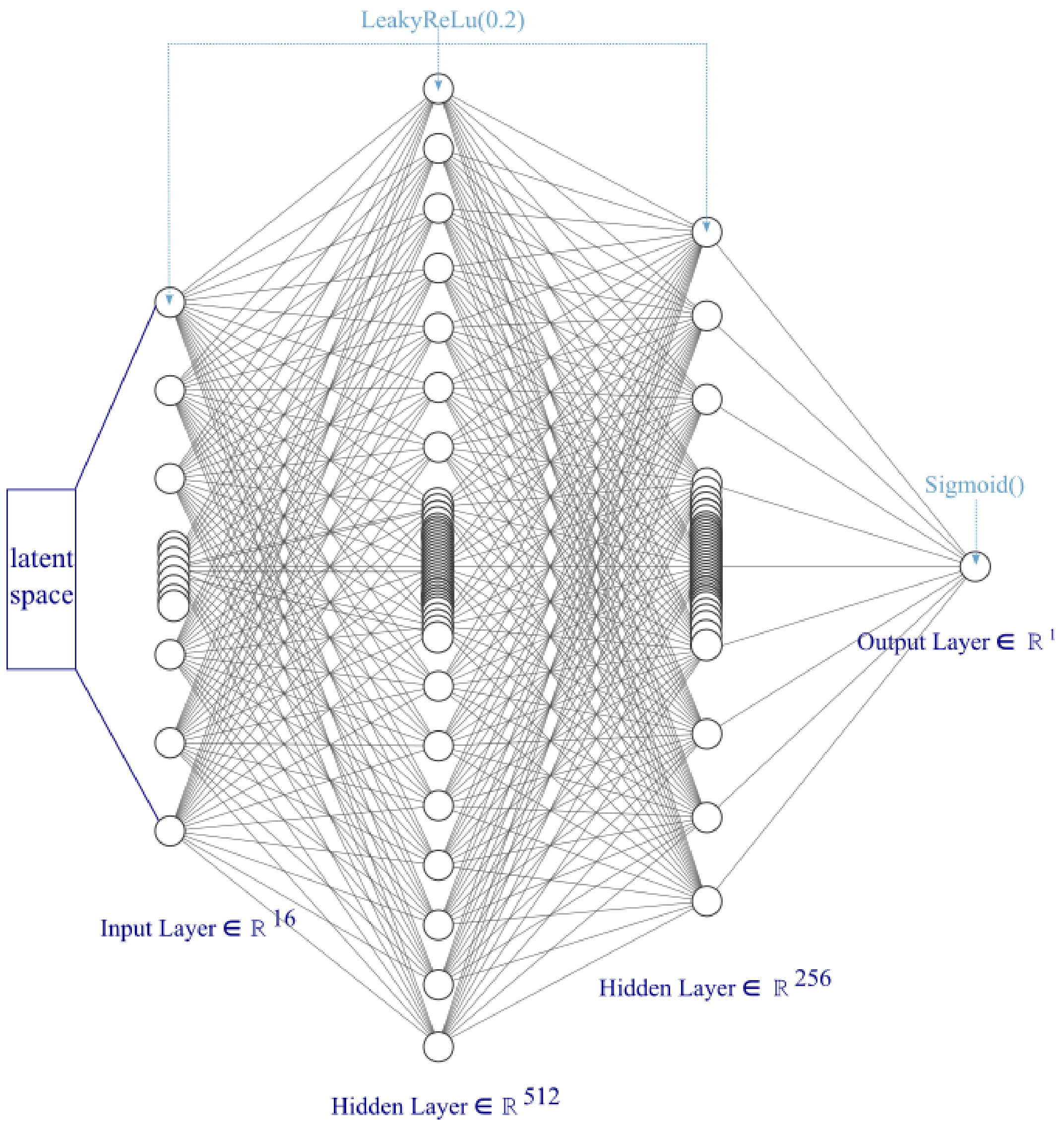


Figure 5.5: discriminator neural network structure

5.2.5 Loss functions

There are two loss functions used in the GAN trained model. The generator and the discriminator have different loss functions. The basic GAN training mechanism is as follow,

$$\min_{G} \max_{D} E_{x \sim p_{data}}[logD(x)] + E_{z \sim p(z)}[log(1 - D(G(z))]$$

The generator G and the discriminator D can be found using alternating SGD in two stages: (a) Train the discriminator to distinguish the true samples from the fake samples generated by the generator. (b) Train the generator so as to fool the discriminator with its generated samples.
Both, the adversarial network and the autoencoder are trained jointly with SGD in two phases – the reconstruction phase and the regularization phase – executed on each mini-batch. In the reconstruction phase, the autoencoder updates the encoder and the decoder to minimize the reconstruction error of the inputs. In the regularization phase, the adversarial network first updates its discriminative network

The generator loss is made up two parts:

$$g : loss = 0.001 * BCELoss(discriminator(encoded : \\ imgs), 1) + 0.999 * MSELoss(decoded : imgs, real : imgs)$$

BCELoss(): binary cross entropy loss of the discriminator recognize the encoded picture as real;
MSELoss(): the mean square errores between the generated image and the real image.
where BCELoss(discriminator(encoded-imgs), valid) is the adversarial loss part;

The discriminator loss is made as an average of the binary cross entropy:

$$real : loss = BCELoss(discriminator(z), 1)$$
$$fake : loss = BCELoss(discriminator(encoded : imgs, 0)$$
$$d : loss = 0.5 * (real : loss + fake : loss)$$

real loss: BCE loss of the discriminator recognize the fake image as real
fake loss: BCE loss of the discriminator recognize the real image as fake

5.3 training details

This section would explained the training procedure of the model. And specific trials and results.

5.3.1 overfitting

Due to the limited capacity of training and computation time, it is reasonable to train and first see the result of the model trained on an overfitting dataset.
The overfitting here, means with a limited number of samples, small variation in the dataset, I trained intentionally train the model to be overfitted, to give a primitive check on the result of the model and the training.
Therefore, the overfitting dataset is made up of 32 randomly selected images from the whole dataset, illustrated in figure 5.6.

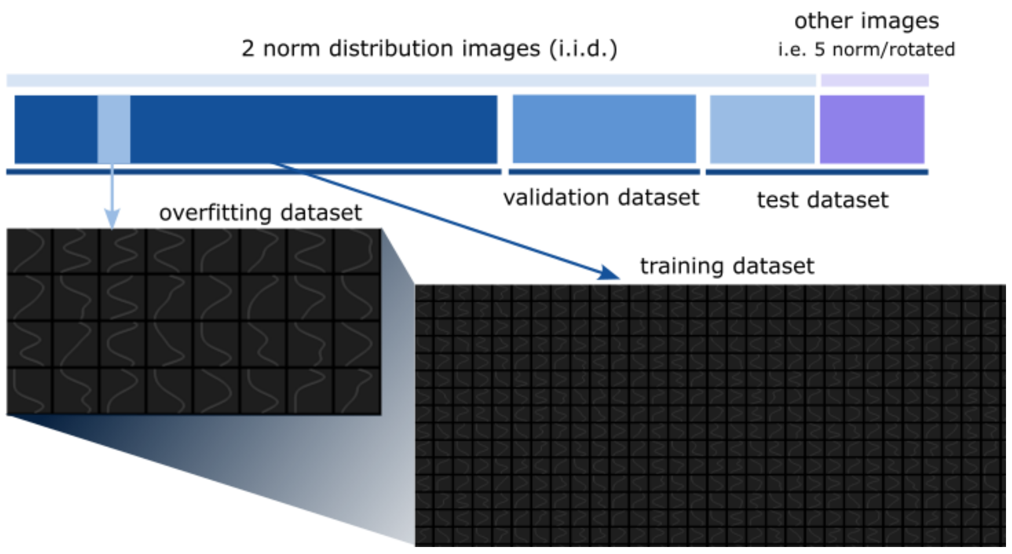


Figure 5.6: overfitting dataset

The model would be able to generate a relative good result in overfitting. The overfitting training procedure also gives a brief forecast of how many epochs may be necessary for the training on the entire dataset, which batch-size should be chosen and how possible it is to generate a good result, helping to estimate the training procedure on the whole dataset.

5.3.2 information density

Since the spline is only on pixel information. When training directly with the dataset, the information is not enough, this would lead to difficulty in convergence, as illustrated in figure 5.7. The second row was the pre-trained result after 200 epochs. It is clear for the images with less dense of information it is difficult to converge. Therefore, the

input are loaded in the dataloader directly with a resize to 32x32 and a dilation of 3 times.
This resize and dilation could also be possibly absorbed by the convolution neural network. But to reduce the computation cost, and make the network more interpretative, I used the dataloader as a small pre-processing.

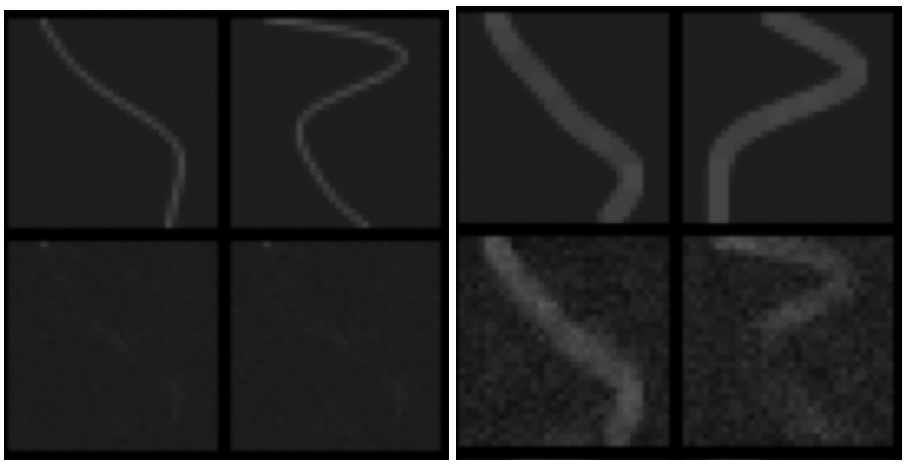

(a) without dilation (b) with 3 time dilation

Figure 5.7: with/out dilation in overfitting

5.3.3 DOF

The original training set used 5 randomly weighted normal distribution as training set, which made it difficult to converge. After changed to 2 normal distribution based dataset, the model is easier to reach convergence. As explained in data creation part. High DOF requires also more samples to make the features representable.

5.3.4 latent space dimension

The latent space is also impacting the training same as the DOF. It is necessary, not to extract the features too much. For MNIST dataset, the latent space dimension is 10. In the case of our dataset, I use 16 as the latent space, so that not too much information is lost during the feature extraction.
The latent space is also directly related to the model structure. Though, it is possible

to change only in the linear layer, but that would make not many difference when the model is not converging.

5.3.5 batchsize

With the help of the overfitting training, it is easy to see and find a good result at the very beginning of the dataset. Therefore, it is also possible to try different batch-sizes in advance, which would make the result in later training more feasible.
Through the overfitting period, it is easy to see, that the batchsize would lead to different converge speed. Bigger batchsize is time consuming as there are fewer iterators in each epoch, while smaller batchsize makes the gradient more unstable, the stochastic probability is larger. The overfitting period can give a hint of whether larger or smaller batchsize should be used.

5.3.6 loss function

Different loss functions may have different results in the training. For the case of our experiment, there are possiblity of BCELoss(), L1Loss(), MSELoss(). They would give out different training results and different training times.
This hyperparameter is not possible to selected with only overfitting, as through overfitting, no clear difference is observable. The proper way to select this hyperparameter is to train it from the AE, VAE, when the model is relatively easy to converge, choose the proper loss function for the generator, which would also improve training efficiency.

6 Evaluation

This chapter is about the evalutation methods of the regularizer. The evalution methods for the regularizer is usually connected to the inverse problem, but in the experiement, training regularizer together with the whole inverse problem is not realistic. Therefore, we propose a method that would be independent from the inverse problem. Whether it would help improve the result in the inverse problem is remain to be researched, but it can be used as a primitive evaluation of the result.

Ideally, a regularizer would be able to distinguish between the normal images and the images with strong noises. Therefore, the inversion results with very strong noises would get punished and in turn be trained to get a result less noised.

Ideally, a regularizer would be able to distinguish between the normal images and the images with strong noises. Therefore, the inversion results with very strong noises would get punished and in turn be trained to get a result less noised.

However, it is not possible to directly illustrate and model out what kind of "noised" result the inversion part would calculate out. And the "authentic defects" are also only pseudo defects modeled in the way illustrated in the chapter 4.

Therefore, we propose to add image noises on the images in the dataset, to test the performance of the discriminator. The way used in the experiment to test the performance of the regularizier at distinguishing between authentic and counterfactual images would be the test between original images in datasets and images with manually added noise.

Additionally, the images generated by the variational-autoencoder, can also be seen as a source of counter-facts. However, after the generator has learned enough information, it would be difficult to see great difference between the "counter fact" and the "pseudo facts". The generated images must be manually selected to reach the requirements.

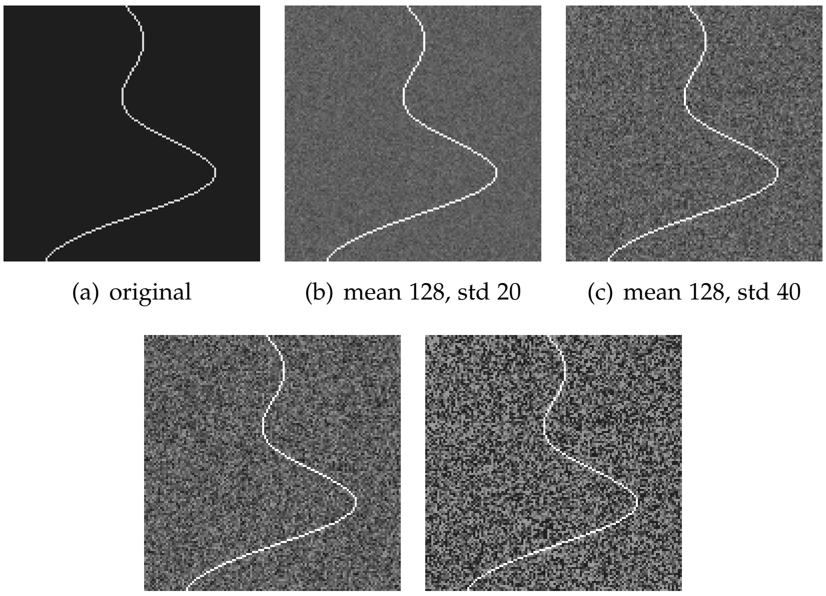
The following sections in this chapter would give examples of the "counterfactual" images used in the experiments, mainly the noised images and the generated images.

6.1 noised images

And there are different kinds of possible image noises included. These noise are usually image based, rather than defect based, therefore, the outperform of distiguish the images noise would indicate possibilities of ruling out factors that are not knowledge-based, and may have little to do with the defect mechanism, which would be another possible advantage.

6.1.1 Gaussian noise

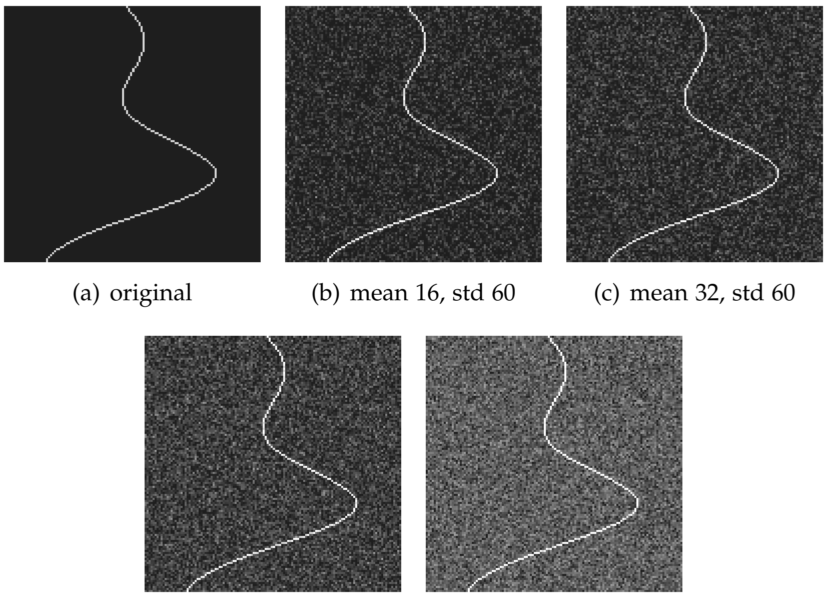
The Gaussian noise is added independent at each pixel, and independent of the signal intensity, caused primarily by Johnson–Nyquist noise (thermal noise), including that which comes from the reset noise of capacitors ("kTC noise"). [Oht17] And the knowledge based behind the adding of the Gaussian noise, according to the literature, is that it is a major part of the "read noise" of an image sensor, that is, of the constant noise level in dark areas of the image[Nak17], which is exactly the case we are facing for the dataset created in data creation set.



(a) original (b) mean 128, std 20 (c) mean 128, std 40

(d) mean 128, std 60 (e) mean 128, std 120

Figure 6.1: Gaussian noised images with increased std



(a) original (b) mean 16, std 60 (c) mean 32, std 60

(d) mean 64, std 60 (e) mean 128, std 60

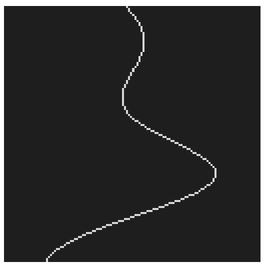
Figure 6.2: Gaussian noised images with increased mean

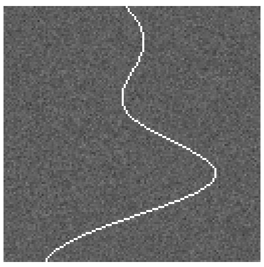
It is the most easily applied and useful image noise, as it is easy to control the mean and variance of the noise given to pixel. It is possible to have a serial of noised the images with different mean and standard deviation of the Gaussian noise (illustrated in figure 6.1 and figure 6.2) or add on Gaussian noise on to the images for several times((illustrated in figure 6.3)), and see the performance of the regularizer.

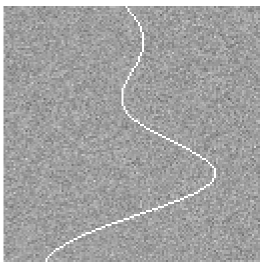
6.1.2 Salt-and-pepper noise

The other possible noised added is the salt-and pepper noise. Fat-tail distributed or "impulsive" noise is sometimes called salt-and-pepper noise or spike noise. An image containing salt-and-pepper noise will have dark pixels in bright regions and bright pixels in dark regions.
As our images created from the dataset are mainly dark parts, the threshold to create the salt-and-pepper noise should be concentrated from the bright side, close to 255, the code to add on salt-and-pepper noise is illustrated in figure 6.4. The different threshold would lead to different levels of noise, displayed in figure 6.5

The additive result would seem similar to different thresholds if applied with several times of uniform noise, illustrated in figure 6.6


(a) original


(b) 1x Gaussion noise


(c) 2x Gaussion noise

(d) 3x Gaussion noise

(e) 4x Gaussion noise

Figure 6.3: additive Gaussian noised images

6.1.3 Quantization noise(uniform noise)

The noise caused by quantizing the pixels of a sensed image to a number of discrete levels is known as quantization noise. It has an approximately uniform distribution. Therefore, a uniform sampled noise is added to the image. However, for uniform distribution, there would be no feature on itself, only added on noise as illustrated in figure 6.7

6.2 generated images

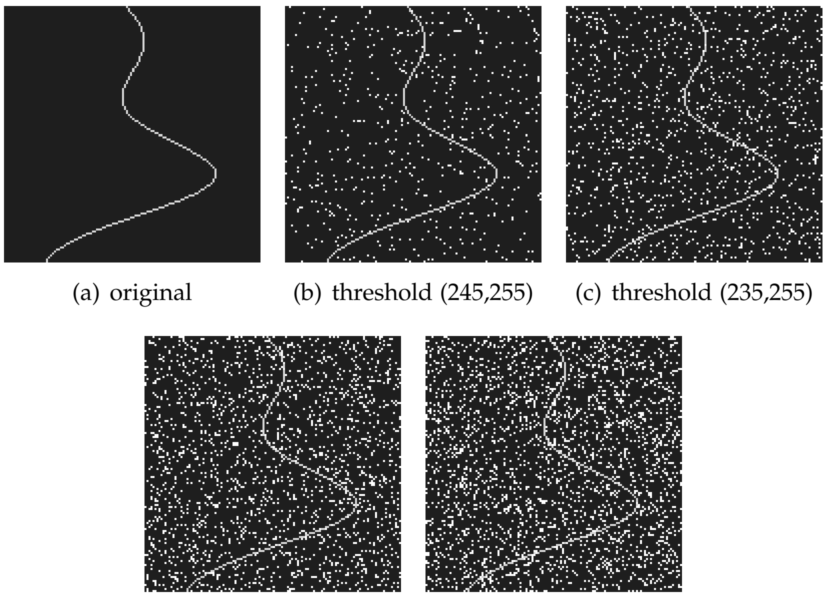
Poorly generated images from the autoencoder would also be possible solutions for the evaluation, illustrated in figure 6.8 The detailed information regarding the autoencoder is explained in chapter 5 and the results of the autoencoder would be displayed in chaper 7.

```
for filename in namelist:
    img = Image.open(filename)
    imp_noise=np.zeros(img.shape,dtype=np.uint8)

    # use cv2 package to create uniform sample
    cv2.randu(imp_noise,0,255)

    # set binary threshold for the sampling to get salt-pepper noise
    imp_noise=cv2.threshold(imp_noise,245,255,cv2.THRESH_BINARY)[1]
    img_in = cv2.add(img,imp_noise)
```

Figure 6.4: Code of adding salt and pepper noise



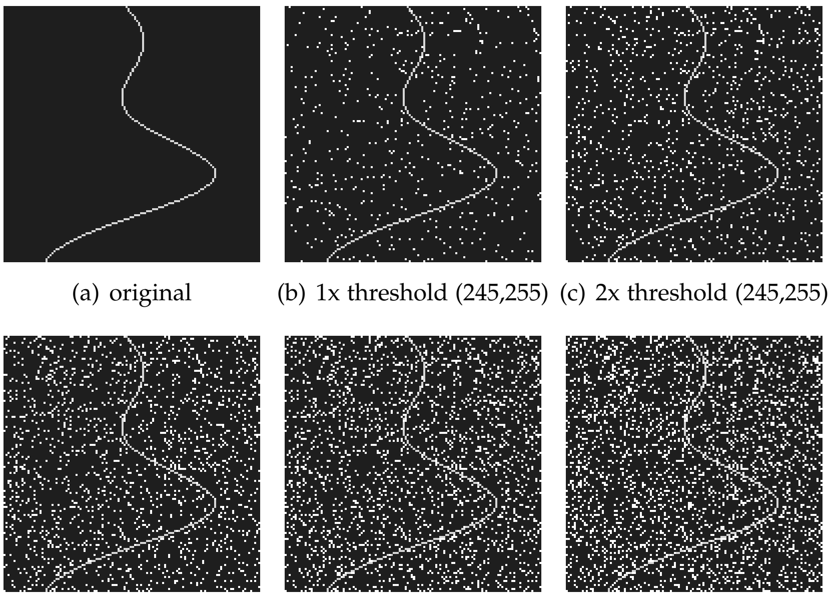
(a) original (b) threshold (245,255) (c) threshold (235,255)

(d) threshold (225,255) (e) threshold (215,255)

Figure 6.5: salt and pepper noised images with different threshold

6.3 evaluation model

The evaluation of the regularizer would be using the encoder of the autoencoder and the discriminator instead of the decoder. The structue is illustated in the figure 6.9. The evaluation need the encoder to make the pictures into the latent space and get the discriminator working from there. The whole encoder and discriminator combined

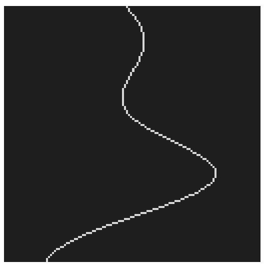


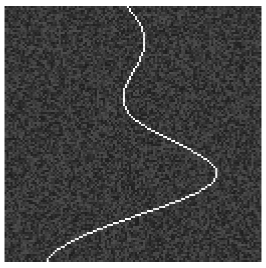
(a) original (b) 1x threshold (245,255) (c) 2x threshold (245,255)

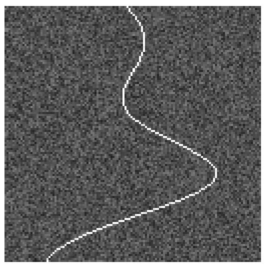
(d) 3x threshold (245,255) (e) 4x threshold (245,255) (f) 5x threshold (245,255)

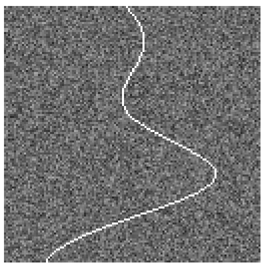
Figure 6.6: additive salt and pepper noised images

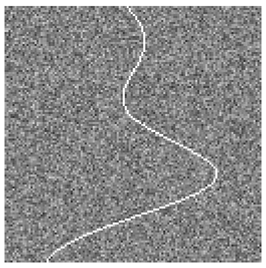
form the regularizier, as shown in the figure.
In this way, it would be possible to evaluate the performance of the regularizer. The detailed performance result would be displayed in chaper 7


(a) original


(b) 1x threshold (245,255)


(c) 2x threshold (245,255)


(d) 3x threshold (245,255)


(e) 4x threshold (245,255)

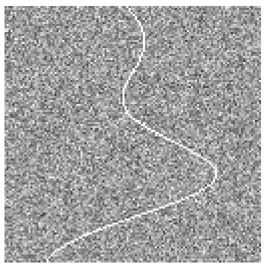
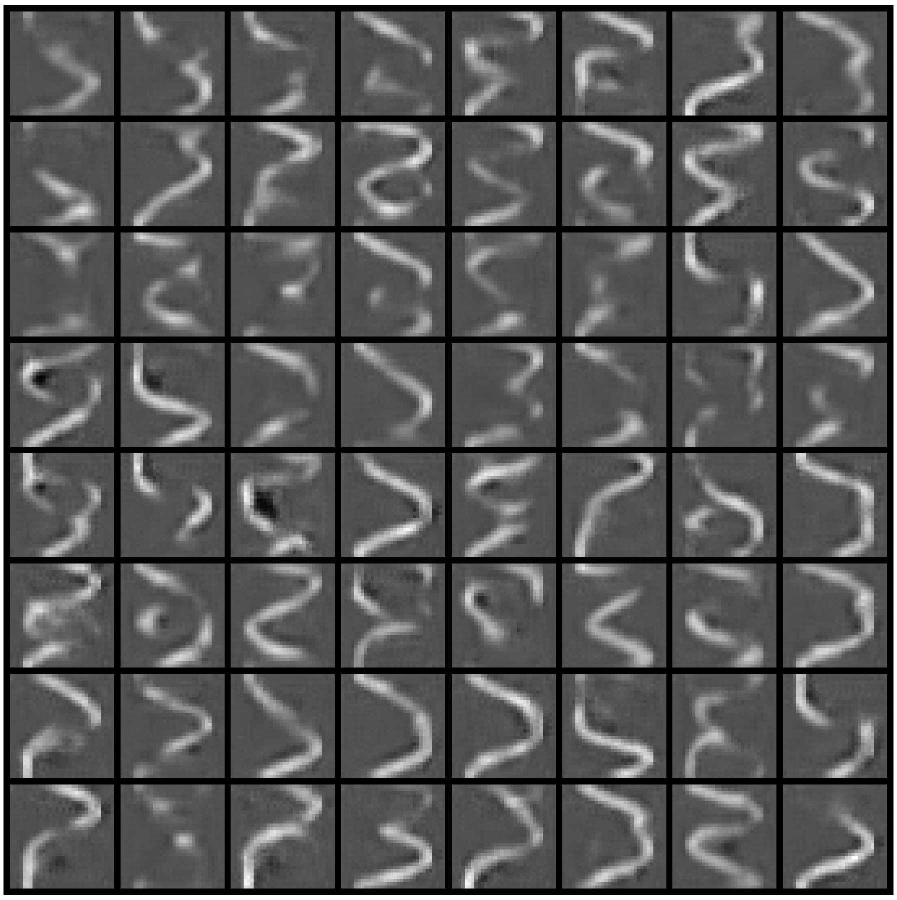
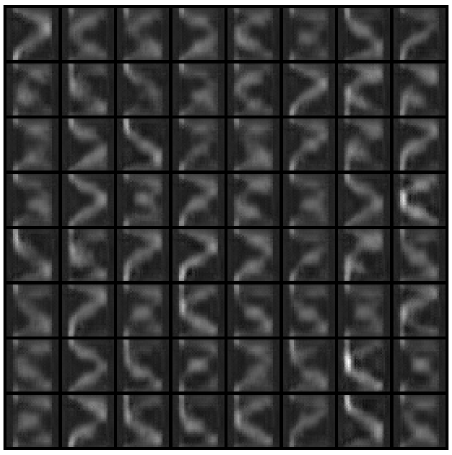

(f) 5x threshold (245,255)

Figure 6.7: additive uniform noised images


(a) generator after 13 epoch


(b) generator after 12 epoch

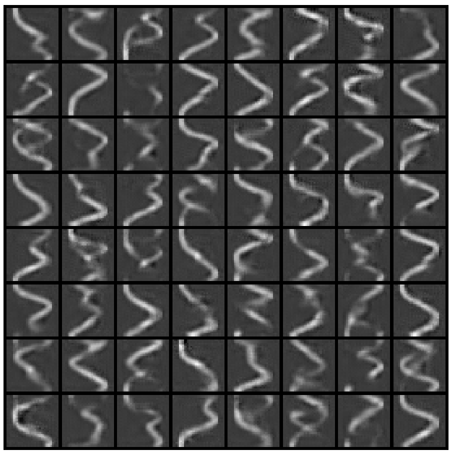

(c) generator after 15 epoch

Figure 6.8: generated images

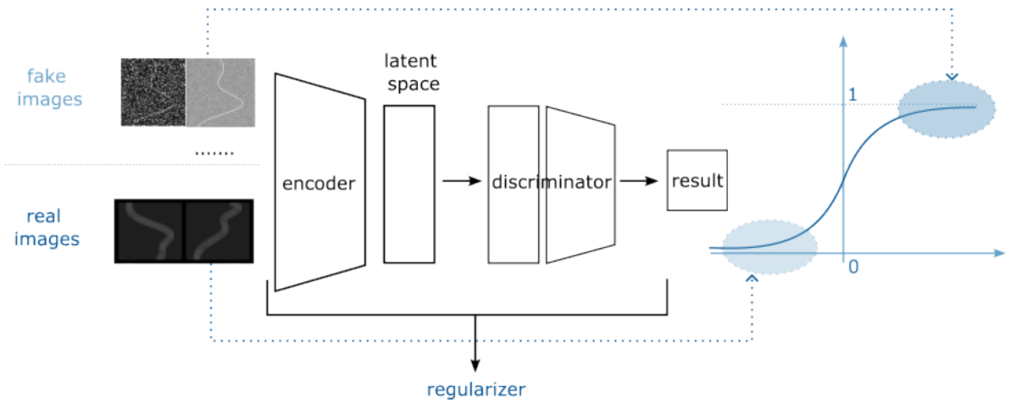


Figure 6.9: evaluation structure of the regularizer

7 results

In this chapter, I will illustrate the results of the experiments carried out during the project.
The section would be divided into training results, generator results and discriminator results. The training results would show the results from the training process, the loss function plot. The generator results mainly shows the performance of the result trained by the generator, how good the GAN trained VAE is performing. The discriminator result shows the performance of the discriminator, how good it can rate between real images and noised images.
Several experiments have been made before reaching the final results, using different batch size, different loss functions, therefore, first the main results would be illustrated. And then, a compasion would also be made between different experiments.

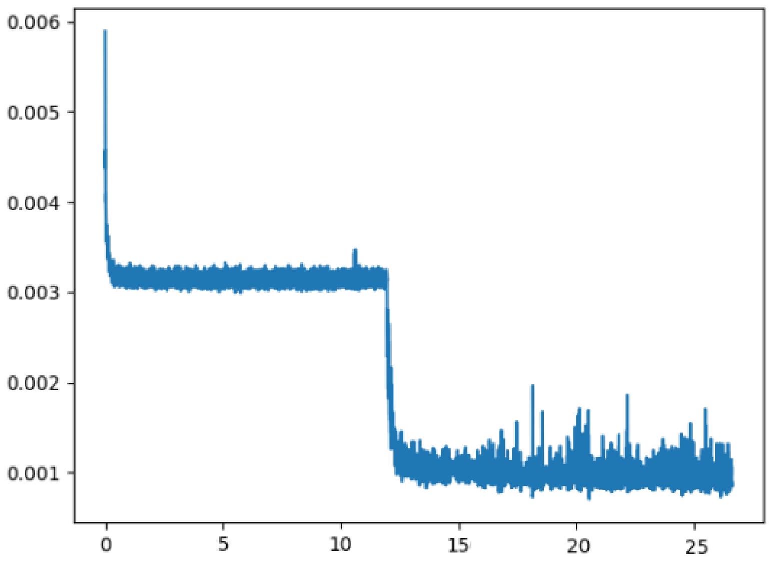


Figure 7.1: training loss plot

7.1 training results

This section shows the change of the loss function during training. It is clear to see, that the training loss has two stages of decreasing, illustrated in figure 7.1.
The first stage of decrease happens in the very beginning of the training, the loss value is reduced to a certain degree. Then, after several epochs, the loss function remain stable (till after 12 epochs), but the performance of the autoencoder is not good, illustrated in figure 7.2. Here, in the output results, the original images are shown in a row, and the output of the autoencoder is shown in the row below. It is clearly to see, during the staged-period of the loss value, the performance of the autoencoder remains poor, comparing to the performance after the second loss value decrease, illustrated in figure 7.3.

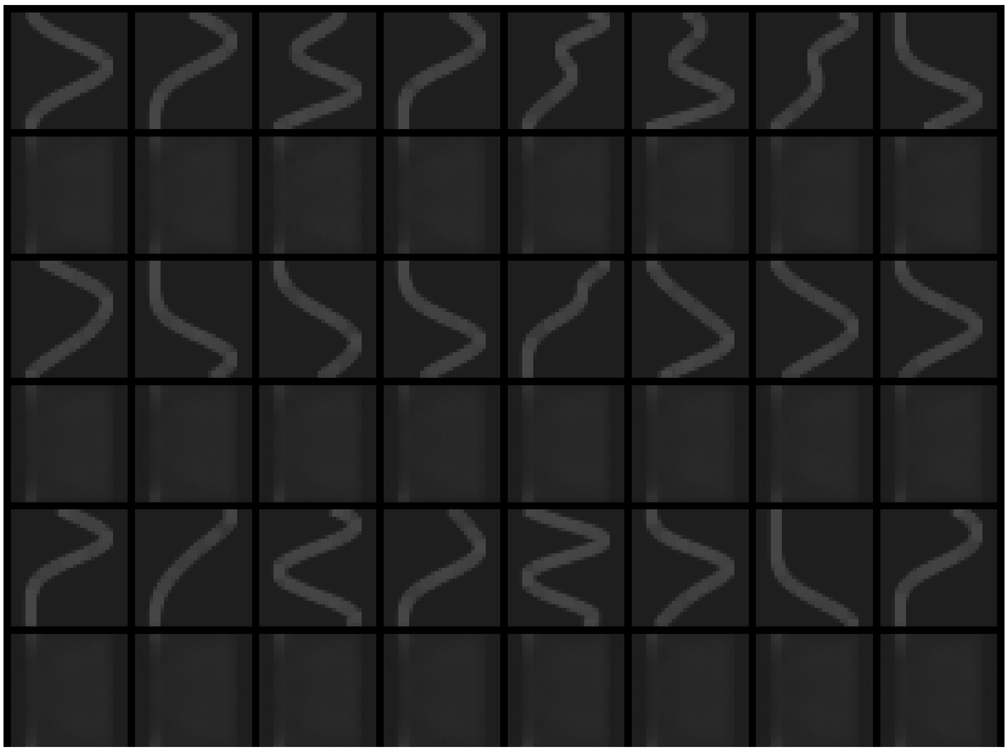


Figure 7.2: AAE output result after 6 epochs

The critical decrease of loss value happens in the 12th epochs. It is shown in the figure 7.4. The training result is getting much better and resembles more and more the original input in this epoch.

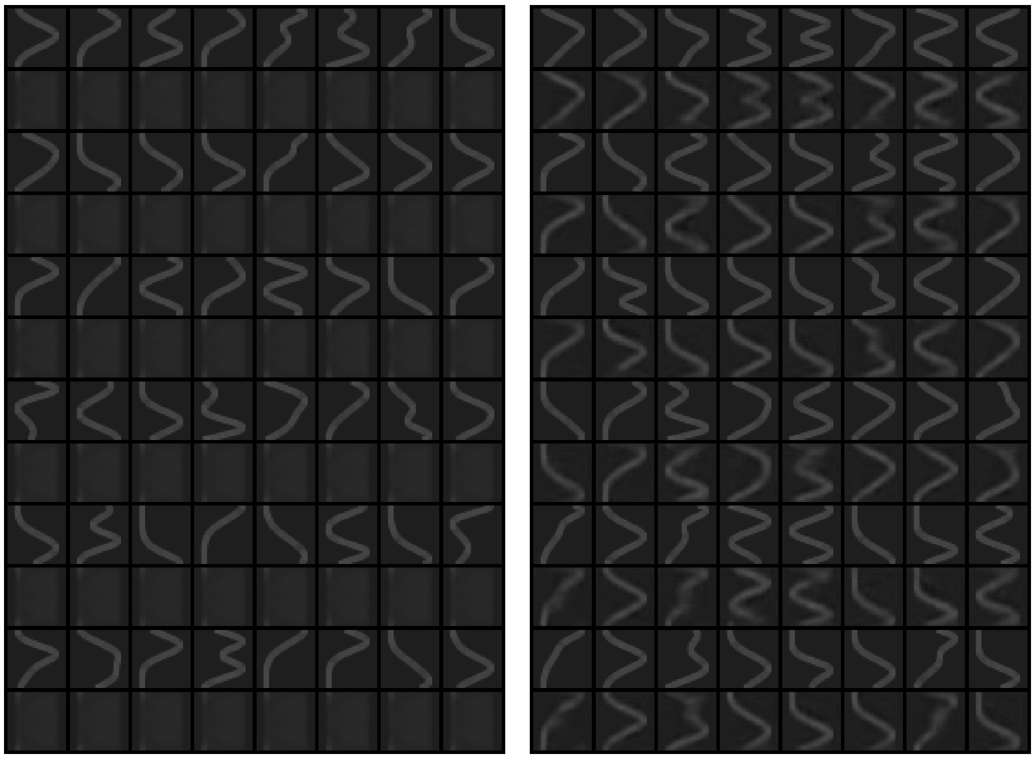
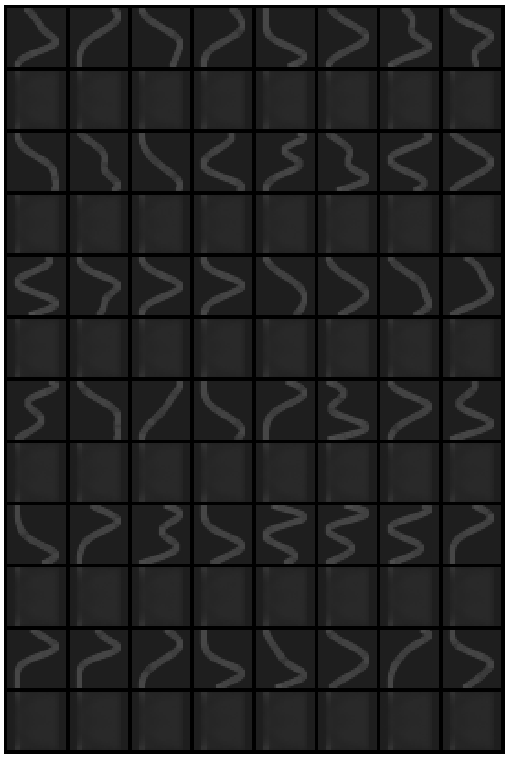
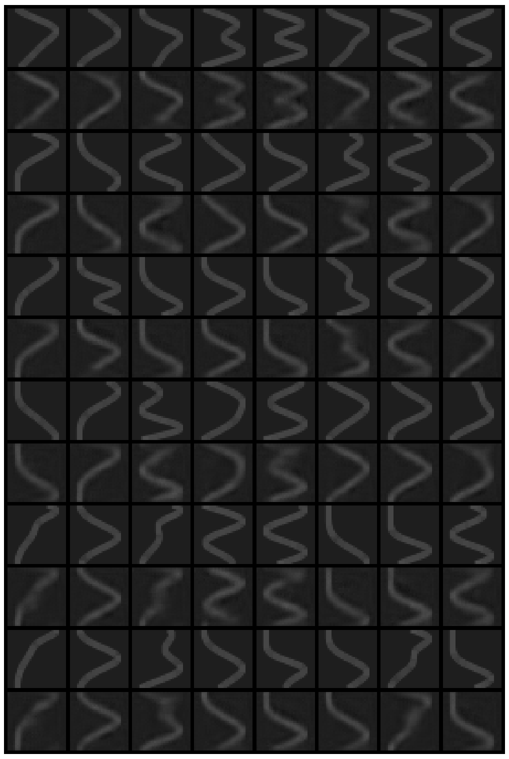

(a) training after 6 epoch (b) training after 12 epoch

Figure 7.3: AAE output result after 6 and 12 epochs

Meanwhile, from the results in other figures i.e. figure 7.5 and figure 7.6, it is possible to see, after the training loss value has decreased to certain degree, the validation output also performance a satifying result, with limited performance weakness comparing to the training output, shown in figure 7.6. This indicates, that the generator has reached a relatively good performance level (illustrated in detail in the next section 7.2) when the loss value has reached its second decrease stage.

Additionally, the decrease in loss value is also highly relevant with the performance of the discriminator, illustrated in figure 7.7.
The first row is the output of auto-encoder on training data. The second row of plots is the generator loss value plot, the third row of plots is the discriminator loss value plot, i.e. the adversarial loss function value plot.


(a) training after 11 epoch


(b) training after 12 epoch

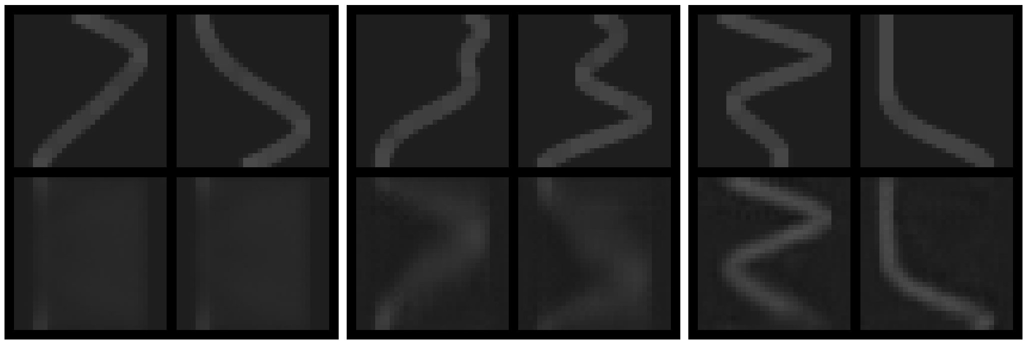
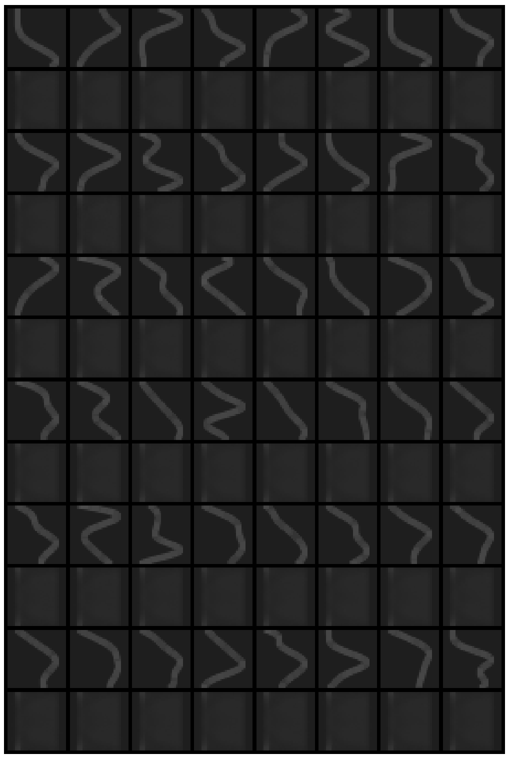
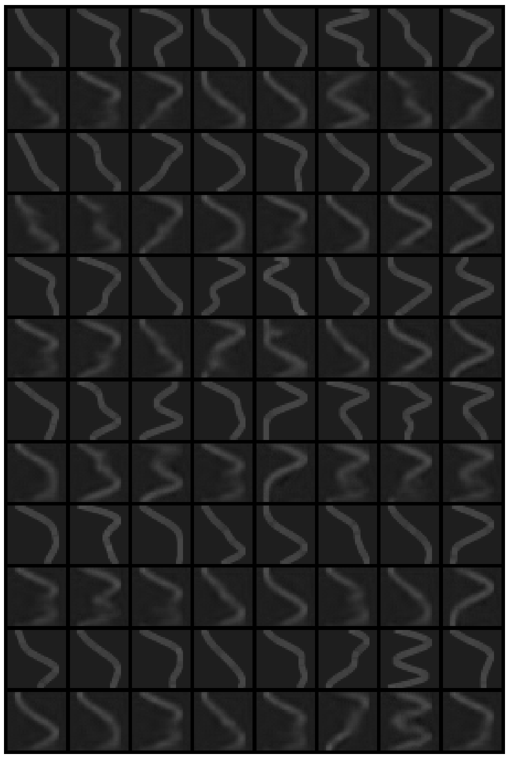

(c) training output in between

Figure 7.4: AAE output result after 11 and 12 epochs

It can be observed, after the generator loss value reaches its second stage, the value and variances of the discriminator increases, which is clearly illustrated in the figure 7.8. Furthermore, it is not observable that the discriminator loss has a tendency of


(a) validation after 11 epoch


(b) validation after 12 epoch

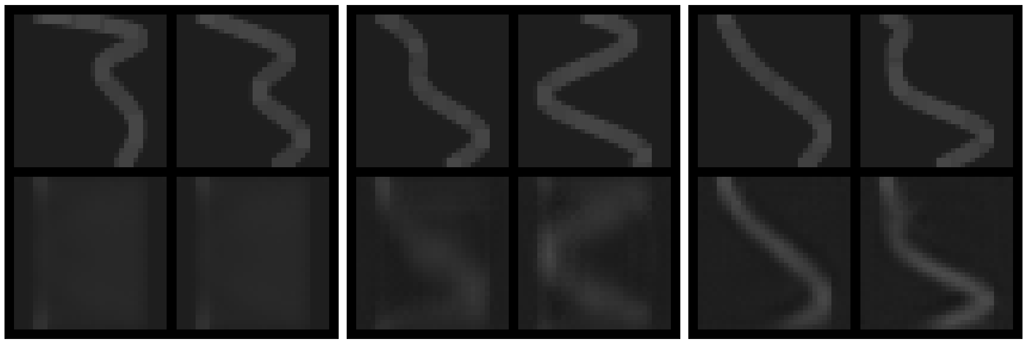
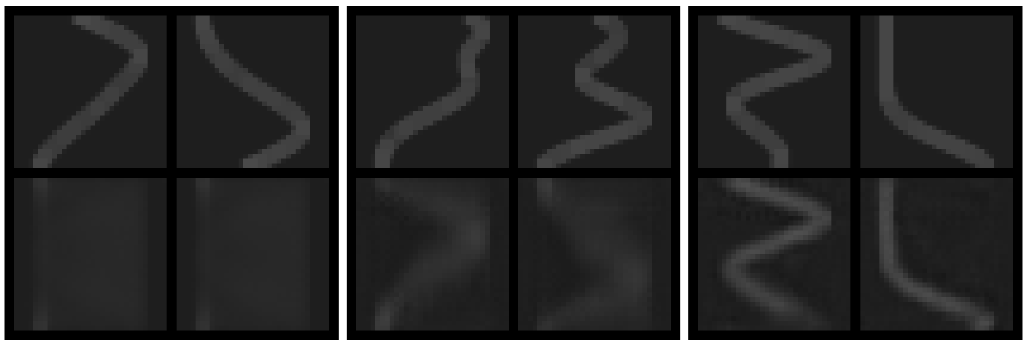

(c) validation output in between

Figure 7.5: AAE output result after 11 and 12 epochs

convergence. And the evaluation of the discriminator would be illustrated in detail in section 7.3


(a) training output between 12th epoch

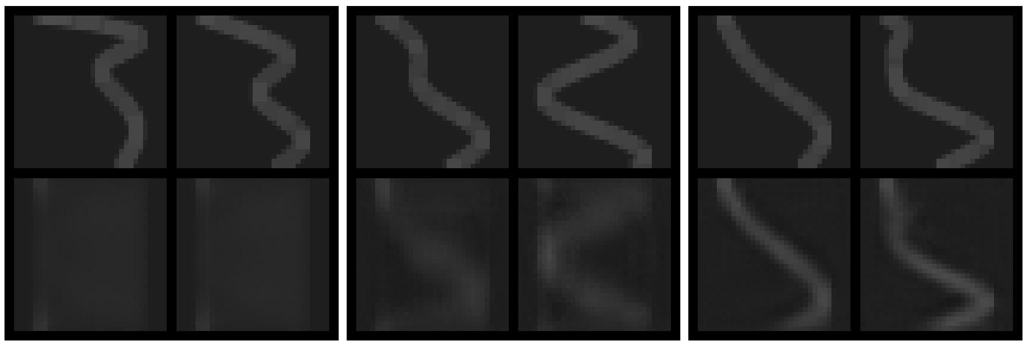
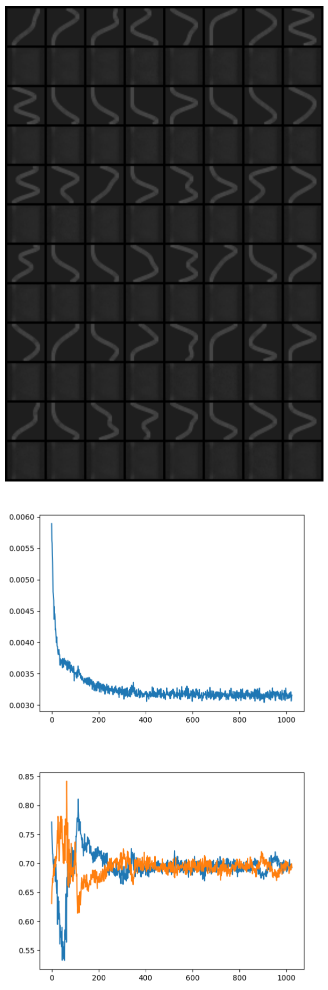

(b) validation output between 12th epoch

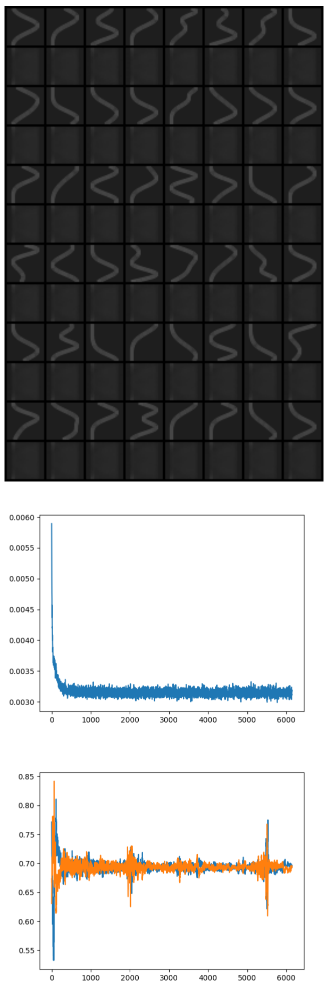
Figure 7.6: Comparison between training and validation output

7.2 generator results

In this section the result of the generator in the experiment would be displayed. The generator is the part of the decoder part of the autoencoder. Although, it is not directly correlated with the regularizer trained. It is possible to use the generated images as the counterfactual images to train the discriminator to understand better of the true images.

The plots shown below (figure 7.9 and figure 7.10) are the series of output from the decoder, when a random normal distribution is sampled in the latent space and fed into the decoder.
The shown images are the group of output results in the critical epochs 11th, 12th, 13th, when the generator loss value reached its stage after second decrease. It can be seen that the improvement of generator performance occurred later than the second decrease stage, illustrated in figure 7.9. Due to the variational autoencoder requirements, the


(a) training after 1 epoch


(b) training after 6 epoch

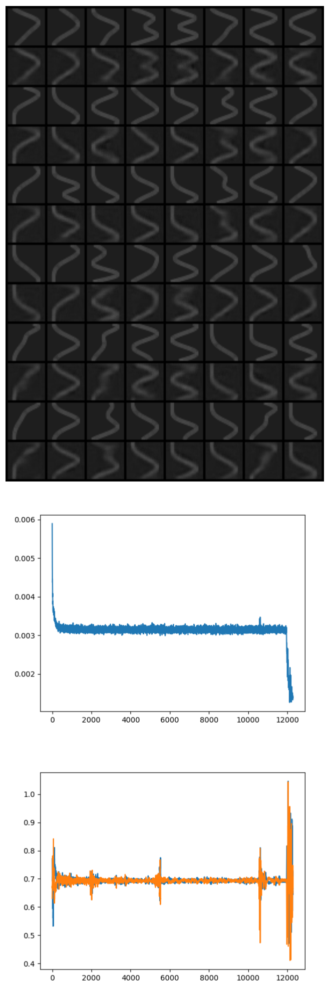
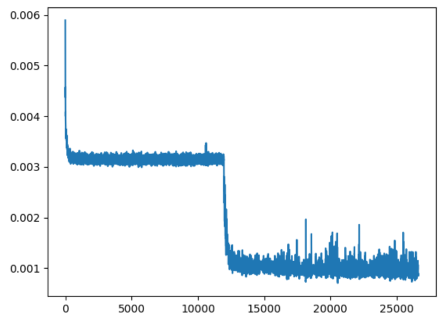
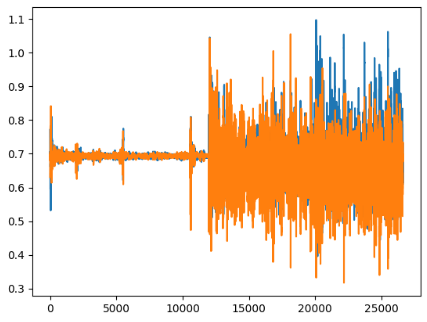

(c) training after 12 epoch

Figure 7.7: loss function and generator result

training for the training for the variational part is more time consuming.
The better results occurred in the around 25 to 50 epochs, as shown in the figure7.10. The generated results are already really close to the created datasets. In this way, the decoder can be used as a easy dataset-generator for further study, as generator for basic facts, or counterfactual "fake" or "noised" images, according to the research setting.



(a) generator loss after 25 epoch



(b) discriminator loss after 25 epoch

Figure 7.8: AAE output result after 6 and 12 epochs

7.3 discriminator results

In this section, the performance of the discriminator would be displayed. As the discriminator is the regularizer, the performance of the discriminator would be evaluated by performing on the original data and the noised image dataset as illustrated in the section 6.3.
The evaluation would be divided into three parts. A general frequency table of the results, indicating the distribution of the score given by the regularizer.
The average score plot with different degrees of noise in different noise type, this would trying to figure out, whether the regularizer is sensitive to the complexity of the noise.
The average score plot of the generated images, this part the performance of the regularizer would be evaluated on the generated dataset, checking whether the regularizer is possible to distinguish between the real images and the generated images.

7.3.1 general result

This figure 7.12 shows the frequency table of the score. It can be seen the blue results are the real images, and the others are noised images. The distribution of the scores are relative separable. The mainly distributed smaller than 0.5, and the noised images distributed larger than 0.5.

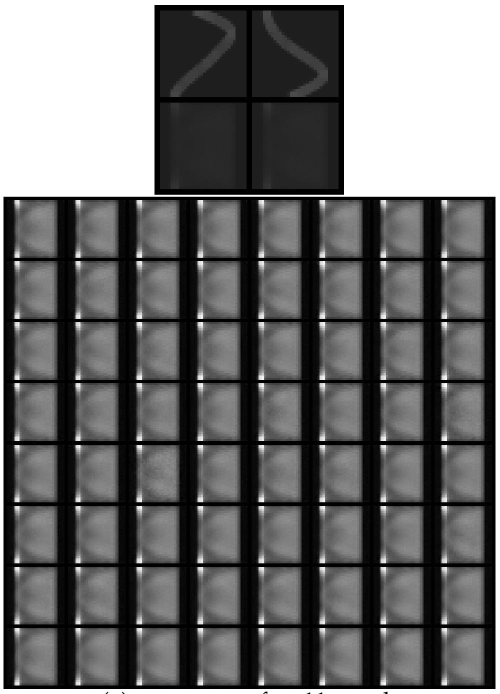
7.3.2 noise specific result

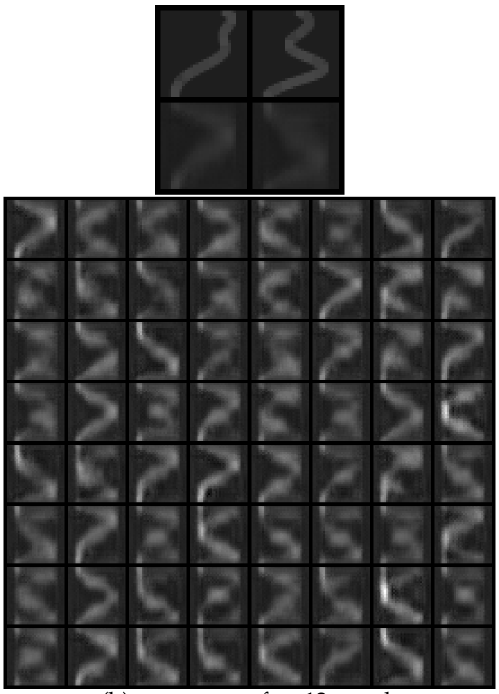
Gaussian noise

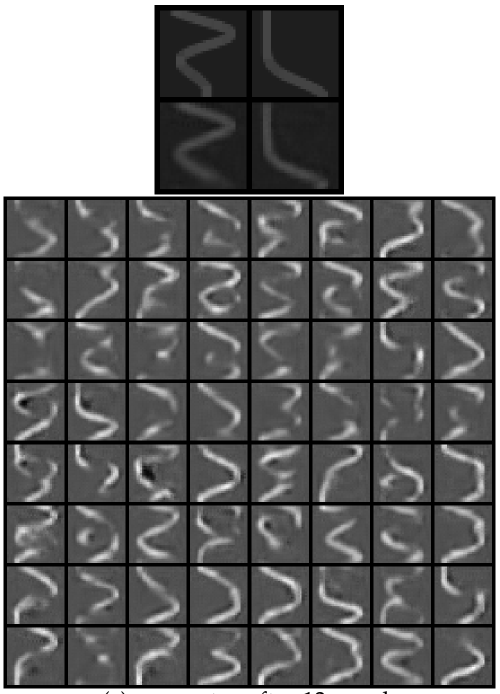
This is the two plots when add on Gaussian noises.
The figure 7.13(a) shows the score plot from the regularizer when Gaussian noise is added to the image. It is shown that with second Gaussian noise added, the regularizer is already giving lower score than the original one, which means the regularizer is not resistant to the addictive Gaussian noise.
The second figure, figure 7.13(b), shows the score plot of the regularizer when Gaussian noise with different standard deviation is added on the plot. The plot, indicate that the there are siginificant difference between the original image and the Guassian nosed image. However, the regularizer is not sensitive to the changes in the standard deviation of the Gaussian noise. The regularizer cannot give a higher punishment to the picture with more noise.

other noise

Other noise have similar results as the gaussian, as seen in the distribution, illustrated in figure 7.12. The distribution of the noised images are relatively similar, but insensitive to the level of noise.


(a) generator after 11 epoch


(b) generator after 12 epoch


(c) generator after 13 epoch

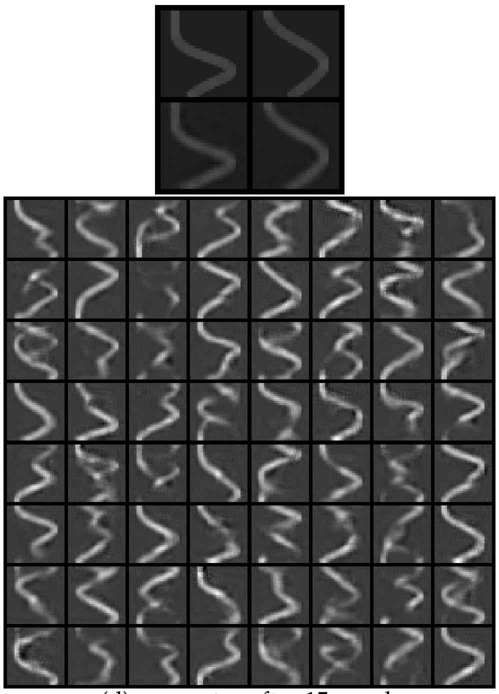
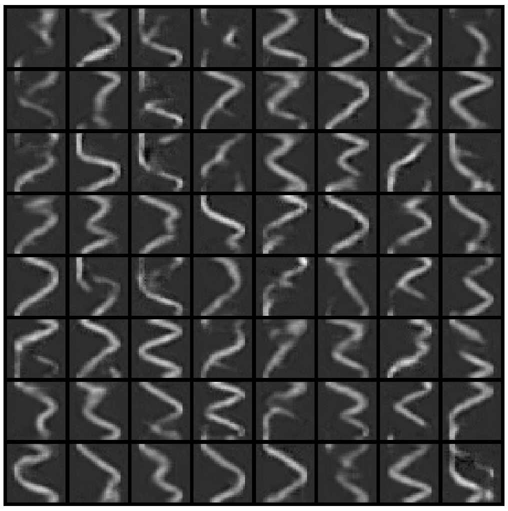
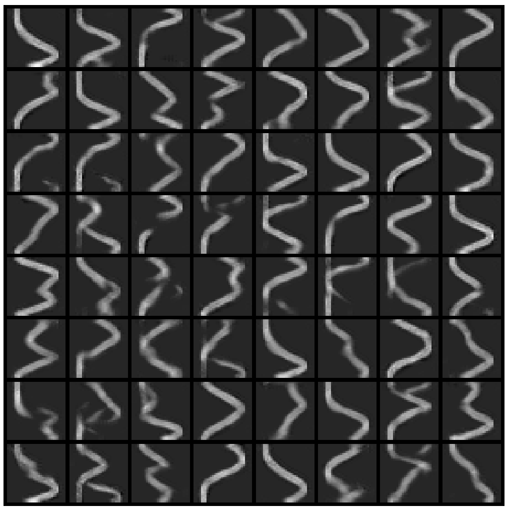

(d) generator after 15 epoch

Figure 7.9: generated images in critical epochs


(a) generator after 15 epochs


(b) generator after 25 epochs

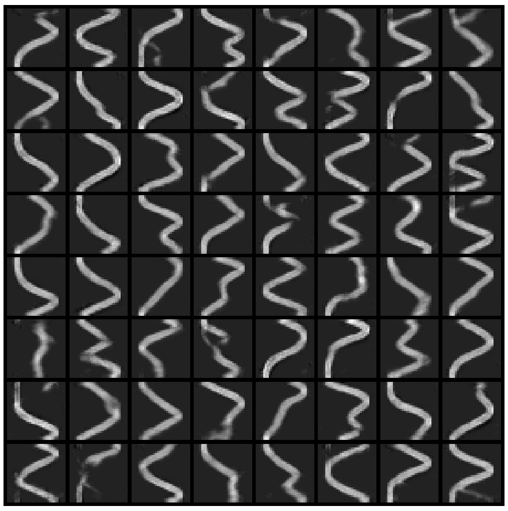

(c) generator after 40 epochs

Figure 7.10: generator results

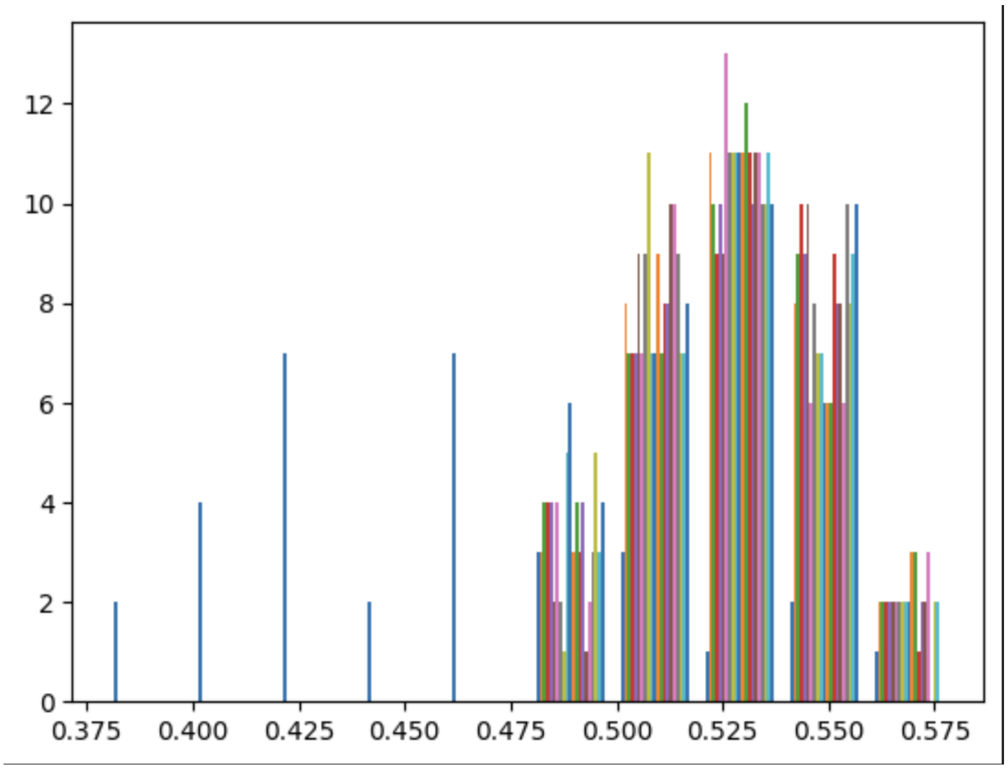


Figure 7.11: evaluation structure of the regularizer

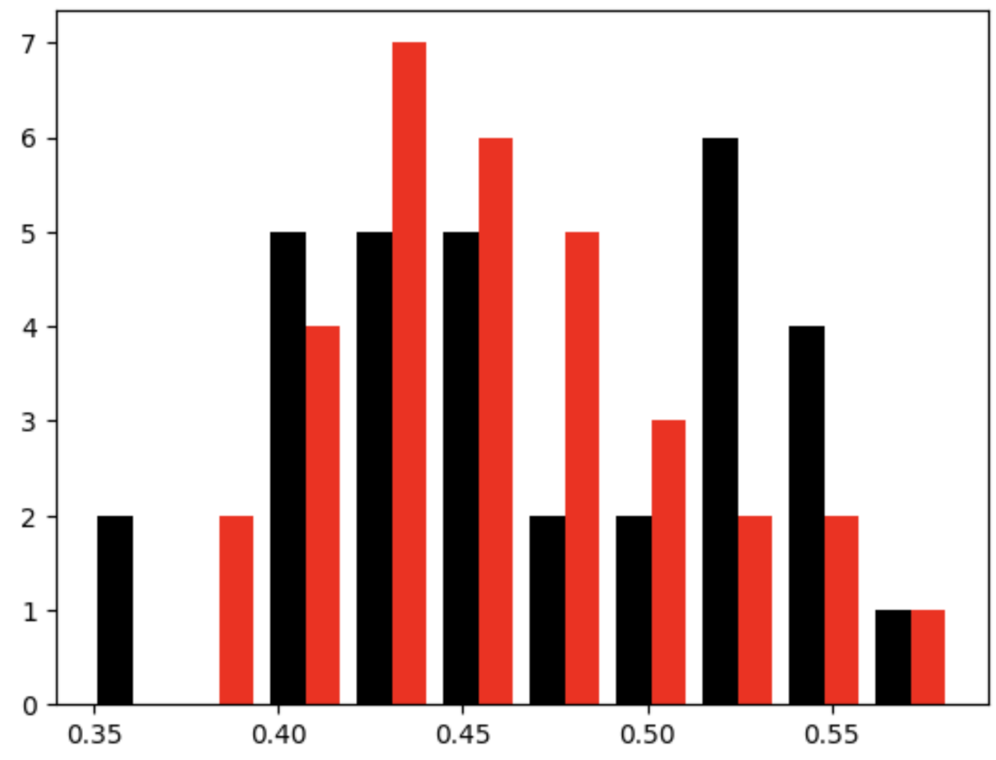
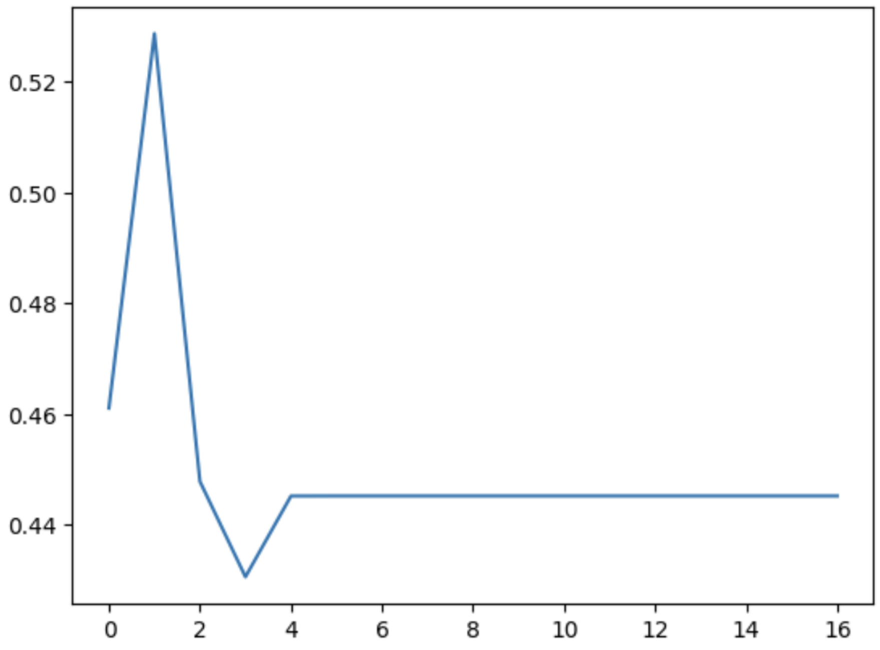
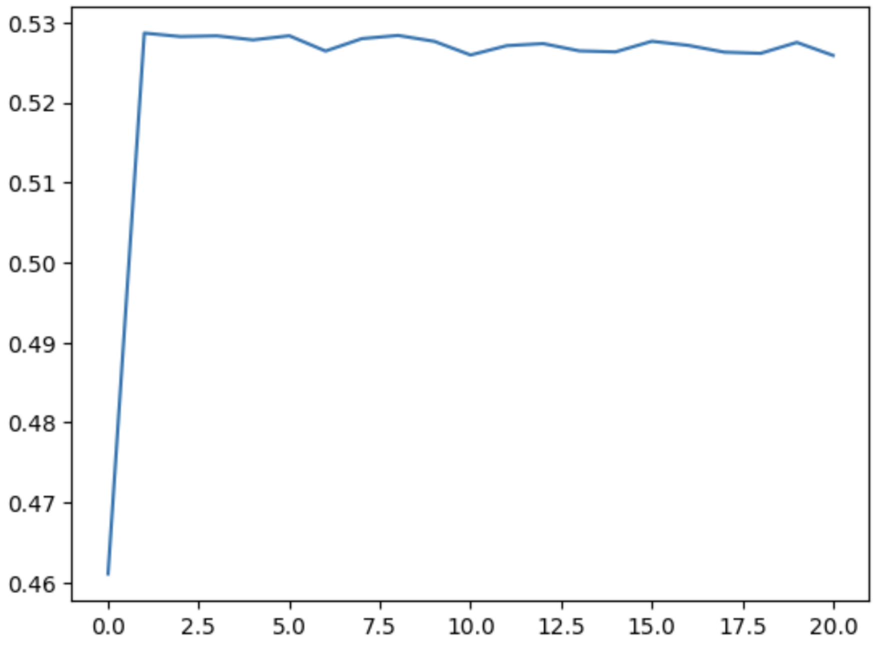


Figure 7.12: evaluation structure of the regularizer



(a) additive Gaussian



(b) Gaussian with different std

Figure 7.13: regularizer agaisnt Gaussian noise

8 Conclusions and discussions

This chapter conclude the work of the whole project, concluding the findings of the whole project, and discuss the further research possibilities. The conclusions would focus on the findings, the contributions of the project and possible further extensions. The discussions would focus on the possible critiques on the research and possible improvement.

8.1 Conclusions

The project basically is illustrated in the figure 8.1. Firstly, the data is created to mimick the defect in the need of the project, and a VAE is trained with GAN structure. An adversarial discriminator is used to force the VAE to generate better images. And the discriminator is forced to distinguish the real images and the fake images. After the convergence in the VAE generator, the encoder and the discriminator is combined to use as the regularizer.

8.1.1 data creation

The project has put some efforts in the mimicking of the defect, trying to finding a extendable way for further full waveform inverse problems settings. The dataset turns out to work well with experiment settings.
The main data used in the experiments were the 2-norm distribution based spline. This DOF is possible for the model to learn in a reasonable time, reaching a convergence. And, interestingly, the result is surprisingly robust for 5-norm distribution based spline. It is possible for the VAE to reconstruct the 5-norm distribution base spline during the validation process. Therefore, a more thorough research on the dataset construction and the relation of performance between different DOF distribution sets could be conducted. We may see the transfer ability of the model between high DOF and low DOF dataset.

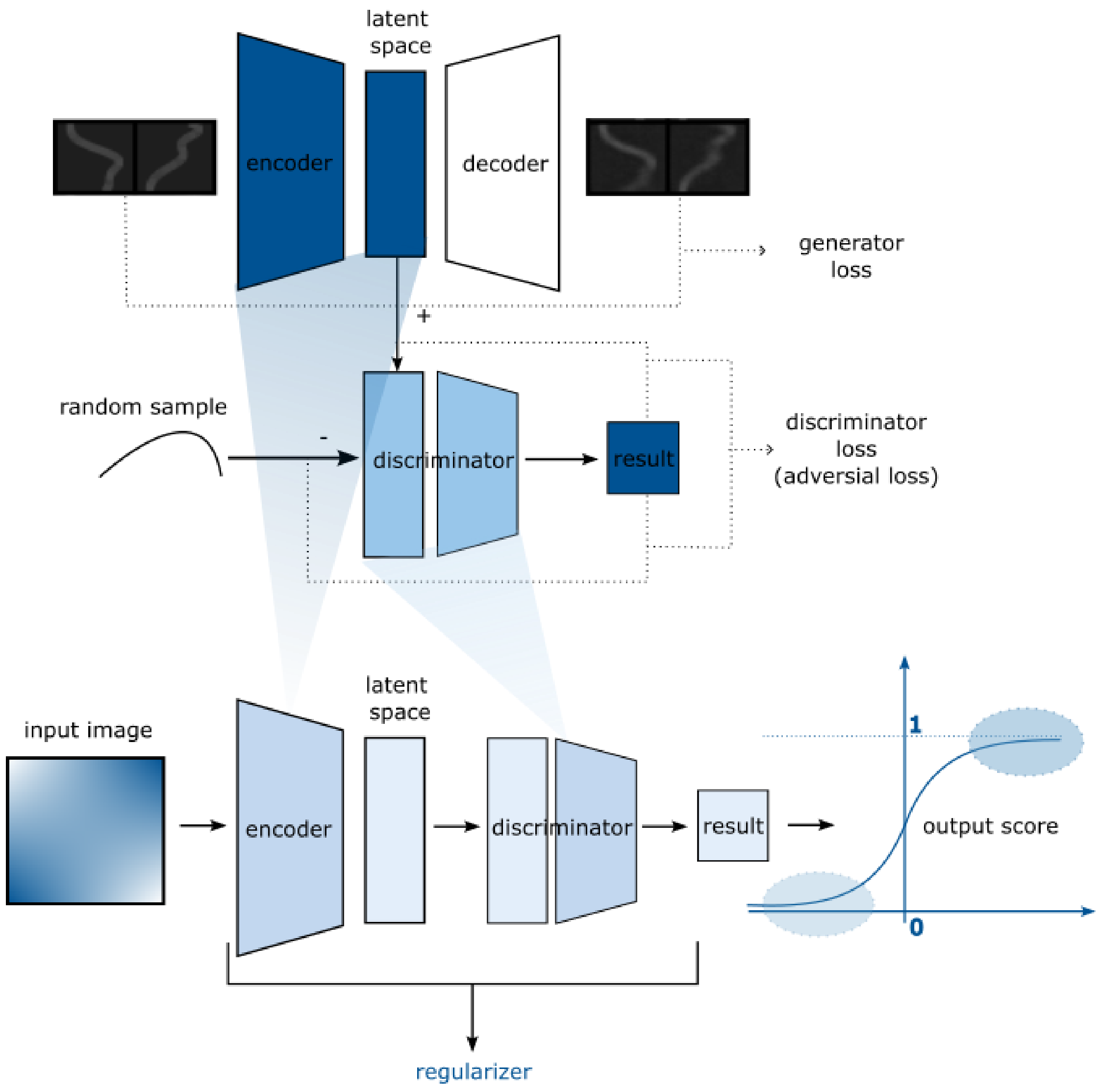


Figure 8.1: conclusion model of the project

8.1.2 GAN-training

To conclude, it is quite difficult to train out a proper GAN networks. The adversarial part adds more challenges to the reach of convergence.
From the initialization, serveral initialization would lead to no convergent solutions. Espeicially, for GAN-trained network, with no inital information for the discriminator, it would be highly possible, that the training lead to no possible result.

The model structure, mainly how fast the features should be extracted, how many layers should be included to extract the features, and how large the latent space should be, are highly related with the data structure. When there are too many weights to learn, although the extraction is not extreme, if the data is not sampling enough for the features, it would be difficult for the model to learn useful information.
The batchsize is also essential to the data training due to the same reason, the model has to got either too much information making the features sparse, or too little information, not able to extract any feature. Therefore, it is important to choose the proper batchsize.

To select the proper hyper parameters mentioned aboove for the training, "overfitting" processs, training the model with small number of images, i.e. small variance in the training data, would on one hand save the training time, and on the other hand receive a brief understanding of the training.
However, it is also possible that the convergence in the overfitting scenario would not be transmittable in a larger dataset scale. A smaller batch-size at the beginning to mimic the overfitting situation might help understand the initialization situation. However, a small batchsize would also lead to changes in gradient, it would be difficult to find the proper gradient that would lead to converge. Therefore, there is no firm answer to the application of the training details. It must be altered according to the result of the current training. The speciality of the GAN trained model is the adversarial part, which is in need of a proper initialized discriminator.

The most siginificant feature of the training process, is the two stages of the generator loss, which is not occurred during a traditional VAE, indicating the adversarial contribution to the generator. This makes the generator more stable and of better performance. However, it made the training for discriminator relatively difficult and not possible to converge. The significant decrease in the training loss leads to the oscillation of the discriminator loss.
A possible soluton would be to train a discriminator at the beginning, and try to improve the performance of the discriminator and the generator together. However, this is not very realistic in the problem setting given.
Another possible solution, is to use the poorly trained generator, the generator at the critical epochs, to trained the initial discriminator. Although this would need a lot of manual interference of the training, it would be possible to reach better results.

As mentioned, the trained generator would be a good source of images. It is already used in the evaluation of the regularizer. There would be more use of the generator. Since, usually people concern more of the performance of the generator, rather than the discriminator. Therefore, there would still be large research gap for research in the training of the discriminator during a GAN-training.

8.1.3 Regularizer

From the evaluation of the regularizer, it is possible to conclude that the regularizer would be able to distinguish from the image that it has been trained with, and distinguish these images from the manually added noised images.

However, the score difference between the values are not very clear. The most scores located between around 0.5, which is not giving large enough punishment to specific data.
Another not ideal situation, would be the insensitivity to the noise level. When given more noised images, the regularizer is not able to recognize and give out higher punishment.

To address the two problems, one possible solution would be, include the specific data into the training scenario, train the model's ability to deal with specific kind of true and fake images. For example, including the image and the noised image in the input, teach the discriminator to distinguish between each other.
This would however diminish the meaningfulness of a GAN-trained model. We use the GAN trained model counting on the generalization ability in faced of attacks that it would bring with the adversarial training process. If we add on the thorough noised dataset for the model to distinguish, it would be more like two separately trained models.
Therefore, it is high time we carefully include some of the noised/counterfactual images in the training dataset, trying to reach a balance between the performance of the model and the generalization ability of the model.

Another possible reason would be that, in the training process for the regularizier, the adversarial loss is not taking too much part of the whole loss, making the update of the discriminator rather small and trivial.
Therefore, a possible solution would be increase the percentage of the adversarial loss. However, if we try to increase the percentage of the adversarial loss. It would be difficult for the whole model to reach the convergence. This is intuitively reasonable, as the initialization of the discriminator is extremely difficult, when the generator has no information of a gradient update, the adversarial part would make the convergence to the correct gradient much more difficult.
A possible solution to this would be, only increase the percentage of the adversarial

part when using a pre-trained generator as initialization of the model. This solution has been tried during the experiement, but did not reach significant result. This may be due to the limit capacity and time invested in hyper-parametering and experimenting. It possible of solving the problem in this way should not yet be ruled out.

8.2 Discussions

An important critique would be, based on the problem setting, it would be difficult to have a pre-trained discriminator. Since, the kind of "attack" the discriminator is faced with is not clear, it would not possible to directly trained such discriminators. This would lead to the difficulty in training the GAN model.
For the primitive discriminator, a possible solution would be make a discriminator based on the knowledge-driven based model. And try to improve the knowledge based model by data-driven methods, for example use the GAN trained model structure.

Another reason to combine the knowledge-based model and the data-driven model would be due to the solution of the inverse problem. The inverse problem choose already the data-driven model, trying to seek a solution using the machine learning, making the regularizer necessary for a better solution.
It is possible that in this scenario, a knowledge-based regularizer would outperform the data-driven regularizer. Theoretically, the two kinds of regularizers are both apppli-cable, but both have weakness. Further reseach maybe done to check the performance difference between them, or combine them together for better regularization of a data-driven inverse problem.

Also, the evalution of the current regularizer is relatively primitive. The accurate performance evaluation of a regularizer can only be done together with the inverse problem. Therefore, further study shall be done to validate the accurate performance of a GAN trained regularizer.

List of Figures

List of Tables

Bibliography

[Ara+18] M. Araya-Polo, J. Jennings, A. Adler, and T. Dahlke. "Deep-learning tomography." In: *The Leading Edge* 37.1 (Jan. 2018), pp. 58–66. ISSN: 1070-485X, 1938-3789. DOI: 10.1190/tle37010058.1.

[Arr+19] S. Arridge, P. Maass, O. Öktem, and C.-B. Schönlieb. "Solving inverse problems using data-driven models." In: *Acta Numerica* 28 (May 1, 2019), pp. 1–174. ISSN: 0962-4929, 1474-0508. DOI: 10.1017/S0962492919000059.

[BKM17] D. M. Blei, A. Kucukelbir, and J. D. McAuliffe. "Variational Inference: A Review for Statisticians." In: *Journal of the American Statistical Association* 112.518 (Apr. 3, 2017), pp. 859–877. ISSN: 0162-1459. DOI: 10.1080/01621459.2017.1285773.

[Doe21] C. Doersch. *Tutorial on Variational Autoencoders*. Jan. 3, 2021. DOI: 10.48550/arXiv.1606.05908. arXiv: 1606.05908[cs,stat].

[Ger70] M. L. Gerver. "Inverse Problem for the One-dimensional Wave Equation." In: *Geophysical Journal International* 21.3 (Dec. 1970), pp. 337–357. ISSN: 0956-540X, 1365-246X. DOI: 10.1111/j.1365-246X.1970.tb01796.x.

[HS18] D. Ha and J. Schmidhuber. "World Models." In: (Mar. 28, 2018). DOI: 10.5281/zenodo.1207631. arXiv: 1803.10122[cs,stat].

[IS15] S. Ioffe and C. Szegedy. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. Mar. 2, 2015. arXiv: 1502.03167[cs].

[KW22] D. P. Kingma and M. Welling. *Auto-Encoding Variational Bayes*. Dec. 10, 2022. DOI: 10.48550/arXiv.1312.6114. arXiv: 1312.6114[cs,stat].

[Li+20] S. Li, B. Liu, Y. Ren, Y. Chen, S. Yang, Y. Wang, and P. Jiang. "Deep-Learning Inversion of Seismic Data." In: *IEEE Transactions on Geoscience and Remote Sensing* 58.3 (Mar. 2020), pp. 2135–2149. ISSN: 0196-2892, 1558-0644. DOI: 10.1109/TGRS.2019.2953473.

[LSÖ] S. Lunz, C. Schoenlieb, and O. Öktem. "Adversarial Regularizers in Inverse Problems." In: ().

[MDA] D. Maclaurin, D. Duvenaud, and R. P. Adams. "Autograd: Effortless Gradients in Numpy." In: ().

[Nak17] J. Nakamura. *Image Sensors and Signal Processing for Digital Still Cameras*. Google-Books-ID: aaTyk2USX1MC. CRC Press, Dec. 19, 2017. 372 pp. ISBN: 978-1-4200-2685-6.

[Oht17] J. Ohta. *Smart CMOS Image Sensors and Applications*. Google-Books-ID: _7NLzflrTrcC. CRC Press, Dec. 19, 2017. 267 pp. ISBN: 978-1-4200-1915-5.

[Ois+] A. Oishi, K. Yamada, S. Yoshimura, G. Yagawa, S. Nagai, and Y. Matsuda. "Neural Network-Based Inverse Analysis for Defect Identification with Laser Ultrasonics." In: ().