Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Fall Semester 2022 Prof. H.-A. Loeliger

Master's Thesis

Using A Generative Adversarial Network For Generating Microphone Array Data

Gaspard Ulysse Fragnière

Advisor: Adam Kujawski

Acknowledgments

I would like to thank ...

Berlin, 6 February 2023

Gaspard Ulysse Fragnière

Abstract

Here comes the abstract...

Zusammenfassung

Das ist die Zusammenfassung ...

Contents

Chapter 1

Introduction

1.1 Introduction

The problem of Acoustical Source Characterization (ASC) is an important problem of Acoustics. It arises for instance industrial applications such as the characterization of noise on an airplane in a wind tunnel or sound source localization in smart assistants, such as Google Home or Alexa. Traditionaly this problem is tackled with methods based on the physics of sound propagation (e.g. TDoA, beamforming) or with statistical inference (e.g. Sparse Bayesian Learning) **TODO: cite papers here**. Source characterization relies on microphone array data, since accurate result cannot be obtained using data from a single microphone.

The recent success of Deep Learning (DL) based method in other field of research (e.g. Computer Vision **TODO: cite papers here**) led to believe that Deep Neural Networks (DNN) based approaches could provide state-of-the-art result in solving the ASC problem. [1], [2], [3], [4], [5] and [6] propose state-of-the-arts DL-based methods for Source Characterization. **TODO: change all the mentioned paper by only citing the compilation of Grumiaux**

A common issue faced while implementing DL-based methods is that significant quantities of well structured data are required. In the literature, the data has been obtained using the following approaches:

- **Real Mesurement**: To create the different samples of such a dataset, sounds emitted with a loudspeaker or human voices are recorded in a real acoustic enviromnent. Eventhough such a method allows for the creation of perfectly realistic samples, it does not come without any issue. Indeed, it is very tedious and time-consuming to record in different environments. Additionally, all the environment for measurement need to physically exists, which limits the quantity of possible samples. Moreover, to build a high quality data set, expensive equipment is required to have an accurate groundtruth (i.e. precisely identify the location of the sources). In the literature, [7] and [8] have used such an approach.

- **Synthetic Data**: The sounds used are artificial (i.e. white noise, sine

wave). The room acoustic is also simulated. The dry sound is convolved with a simulated Room Impulse Response (RIR) to mimic the effect of room acoustics (e.g. reverberation). Compared to real measurement, this approach allows sample in more diverse environment. Indeed RIR for rooms of arbitrary size, different source position as well as different dry signals can be used for the training. The issue with such a method is the important amount of time and storage required for the creation of the datasets. E.g. [9], [10] and [11] created their datasets in this way.

- **Semi-synthetic data**: The creation of such a dataset is similar the creation of synthetic dataset. The difference lies in the fact that the dry sound source used and the RIR are measured and not simulated. Then, the samples of such a dataset are generated by convolving dry sounds with measured RIR. This method is not the best suited, since it is very time-consuming to generate a data set with enough samples for training a DL-based algorithm. Indeed, measuring all the RIR lead to the issues faced with real measurement. [12] use such an approach for to obtain their data.

It is to be noted that with the above mentioned data creation method arises two issues. On one hand, data cannot be created online. Indeed, when using measurement data or semi-synthetic data, measurement need to be performed and stored before any training can occur. On the other hand, synthetic data can be created on the fly when training a model, but this come with the tradeoff that the current synthetization process do not allow to have data as realistic as measurement data.

1.2 State of the Art

In the past years, DL-based approaches have shown to be able to learn and realistically generate complex data structures (e.g. generation of pictures of faces in the field of Computer Vision **TODO: here cite some paper**). Those breakthroughs lead to believe that similar data generation methods could be used to fix the above-mentioned issues and allow for realistic online data generation. We therefore need to identify what acoustic quantities:

- have already been generated using a DL approach
- are potential feature for a Source Characterization Algorithm.

1.2.1 Generation of Signal

[13], [14], [15] use Generative Adversial Network (GAN) to generate realistic audio waveform. [13] and [14] specifically focus on the generation of audio waveform conditioned on a spectogram (cGAN). On the other hand, [15] designed a GAN to generate realistic audio waveform of single music notes played by an instrument. Those approaches focus on generating single channel data

and hence might not be fit to generate microphones array data, where each data of the different channels are correlated. It is relevant to note that the GAN designed by [13] is the one implemented in [16] in order to compare the accuracy of a network for single source DoA estimation when trained with different sound classes.

1.2.2 Generation of Impulse Response

[17] introduces a GAN approach to generate artificial Acoustical Impulse Response (AIR) of different environment in order to generate data for a NN used for classification of acoustic environment.

[18] proposes a fast method (NN-based) for generating Room Impulse Response (RIR). The input paramaters of the networks used for creating the IR are the desired dimensions of the rectangular room, listener position, speaker position and reverberation time (T_{60}).

Currently, these methods only allow to generate impulse responses for only a single listener, hence they would need to be extended to generate several correlated IR per microphones of the array.

1.2.3 Generation of potential NN feature

[19] proposes an approach to generate another acoustic feature: the phase of the relative transfer function (RTF) between two microphones. In this paper a Variational Auto Encoder (VAE) is designed to simultaneously generate phases of RTF and classifying them by their Direction of Arrival (DoA).

[20] use a GAN to generate Sample Cross Spectra Matrices (CSM) for a given DoA. In their approach, the GAN is trained with data only coming from one DoA, making it unable to generate sample for different DoA.

1.2.4 Other possible approaches to generate the data

In [21] introduce a low complexity model-based method for generating samples of microphones phases. This method proposed is not based on DL. Indeed, it is based on a statistical noise model, a deterministic direct-path model for the point source, and a statistical model. The claim of this paper is that the low complexity of the proposed model makes it suited for online training data generation.

[22] introduce a CNN for denoising (i.e. removing the effects of reverberation and multipath effects) on the Generalization Cross Correlation (GCC) matrix of an array of microphone. More specifically than a CNN, the network used has a encoder-decoder structure.

1.2.5 Take-away of the literature survey TODO: find better title

Several methods were introduced to generate acoustical quantities, but most do not propose methods that are really suited for multi-channel data generation

using deep learning. The approach proposed in [20] is the one that is that seem the better suited for the purpose of this thesis. In this sense, the work proposed here will take its roots in this paper.

1.3 Fundamentals

1.3.1 Propagation model and the Cross Spectral Matrix

If we consider M spatially distributed receivers, J uncorrelated sources and a linear propagation model, the complex sound pressure at the mth microphone is given by

$$p(\mathbf{r}_m, \omega) = \sum_{j=1}^{J} h_{mj}(\omega) q(\mathbf{r}_j, \omega) + n(\mathbf{r}_m, \omega) \tag{1.1}$$

Where ω is the angular frequency and h_{mj} is the transfer function describing the the sound propagation from the jth source to the mth sensor. $n(\mathbf{r}_m, \omega)$ models independent noise. The above equation can be rewrittem as a matrix form

$$\mathbf{p} = \mathbf{H}\mathbf{q} + \mathbf{n} \tag{1.2}$$

The Cross Spectral Matrix (CSM) or the Sample Covariance Matrix (SCM) is a quantity used in most beamforming algorithm. It can be approximated using Welch's method (**TODO: cite here paper**) as

$$\hat{\mathbf{C}} = \frac{1}{B} \sum_{b=1}^{B} \mathbf{p}\mathbf{p}^H \tag{1.3}$$

with B snapshots. The CSM is used as the starting point in many microphone array methods, because most information about the location of a source signal are contained in it.

1.3.2 Eigendecomposition and Rank I Cross spectral matrix

If we consider a Cross spectral matrix $\mathbf{A}$ of dimension $M \times M$, it can be factorized into a canonical form: a representation by its eigenvalues and eigenvectors using:

$$\hat{\mathbf{C}} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^H \tag{1.4}$$

where $\mathbf{V} = [\mathbf{v}_1^T, \ldots, \mathbf{v}_M^T] \in \mathbb{C}^{M \times M}$, $\mathbf{v}_i$ being the ith eigenvector and where $\mathbf{\Lambda} \in \mathbb{R}^{M \times M}$ is a diagonal matrix, where $\mathbf{\Lambda}_{ii}$ is the ith eigenvalue, corresponding to the ith eigenvector. The eigendecomposition is useful because the eigenvalues provide good insights to separate signal from noise., as well as estimating a source strength.

[23] uses the eigendecomposition to introduce the Rank I Cross Spectral Matrix. If we consider approximated CSM $\hat{\mathbf{C}} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^H$, then the Rank I CSM $\hat{\mathbf{C}}_i$ can be computed with:

$$\hat{\mathbf{C}}_i = \mathbf{v}_i \mathbf{\Lambda}_{ii} \mathbf{v}_i^H \tag{1.5}$$

As we'll see in the next subsection, the rank I CSM can be used to create a beamforming map for only one eigenvalue, corresponding to one audio source.

1.3.3 Conventional beamforming

Beamforming is a technique to characterize sound sources using the different signals of an array of microphones. All the microphones of the array record the sound of the source, all with different time delays, depending on the different microphones position, as well as source's location. Using the data from every microphones, a beamforming map can be created, namely a map where the maximum value is at the position of the sound source.

More formally, the situation introduced above, in the propagation model, is considered. We remind that we have M microphones organised as an array but this time only with a single source (i.e. $J = 1$). Moreover we have the sound pressure vector $\mathbf{p} \in \mathbb{C}^M$ at every microphone and the corresponding approximated CSM $\hat{\mathbf{C}} = \frac{1}{B}\sum_{b=1}^{B} \mathbf{p}\mathbf{p}^H$.

We can then define a scan grid of N potential position of sound sources. For the sources, the assumption made is that the propagation model is a monopole source. Each position of the grid has a position vector ξ_n. For each grid point ξ_n, we compute the expected signal that would be recorded by each microphone. Every microphone as a position vector x_m, for $m \in [1, \dots, M]$. This allows to introduce the steering vector $g_{n,m} \in \mathbb{C}^M$, defined as

$$g_{n,m} = \frac{\exp(-2\pi i f \Delta t_{n,m})}{4\pi ||x_m - \xi_n||} = \frac{\exp\left[-2\pi i f \frac{||x_m - \xi_n||}{c}\right]}{4\pi ||x_m - \xi_n||} \tag{1.6}$$

Where f is the sound frequency under consideration and $\Delta t_{n,m}$ is the propagation time from source n to microphone m. $c = 343$ [m/s] is the speed of sound in air. Other definition of the steering vector for different propagation model can be found in [24].

Using the steering vectors $g_{n,m}$ for $(n, m) \in [1, \dots, N] \times [1, \dots, M]$ and the approximated CSM $\hat{\mathbf{C}}$, we can compare the model sound pressure, with the recorded sound pressure. By finding a match, the source location can be identified. To do so, we compute the source autopower per scan grid point ξ_n:

$$A(\xi_n) = \frac{1}{2} \frac{\mathbf{g}_n^H \hat{\mathbf{C}} \mathbf{g}_n}{||\mathbf{g}_n||^4} \tag{1.7}$$

This provides us with a source map. An example of source map can be seen in Fig.1.1. Moreover, [23] shows that equation 1.7, can also be used with a Rank I CSM $\hat{\mathbf{C}}_i$, giving

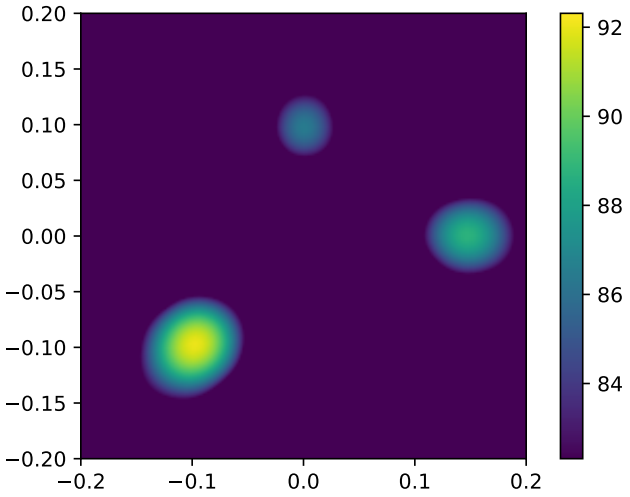

Figure 1.1: Example of beamforming source map, with three sources.

$$A(\xi_n) = \frac{1}{2}\frac{\mathbf{g}_n^H \hat{\mathbf{C}}_i \mathbf{g}_n}{||\mathbf{g}_n||^4} \tag{1.8}$$

This gives us a beamforming map corresponding to only one single source, the one with associated eigenvalue $\mathbf{\Lambda}_{ii}$.

1.3.4 GAN

[25] introduce a new approach to solve the problem of generative model. The goal is to learn the probability distribution that was used to generate samples, by observing them.

The method introduced in this paper is called Generarive Adversial Network (GAN). The idea is to create a game (in the game theory sense) where two networks compete against each other. The first network is called a discriminator and its goal is to determine real from fake samples. The other network is a generator with the aim to produce data realisitic enough that the discriminator cannot determine that it has been generated.

The generator takes as input a random vector and use it generate a sample. This random vector is referred to as latent variable. The vector space of latent variable is called latent space. After training, the generator should be a mapping from the latent space to the data space. The latent space is a representation of smaller dimension of the data space.

The discriminator takes as input a real or generated sample and predicts its authenticity. The discriminator is a simple binary classifier. The real sample come from the datasets and the fake are outputs of the generator.

Both models are trained simultaneously. First the generator create a batch of fake sample. This batch is then fed to the discriminator, alongside a batch of real samples. For each of them, the discriminator makes a prediction about their authenticity. The discriminator then gets updated based on how accurate it was at classifying samples and the generator based on how many times fake sample were able to fool the discriminator. We can see here that the training of both networks is supervised, on the contrary of typical generative models.

In a gaming theory framework, both networks are competing in a zero-sum game. This means that if one network performs well at its task and gets rewarded by little weights update, the other must have performed poorly and hence gets penalized by heavy weights updated. E.g. if the discriminator was successful at classifying all samples, it means that the generator had not been able to fool the discriminator by producing any realistic fake samples.

1.3.5 DCGAN

[26] introduce GAN, but unlike in [25], the architecture of the discriminator and generator is not achieved with regular perceptron, but with Convolutional layer. We first remind how a regular perceptron layer works. For an input vector $\mathbf{x} \in \mathbb{R}^k$ and output vector $\mathbf{y} \in \mathbb{R}^l$, a perceptron layer consists of two parts:

- an activation $\mathbf{a} = \mathbf{W}\mathbf{x} + \mathbf{b}$
- a non-linearity $y = f(\mathbf{x}; \theta) = \sigma(\mathbf{a})$

where $\mathbf{W} \in \mathbb{R}^{l \times k}$ are the weights of the perceptron and $\mathbf{b} \in \mathbb{R}^l$ its bias. $\sigma(\cdot)$ is called the activation function and is the source of non linearity in neural networks. An example of such activation function is the Rectified Linear Unit (ReLU) where:

$$\sigma_{\text{ReLU}}(\mathbf{a}) = [\max(a_0, 0), \ldots, \max(a_{l-1}, 0)]^T \tag{1.9}$$

We can now introduce the architecture of a convolutional layer. Without loss of generality, we assume that our network reecive as input an square image with a single channel, i.e. $\mathbf{H_0} \in \mathbb{R}^{d_0 \times (m \times m)}$ with $d_0 = 1$, transforms it with one square kernel, i.e. $\mathbf{K} \in \mathbb{R}^{s \times (r \times r)}$ with $s = 1$ to output another square image with a single channel, i.e. $\mathbf{H_1} \in \mathbb{R}^{d_1 \times (n \times n)}$, with $d_1 = 1$

The relationship between input image $\mathbf{H_0} \in \mathbb{R}^{m \times m}$, output image $\mathbf{H_1} \in \mathbb{R}^{n \times n}$ and kernel $\mathbf{K} \in \mathbb{R}^{r \times r}$ is defined as:

- an activation $\mathbf{A} = \mathbf{H_0} * \mathbf{K}$
- a non-linearity $\mathbf{H_1}_{(i,j)} = \sigma(\mathbf{A}_{(i,j)})$

[26] proposes a set of constraint that makes the use of convulotional layers possible in GAN setting.

1.3.6 WGAN

[27] introduce a new type of Generative Adversial Network (GAN): the Wasserstein GAN (WGAN). The claim is that WGAN improves the stability in learning and get rid of typical problem of the traditional GAN approach such as Mode Collapse or Convergence failure. Mode collapse is when a GAN fails to fully converged because the generator learns only to generate a subset of the real data in order to fool the discriminator. This leads to a situation where the GAN is able to produce realistic data but not all kind of data, which is not a desired behaviour. More specifically, [27] provides the following insights:

- Introduces the Earth-Mover (EM) or Wasserstein distance. It is a metric used to quantify the distance between two probability distributions.
- Analyses how the EM distance behaves compared to other distances between probability distribution (e.g. Kullback-Leibler distance).
- Define a GAN minimizing an approximation of the EM distance, namely the WGAN.
- Show that unlike traditional GANs, WGANs do not need to maintain a balance when training the discriminator and generator. Indeed in regular GAN approach, it was crucial to avoid the discriminator to become too good before the generator, since this would prevent the generator to learn any distribution.
- Provides useful insights to determine whether the WGAN has converged or not. Determining convergence was a hard problem with the regular GAN approached and it was typically assessed by visually determining if the produced samples were realistic or not.

The Earth-Mover or Wasserstein distance

The goal of WGAN, remains the same as GAN, namely solving the problem of generative model. More precisely, this mean approximating the probability distribution P_r of some data by a distribution P_θ. For this reason, it is necessary to have metrics to quantify distance between two probability distributions P_r and P_m. Example of such distances are the Kullback-Leibler divergence or the Jensen-Shannon divergence. In WGAN, the distance used is the Earth-Mover (EM) distance or Wasserstein-1, defined as

$$W(\mathbb{P}_r, \mathbb{P}_m) = \inf_{\gamma \in \Pi(\mathbb{P}_r, \mathbb{P}_m)} \mathbb{E}_{(x,y)\sim\gamma}[||x - y||] \tag{1.10}$$

Where $\Pi(\mathbb{P}_r, \mathbb{P}_m)$ is the set of all joint distribution $\gamma(x, y)$ whose marginals are respectively $\mathbb{P}_r$ and $\mathbb{P}_m$. Informally, $\gamma(x, y)$ shows how much "mass" must be carried to transform $\mathbb{P}_r$ into $\mathbb{P}_m$. The EM distance is then "the cost" of the optimal "transport".

[27] shows that the EM distance is better than the following metrics to quantify distances between probabilty distributions, namely

- the Total Variation metric, defined as

$$\delta(P_r, P_g) = \sup_A |P_r(A) - P_g(A)| \tag{1.11}$$

- The Kullback-Leibler divergence defined as

$$KL(P_r||P_g) = \int_x \log\Big(\frac{P_r(x)}{P_g(x)}\Big) P_r(x)dx \tag{1.12}$$

  It is important to note that $KL(P_r||P_g) \neq KL(P_g||P_r)$ (i.e. not symmetric)

- The Jensen-Shannon divergence. If we have P_m a mixture distribution with $P_m = \frac{P_r+P_g}{2}$, then Jensen-Shannon divergence is defined as

$$JS(P_r, P_g) = \frac{1}{2}KL(P_r||P_m) + \frac{1}{2}KL(P_g||P_m) \tag{1.13}$$

[27] uses the following example to explain the reelvance of the EM distance. If we consider probability distributions over $\mathbb{R}^2$. Let us consider the data distribution $(0, y)$ with, $y \sim U[0, 1]$. We consider the the family of distribution P_θ, where $P_\theta = (\theta, y)$ again with $y \sim U[0, 1]$. We want the distance metric to move closer to zero, as $\theta \to 0$. For such a distribution, this is how the different distances behave:

- Total Variation

$$\delta(P_0, P_\theta) = \begin{cases} 1 & \text{if } \theta \neq 0 \\ 0 & \text{if } \theta = 0 \end{cases} \tag{1.14}$$

- Kullback-Leibler Divergence

$$KL(P_0||P_\theta) = KL(P_\theta||P_0) = \begin{cases} +\infty & \text{if } \theta \neq 0 \\ 0 & \text{if } \theta = 0 \end{cases} \tag{1.15}$$

- Jensen-Shannon Divergence

$$JS(P_0, P_\theta) = \begin{cases} \log(2) & \text{if } \theta \neq 0 \\ 0 & \text{if } \theta = 0 \end{cases} \tag{1.16}$$

- the Earth mover distance

$$W(P_0, P_\theta) = |\theta| \tag{1.17}$$

This simple example shows that there exist distributions for which the Total Variation, the Kullback-Leibler Divergence and the Jensen-Shannon Divergence do not converge, whereas the Earth mover Distance does. [27] then confirms the insight provided by this example with the the two following theorems (without proof here, but the proofs can be found in [27])

Theorem 1: Let P_r be a fixed distribution. Let Z be a random variable. Let g_θ be a deterministic function parametrized by θ and let $P_\theta = g_\theta(Z)$. Then

1. If g is continuous in θ, then so is $W(P_r, P_\theta)$
2. If g is sufficiently nice, then $W(P_r, P_\theta)$ is continuous everywhere and differentiable almost everywhere.
3. Statements 1 and 2 do not hold for neither $JS(P_r, P_\theta)$, $KL(P_r||P_\theta)$ or $KL(P_\theta||P_r)$.

This first theorem shows that out of the four loss functions above-mentioned, only the Eart-Mover has some guarantees of both continuinity and differentiability, which are nice gurantees for a loss function.

Theorem 2: Let P be a distribution and $(P_n)_{n\in\mathbb{N}}$ be a sequence of distributions. The the following is true

1. The following statements are equivalent:
   - $\delta(P_n, P) \to 0$
   - $JS(P_n, P) \to 0$
2. The following statements are equivalent:
   - $W(P_n, P) \to 0$
   - $P_n \to P$, with here "$\to$" being convergence in distribution for random variables.
3. $KL(P_n||P) \to 0$ or $KL(P||P_n) \to 0$ implying statement 1.
4. Statement 1 implies statement 2.

The above theorem proves that every distribution that converges under the Kullback-Leiber divergence (in any direction), the Jensen-Shannon divergence or the Total Variation also converges under the Earth-Mover distance. It also shows that a smaller difference between two probability distribution implies a smaller EM distance.

Unfortunately the EM distance is intractable, due to the infinum part in its equation. But using the Kantorovich-Rubinstein duality, it can be reformulated as:

$$W(\mathbb{P}_r, \mathbb{P}_\theta) = \sup_{||f||_L \leq 1} \mathbb{E}_{x\sim\mathbb{P}_r}[f(x)] - \mathbb{E}_{x\sim\mathbb{P}_\theta}[f(x)] \tag{1.18}$$

Where the supremum is over 1-Lipschitz functions. It is important to note that if we replace $||f||_L \leq 1$ by $||f||_L \leq K$, i.e. consider also the K-Lipschitz functions, then we obtain $K \cdot W(\mathbb{P}_r, \mathbb{P}_\theta)$. Hence, for a family of functions $\{f_w\}_{w\in\mathcal{W}}$ (all functions being K-Lipschitz), we can consider solving the optimization problem:

$$\max_{w\in\mathcal{W}} \mathbb{E}_{x\sim\mathbb{P}_r}[f_w(x)] - \mathbb{E}_{z\sim p(z)}[f_w(g_\theta(z))] \tag{1.19}$$

Note: we can approximate the solution of the above mentioned problem with a Neural Network with weights $\mathcal{W}$. $\mathcal{W}$ need to be compact to assure that the function f_w are K-Lipschitz. Therefore in order to have $\mathcal{W}$ being compact, [27] proposes to simply clip the weights, such that they lay in a small box $[-c, c]^l$, e.g. with $c = 0.01$

Necessary changes to turn a GAN into a WGAN

Implementation of a WGAN requires a few changes from implementation of a regular GAN, i.e.

- Use a linear activation function in the output layer of the "discriminator" model (instead of sigmoid). The "discriminator" then becomes a critic that quantify the realness of a sample, instead of discriminating between real or fake.
- Use Wasserstein loss to train the critic and generator models that promotes larger difference between scores for real and generated images.
- Constrain critic model weights to a limited range after each mini batch update (e.g. [-0.01,0.01]). As seen above, this is to ensure, that the function estimated in the for approximting the Wasserstein distance are K-lipschitz, a necessary condition.
- Update the critic model more times than the generator each iteration (e.g. 5). Contrary as in a GAN, this is not a issue. Indeed, switching to the EM distance, allow for more stability when training the two networks in the WGAN. Moreover, the fact that the EM distance is continuous and differentiable means that we should train the critic until optimality.
- Use the RMSProp version of gradient descent with small learning rate and no momentum (e.g. 0.00005).

1.3.7 WGAN-GP

As we have seen above, WGAN improve the stability in training the critic ($\approx$ discriminator). But it is still subject to poor sample generation, convergence failure or mode collapse. This is due to the weight clipping happening while training the critic. In order to remedy to this, [28] propose to replace weight clipping by the introduction of a penalization of the norm of gradient of the critic with respect to its input. The issue is that trying to orient the critic to 1-Lipschitz function by weight clipping, biases the critic for too simple function. [28] observes that implementing the Lipschitz constraint using weights clipping leads to either exploding or vanishing gradient, unless the threshold c used for the clipping is carefully fine-tuned.

The gradient penalty

In order to enforce the Lipschitz constraint, [28] proposes to add a penalty term to the loss function. The loss function then becomes:

$$L = L' + P \tag{1.20}$$

where:

- Original loss function:

$$L' = \mathbb{E}_{\tilde{\mathbf{x}} \sim \mathbb{P}_g}[D(\tilde{\mathbf{x}})] - \mathbb{E}_{\mathbf{x} \sim \mathbb{P}_r}[D(\mathbf{x})] \tag{1.21}$$

- Penalty:

$$P = \lambda \mathbb{E}_{\hat{\mathbf{x}} \sim \mathbb{P}_{\hat{\mathbf{x}}}}[(||\nabla_{\hat{\mathbf{x}}} D(\hat{\mathbf{x}}) - 1||)^2] \tag{1.22}$$

The goal of the penalty is to enforce the 1-Lipschitz constraint. Indeed, by definition, a function is 1-Lipschitz if and only if its gradient norm is smaller or equal to 1 everywhere. It can be easily seen that the penalty is here to enforce this constraint. In order to make such a penalty tractable, a soft version of the penalty is considered, where the constraint is only enforced on a the gradient norm of a few random samples $\hat{\mathbf{x}} \sim \mathbb{P}_{\hat{\mathbf{x}}}$.

The sampling distribution $\mathbb{P}_{\hat{\mathbf{x}}}$ is defined by sampling uniformly on on a line between a pair of points respectively sampled from $\mathbb{P}_r$ and $\mathbb{P}_g$. This was proven experimentally to give sufficiently good results.

In [28], the penalty coefficient λ was set always to 10 in all experiences done. No batch normalization was used in [28]. They claim that batch normalization shifts the discriminator problem from trying to match a single input to a single output from trying to macth a batch input to a batch output. This makes the penalty invalid, since the penalization is perfomed with each input individually and batch normalization introduce correlation between samples. Instead of batch normalization, [28] recomnends layer normalizations.

1.3.8 Assessing performances of a WGAN or WGAN-GP compared to GAN

A problem with regular GANs (from [25]) is that it is really hard to know when a good model has been found, from looking only at the loss function. The quality of a model is typically assessed by visually looking at the generated sample and deciding if they are realistic or not. But [27] shows that in a WGAN, the Wasserstein loss is a meaningful loss function, that is correlated with the quality of the sample produced by the generator. This is not true in a regular GAN and is actually one of the main selling point of a WGAN over GAN.

[28] shows that this property of the WGAN still holds true for the WGAN-GP. The critic loss should typically start at zero, then drops and work its way back to zero. This is when the WGAN-GP has converged and is hence producing realistic samples.

Chapter 2

Methods

First, it is relevant to note that the data used for source characterization in [1], [3], [4], [6] is the Cross Spectral Matrix (CSM), i.e. a direct representation of the signals received in the array of microphone. Indeed those approaches do not use direct recording of microphone input but instead this feature extracted from the raw data. This is crucial because it means that recording, simulating or generating raw microphone data is not necessary, if realistic CSMs could be generated directly.

The CSM can be calculated analytically (as done by Castellini **TODO: add citation**), or an estimate of the true CSM can be calculated (Kujawski and Sarradj 2022 **TODO: add citation**). The latter is often done by from the measured time means of Welch's method (Equation 1.3). Calculating the CSM analytically is fast, but do not yield fully realistic CSM. A third approach is to draw from a Wishart distribution. This distribution approximates the CSM estimate given by Welch's method without the need to sample the time data. Instead, on can directly sample the elements of the estimated CSM from specific random distributions.

It is important to note that there is a difference between the true CSM (analytically) and the estimated (Welch's method, Wishart distributed sampling). The latter is the one occurring in reality as the number of averages is bounded.

In this chapter, methods to generate approximation of Cross Spectral Matrix are inverstigated. The idea behind our methods takes its root in [20]. In [20], A WGAN-GP is used to generate the B snapshots $\mathbf{p}\mathbf{p}^H$ (see equation 1.3). Instead, here, WGAN-GPs are designed to generate the eigenvalues and eigenvectors of of the eigendecomposition of a CSM (see equation 1.4). The eigenvectors could be normalized and the eigenvalues be scaled such that they all lies in $[0, 1]$. Both these normalization and scaling process could allow to reduce the size of the space of data to generate and hence improve performances.

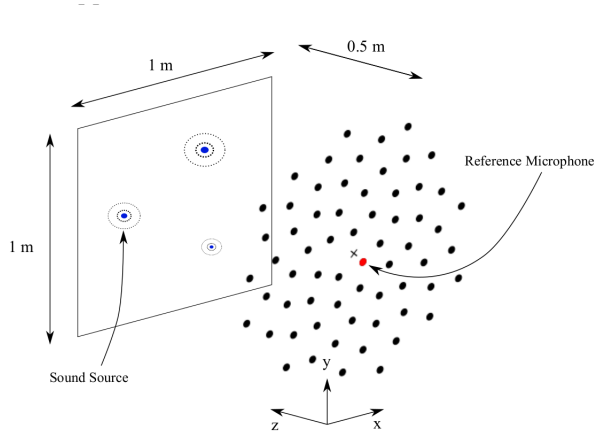

Figure 2.1: Example of the measurement setup used, with three audio sources

2.1 Data

The goal of this thesis was to generate data as realistic as possible, by learning the distribution of Cross Spectral Matrices (CSM) of recorded pressure vectors. As mentioned in the introduction using only real measurment was not a feasible, since the number of measurement required for training is too big. For this reason, the chosen approach was to first pretrain the models using the eigendecomposition of synthetic CSMs and then fine-tune our model using eigendecomposition of CSMs of recorded pressure vectors p.

Measurement

In order to generate realistic data, measurement needed to be performed. An example of the measurement setup used is displayed in Fig.2.1. It is made of audio sources located in a 1m $\times$ 1m plane located half a meter in front an array of 64 microphones. The microphones are organized in a circular pattern, with the central one acting as the reference microphone. This means that when the CSMs are computed, they are normalized using the autopower of the reference microphone. When performing the actual measurement, we only consider the case of a single audio source. The loudspeaker used in our measurement setup is displayed in Fig.2.2 and the microphone array in Fig.2.3.

The measurements are sound pressure $\mathbf{p} \in \mathbb{C}^{64}$ at the 64 microphones of the array. Every recording is for a different position of the audio source. All measurements have a duration 10s and a sampling frequency of 51.2kHz.

From one measurement, we wanted to create many different approximation of the CSM $\hat{\mathbf{C}}$ to have enough data to fine tune our generative model. To do so, we took slices of $\frac{1}{50}$ of total measurement length (i.e. 0.2s). From this slice, B snapshots needed to be created, for the approximation the CSM $\hat{\mathbf{C}}$ (see equation 1.3). From [29], we know that the number of snapshots B needed to be at least 4 times the number of receivers M of the array (i.e. at least

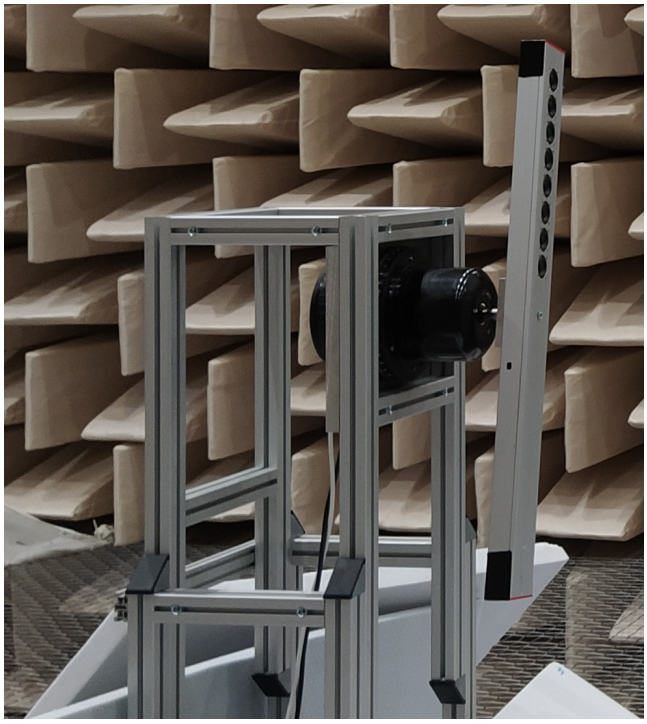

Figure 2.2: Device used as audio source to create the measurement

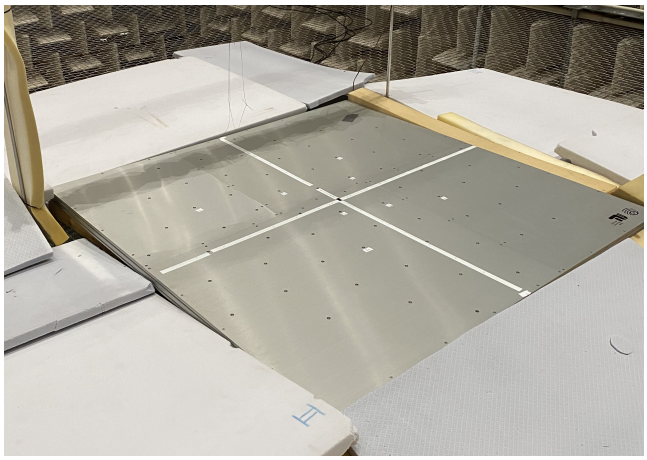

Figure 2.3: Picture of the array of microphone used to create the real measurement

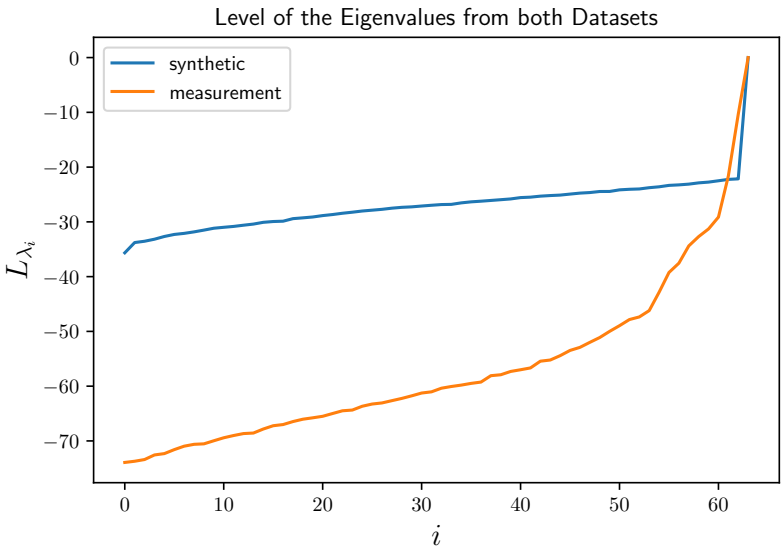

Figure 2.4: Levels of the eigenvalues of a synthetized CSM (blue) and of a measured CSM (orange).

$4 \cdot 64 = 256$). For this purpose we used snapshots overlapping at 75%. Indeed, [29] argues that such a number of snapshots is required, for the eigenvalues of the approximated CSM $\hat{\mathbf{C}}$ to be sufficiently close to the eigenvalues of the real CSM $\mathbf{C}$. Having snapshots overlapping at 75% allowed us to have $B = 317 > 256$.

Should something else be specified about the used microphones/loudspeaker?

Synthetic Data

When a specific measurement was chosen to create the measurement dataset, its source's position was used to create the corresponding synthetic data. The synthetic CSM were sampled from the complex Wishart distribution using the acoular python package.

Comparison between Synthetic and Measured Data

In Fig.2.4, we show a comparison of the level of eigenvalues of a synthetic CSM compared with a measured CSM. In the synthetic level of eigenvalues, we observe first that all the eigenvalues except the first ones are close to zero. When observing the level of the eigenspectrum of the measured data, a slowlier decay can be observed. We conclude that the measured and synthetic data are indeed not fully similar and hence that synthetized data cannot replace measured data. This difference be explained by the fact that the synthetic CSMs are closed to the perfection of a mathematical model and hence do not show presence of natural noise created by reverberation for instance.

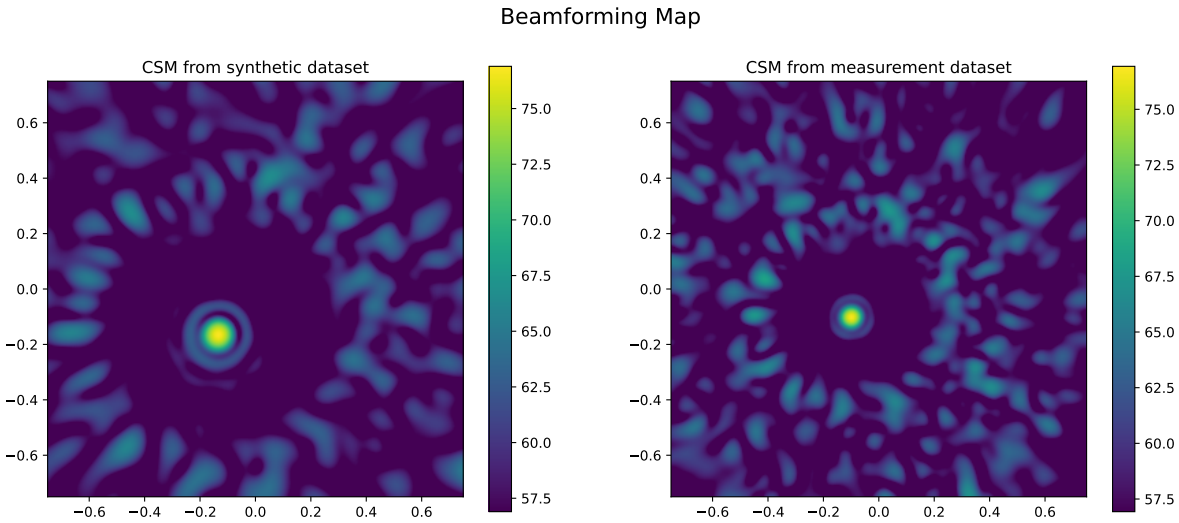

Figure 2.5: Beamforming maps, created respectively with CSMs from the synthetic dataset and from the measurement data

The Datasets Used

When generating the dataset, the case of a single source was considered. The audio source is located at position $[-0.09, -0.111]$ on the plane in front of the microphones array. The CSM are frequency dependent and our datasets consists of CSMs with Helmotz number $He = 16$. The Helmotz number He is commonly used in acoustic to represent frequency and is defined as

$$He = \frac{L_c}{\lambda} = L_c f \tag{2.1}$$

where L_c is the called the characteristic length. Here it is the speed of sound, hence $L_c = 343$ m/s λ is the wavelength under consideration, and hence $f = \frac{1}{\lambda}$ is the frequency under consideration. Example of beamforming maps resulting from CSMs from our datasets are displayed in Fig.2.5 (respectively from synthetic and measurement datasets).

2.2 Architectures

In order to generate the different types of data, WGAN-GPs were designed. The different implementations were adapted from [30]. As mentioned in the fundamentals the basic structure is two competing networks, a generator and a discriminator or critic.

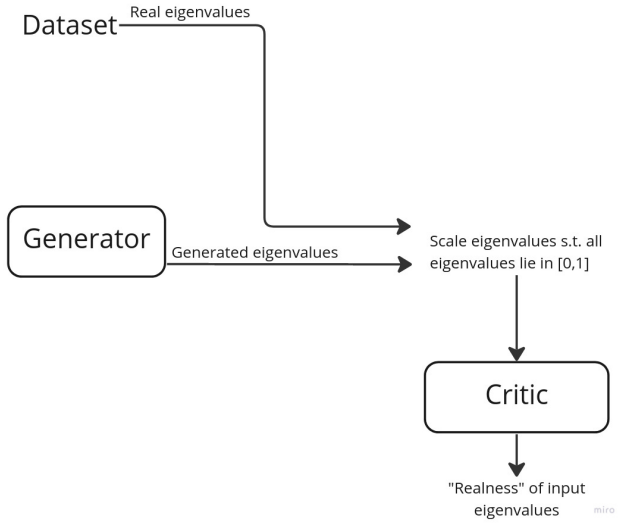

Figure 2.6: Full structure of the used implementation of the WGAN-GP to generate eigenvalues.

2.2.1 Generating Scaled Eigenvalues

The first WGAN-GP created was to generate the scaled eigenvalues $[\lambda_0, \ldots, \lambda_{63}] \in [0,1]^{64}$. These eigenvalues are sorted such that $\lambda_0 \leq \lambda_1 \leq \cdots \leq \lambda_{63}$. Unlike in the implementation of [30], when generating the eigenvalues, both inner networks (generator and critic) do not have a convolutional but a perceptron structure. Indeed, a convolutional structure is relevant when the data to generate is an image. Indeed, a convolutional layer is good at seeing patterns in image, by identifying topological structures in neighbouring pixels. We could have stacked pieces of our vector of eigenvalues $\lambda = [\lambda_0, \ldots, \lambda_{63}] \in \mathbb{R}^{64}$ into an image of dimension 8×8 but this would have resulted in an image where most neighbouring pixels are not sharing any information about each other, hence making the convolutional operation irrelevant at detecting pattern, at least compared to images.

Another changes that was made compared to the implementation of [30], is that a ReLU activation function was added as the last layer of the generator. Indeed this change was made so that the generator do not generate eigenvalues smaller than zero. Moreover, we added a small noise $n = 10^{-100}$ to every eigenvalues produced to ensure that they are positive.

Finally, before being fed to the discriminator, generated eigenvalues are scaled such that $0 \leq \lambda_i \leq 1$ for $\lambda_i \in \lambda$. Normalizing eigenvalues in this way is done to reduce the size of the sample space. This is illustrated in Fig.2.6 **TODO: redo this figure as TikZ**. Both architectures of the generator and critic used are illustrated respectively in tables Tab.A.1 and Tab.A.2.

Generating the Levels of Scaled Eigenvalues

Another approach to generate the scaled eigenvalues $\lambda_0, \ldots, \lambda_{63}$ was to generate them from their level values $L_{\lambda_0}, \ldots, L_{\lambda_{63}}$. The levels are computed with:

$$L_{\lambda_i} = 10 \log_{10}(\frac{\lambda_i}{\lambda_{63}}) \tag{2.2}$$

where λ_{63} is the biggest eigenvalue. Since all values L_{λ_i} are non-positive, the generator has been built such that the two last steps are first a ReLU, followed by a multiplication by -1. This way we can ensure that our network only produces non-positive spectrum. This approach was also justified by the usage of Leaky ReLUs as activation throughout the generator.

Because we were also using Leaky ReLUs in the critic, the first layer of its network is also a multiplicative layer. Therefore, the critic is not trained to detect real from fake levels, but rather real from fake negative levels, which is equivalent.

The architecures used for the generator and the critic are shown respectively in Tab.A.3 and Tab.A.4.

2.2.2 Generating Eigenvectors

In order to generate the eigenvector, we decide to start by only generating the eigenvector corresponding to the highest eigenvalue, that we define as main eigenvector. Indeed, using equation 1.5, once a main eigenvector is produced, its quality can be directly assessed by computing a Rank I CSM, allowing us to perform beamforming and locate identify the source position. We show in Fig.2.7, an beamforming map, created with a rank I CSM obtained from measurement data.

Moreover, first generating only the main eigenvector can be justified by the fact, it is the most meaningful eigenvector. Indeed, each eigenvector belongs to a different incoherent source. If there would to be multiple coherent source, all the energy and their position merges into a single eigenmode. Since we consider the case where there is only one source, only the eigenvector with the biggest index corresponds to a DoA/source and the remaining 63 eigenvectors corresponds to noise.

In Fig.2.8, we plot the histograms of the values of the scalars of the eigenvectors with different index. It can be noticed that the values of the eigenvectors with low index seem to follow a gaussian distribution, whereas the values of the eigenvectors with index close to the maximal index are following a more complex distribution.

Main Eigenvector

Since the main eigenvector is $\in \mathbb{C}^{64}$, a network with a similar architecture as the one used for the eigenvalues can be used. But instead of scaling generated sample, they are normalized before being fed to the discriminator. The full

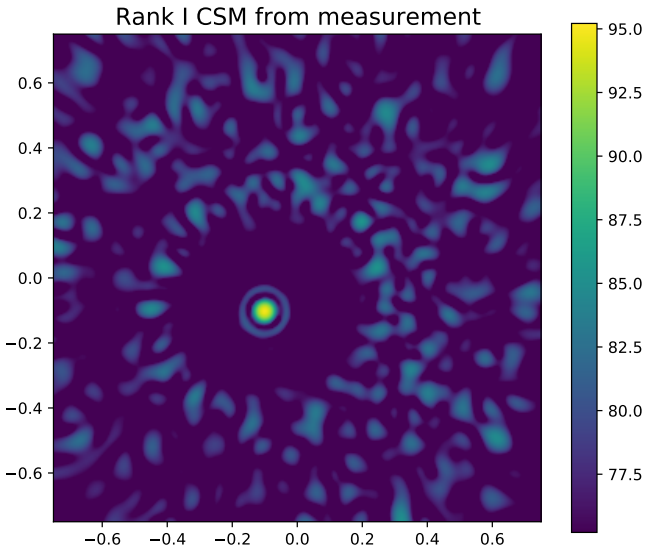

Figure 2.7: Beamforming map, created with a rank I CSM created from measurement data

WGAN-GP structure is illutrates in Fig.2.9 **TODO: redo this figure as TikZ**. The architectures both the generator and critic are illustrated respectively in the tables Tab.A.5 and Tab.A.6.

Noise Eigenvectors

2.2.3 Cross Spectral Matrix Snapshots

2.3 Data Augmentation

An idea for augmenting datasets of CSM of recorded pressure vectors $\mathbf{p}\mathbf{p}^H$ is introduced. First, for a set of received CSM, compute their eigendecomposition $\hat{\mathbf{C}} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^H$. The set of all eigenvalues can then be used to train a WGAN-GP. Using this network, eigenvalues $\hat{\mathbf{\Lambda}}$ can be generated. Then using the generated eigenvalues $\hat{\mathbf{\Lambda}}$ with real eigenvectors $\mathbf{V}$ we can have semi-generated CSM:

$$\hat{\mathbf{C}}_{\text{augm.}} = \mathbf{V}\hat{\mathbf{\Lambda}}\mathbf{V}^H \tag{2.3}$$

TODO: need to mention the evals are generated as levels

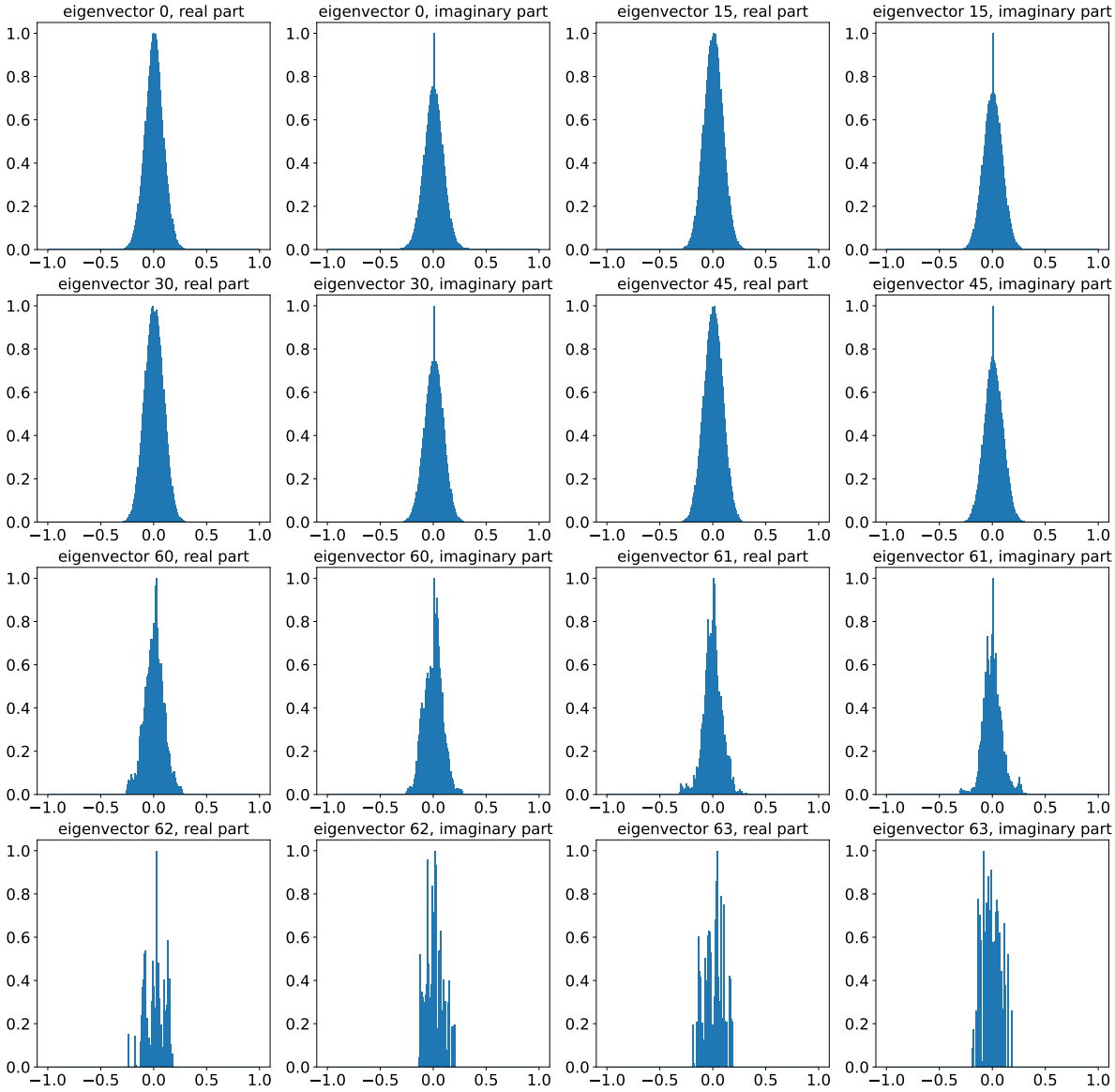

Figure 2.8: Histograms of the values of the scalars of the eigenvectors with different index. The eigenvector with index 63 is the main eigenvector (two last histograms).

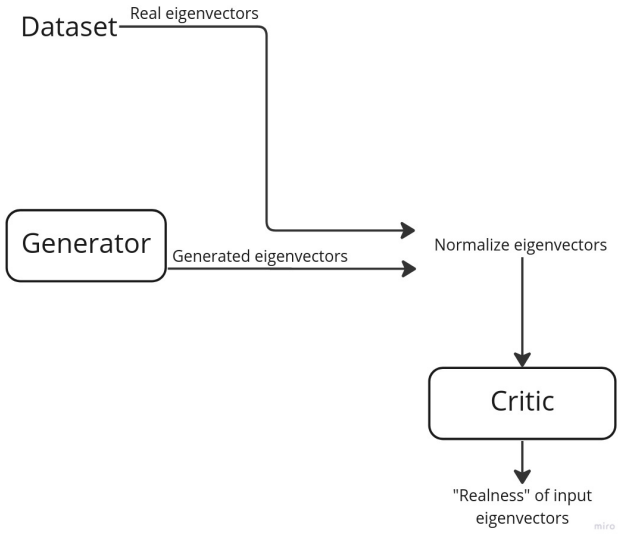

Figure 2.9: Full structure of the used implementation of the WGAN-GP to generate eigenvectors.

2.4 Parameters

The WGAN-GP for generating the scaled eigenvalues, the level of eigenvalues and the main eigenvectors all used the same parameters. The batch size is set to 16 and the latent dimension to 128. The optimizers used both for the critic and generator are Adam optimizers with a learning rate $lr = 0.0002$, $\beta_1 = 0.5$ and $\beta_2 = 0.9$. Every time the generator was trained once, the critic was trained 3 times (i.e $n_{critic} = 3$). The weight of the gradient penalty λ is equal to 10. All three WGAN-Gp were trained on the same GPU (**TODO: find more information about this**). When performing the training, the three networks were not trained with the same number of epoch or step per epoch, namely:

- Training the WGAN-GP for generating the scaled eigenvalue took 19s during the initial training. The initial training consisted of 100 epochs, with 1 step per epoch. The fine-tuning took 10s and consisted of 200 epochs, with 5 steps per epoch.
- Training the WGAN-GP for generating the level of eigenvalue took 19s during the initial training. The initial training consisted of 100 epochs, with 1 step per epoch. The fine-tuning took 1.9s and consisted of 120 epochs, with steps per epoch.
- Training the WGAN-GP for generating the main eigenvector took 84s during the initial training. The initial training consisted of 500 epochs, with 1 step per epoch. The fine-tuning took 32s and consisted of 200 epochs, with 1 steps per epoch.

Chapter 3

Results

3.1 Generating Eigenvalues from their Scaled Values

In Fig.3.1, we show the loss function when respectively training with the synthetic data and fine-tuning with measurement data. It can be observed that the model converged before the fine-tuning and that it did not have to change so much to adapt to generate measurement data, since the critic's loss function remain really close to zero.

In Fig.3.2, we show first an example eigenvalues from the synthetic dataset (color), an example of eigenvalues from the measurement dataset (color), sample eigenvalues generated by the WGAN-GP before the fine-tuning (color) and sample eigenvalues generated by the WGAN-GP after fine-tuning (color). In the second graph, the levels of the same eigenvalues is displayed. In the first graph, it can be observed that the sample eigenvalues generated by the WGAN-GP before fine-tuning seem to follow the eigenvalues from the synthetic dataset and the sample eigenvalues generated by the WGAN-GP after fine-tuning are quite similar to the eigenvalues from the measurement dataset.

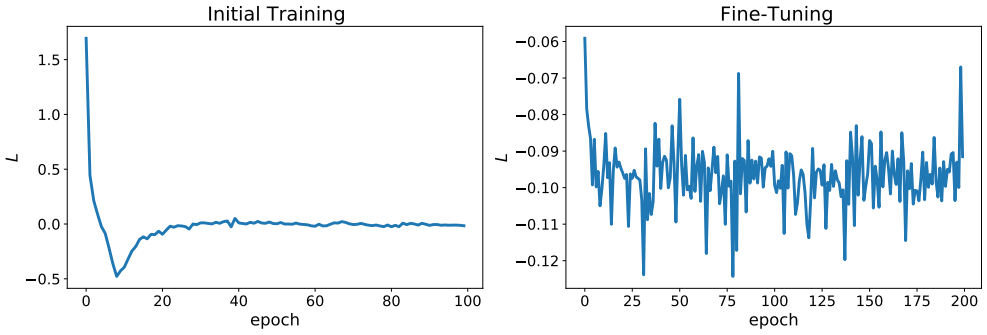

Figure 3.1: The loss functions of the critic of the WGAN-GP for eigenvalues generation, respectively while performing the initial training and while fine-tuning

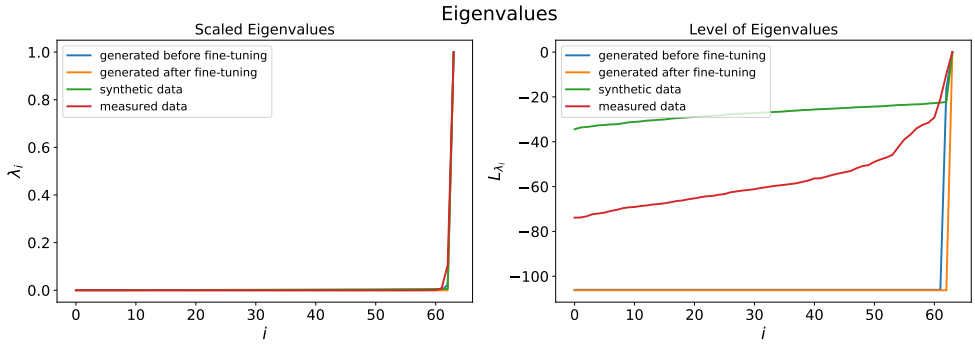

Figure 3.2: The first graph shows eigenvalues of a synthetic CSM (green), eigenvalues of a measured CSM (red), eigenvalues generated with the WGAN-GP for scaled eigenvalues after the initial training (blue) and eigenvalues generated with the WGAN-GP for scaled eigenvalues after the fine-tuning (orange). The second graph displays the levels of the same eigenvalues.

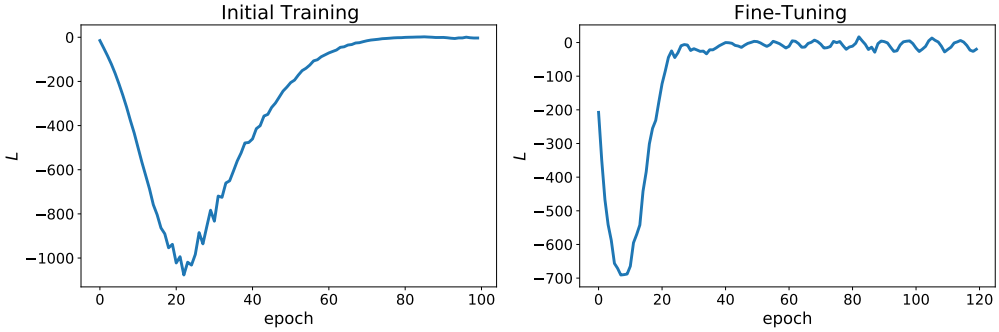

Figure 3.3: The loss functions of the critic of the WGAN-GP for the levels of eigenvalues generation, respectively while performing the initial training and while fine-tuning

In the levels representation of the sample, both the samples generated before and after fine-tuning show a sudden and steep drop in value, reaching a level around 10^{-100}

3.2 Generating Eigenvalues from their Level Values

The losses during training of the critic of the WGAN-GP for generating eigenvalues from their level values are displayed in Fig.3.3. It can be seen that our model converged before fine-tuning. Then, during the fine-tuning, the loss drops at first before converging again.

In Fig. 3.4 is displayed a comparison between levels of the eigenvalues from the synthetic dataset (color), levels of the eigenvalues from the measurement dataset (color), sample levels of eigenvalues generated by the WGAN-GP before fine-tuning (color) and sample levels of eigenvalues generated by the

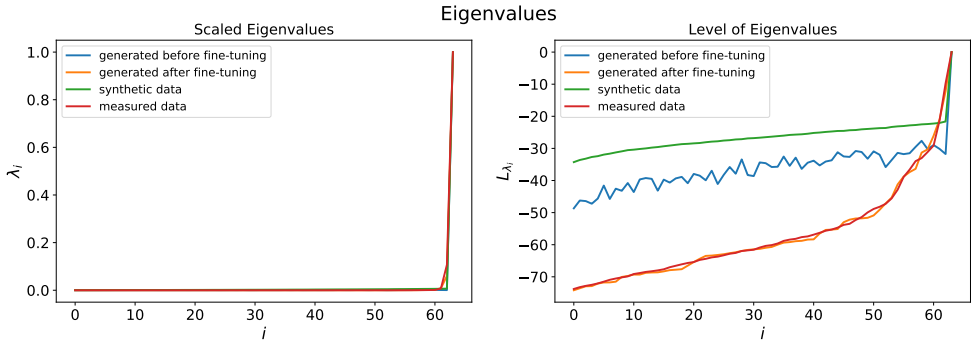

Figure 3.4: The first graph shows eigenvalues of a synthetic CSM (green), eigenvalues of a measured CSM (red), eigenvalues generated with the WGAN-GP for level of eigenvalues after the initial training (blue) and eigenvalues generated with the WGAN-GP for level of eigenvalues after the fine-tuning (orange). The second graph displays the levels of the same eigenvalues.

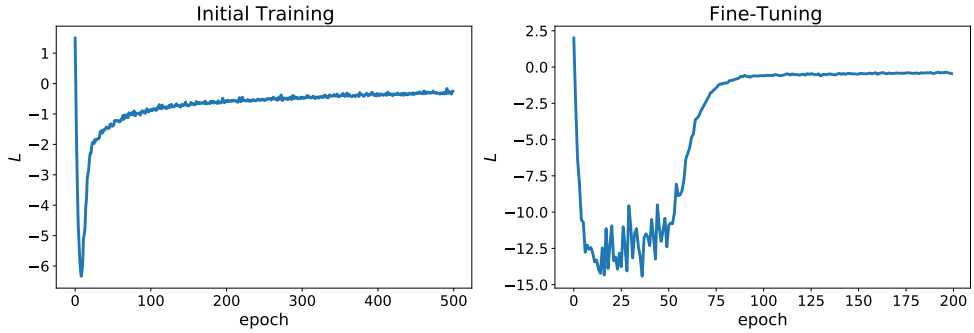

Figure 3.5: The loss functions of the critic of the WGAN-GP for the main eigenvector generation, respectively while performing the initial training and while fine-tuning

WGAN-GP after fine-tuning (color). It can be seen that the sample generated before fine-tuning follow quite well the levels from the synthetic dataset, though not decaying as linearly. The sample generated after fine-tuning display a similar decay as the levels from the measurement dataset.

Then the comparison between the same quantities is shown, but this time not using the levels values L_{λ_i}, but the actual eigenvalues λ_i.

3.3 Generating the Main Eigenvector

When training the network we observe the losses shown in Fig.3.5 (respectively initial training and fine-tuning). It can be seen that the network has converged during the initial training, and had to adapt during the fine-tuning but also shows convergence.

Unlike for the eigenvalues, showing a generated sample of main eigenvector

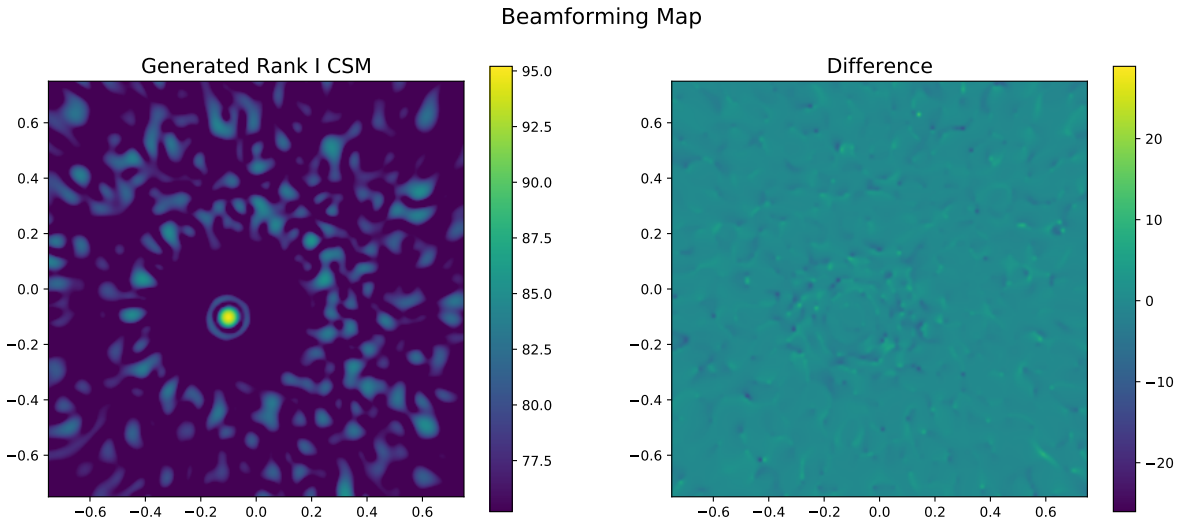

Figure 3.6: The first beamforming map has been perfomed with a Rank I CSM computed with a generated main eigenvector. The second map shows the difference between the beamforming map obtained with Rank I CSM computed respectively with a generated main eigenvector and a main eigenvector from the dataset.

is not going to be really helpful in assessing the quality of the generated sample. Therefore, instead we do the following: generate a sample main eigenvector $\hat{\mathbf{v}}_{63}$ and use equation 1.5 to create a rank I CSM. It is important to note that the eigenvalue $\mathbf{\Lambda}_{63,63}$ corresponding to the main eigenvector is by definition equal to one. Indeed, it is by defintion the biggest eigenvalues and we scaled the eigenvalues such that all eigenvalues lie in $[0, 1]$. We plot first in Fig.3.6 the beamforming map resulting from performing beamforming with the Rank I CSM created from a sampled main eigenvector. In the second map of Fig.3.6, we plot the difference between a beamforming map computed with a generated main eigenvector and the beamforming map computed with a main eigenvector from the measurement dataset (i.e. map shown in Fig.**TODO: add ref**). We can observe that the difference of the two map is almost zero everywhere.

3.4 Generating All the Eigenvectors

3.5 Generating the Snapshots

3.6 Data Augmentation

Fig.3.7 shows in the first map the result of performing beamforming on a augmented CSM issued from the dataset and from a semi generated CSM.

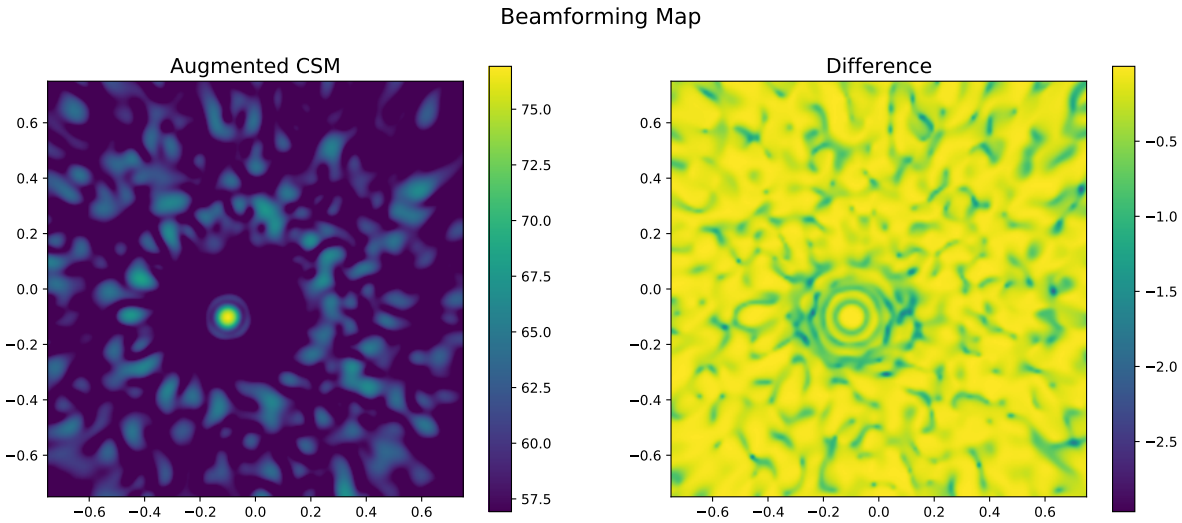

Figure 3.7: The first beamforming map has been perfomed with a CSM computed with the data augemntation scheme. The second map shwos the difference between the beamforming map obtained with CSM from respectively the data augmentation dataset and the training dataset

The second image shows the difference between the beamforming map obtained from an augmented CSM and a CSM from the datasets used to generate it. It can be observed that the map is almost zero everywhere, but some artifacts can still be observed.

Chapter 4

Discussions

4.1 Generating Eigenvalues from their Scaled Values

It was observed in the first plot of Fig.3.2 that the WGAN-GP designed to generate eigenvalues seem to produce sufficiently realistic results, both for synthetic and measurement data when only looking at the eigenvalues, but when looking at the levels of the eigenvalues, the difference between real and generated samples was obvious. This can be explained by the fact, that the WGAN-GP cannot capture very well small numerical variation. This lack of extreme precision is very visible in the eigenvalues close to zero. Indeed, microvariation around zero lead to big variation in the level values. Since the level of the spectrum, is the way used to tell synthetic data from measured data, using a Generative approach producing eigenvalues with unrealistic levels spectrum is not conceivable.

4.2 Generating Eigenvalues from their Level Values

It can be observed that learning the levels of the eigenvalues leads to a network that is able to generate sample levels of eigenvalues more realistic than when generating sample eigenvalues and then computing their levels. But moreover, when converting back the levels of eigenvalues to eigenvalues, it can be observed that the sample are still of higher quality than when generating the eigenvalues directly. We conclude than this second method is better suited for generating eigenvalues of CSM.

4.3 Generating the Main Eigenvector

When observing the beamforming map computed from generated main eigenvector, it can be concluded that WGAN-GP is allowing us to generate sample close to the original main eigenvector, without being 100% similar. Indeed on the beamforming map, the audio source can be easily located, and the

difference of between two beamforming map is not zero everywhere.

4.4 Generating All the Eigenvectors

4.5 Generating the Snapshots

4.6 Data Augmentation

We introduce a method for data augmentation that produce good results

- It produce good results
- Say it is a very efficient way to have data augmentation
- Mention that maybe the sample generated this way are too close from original to be really a good data augmentation technique

4.7 General Discussion

Summary of what has been done

We can generate realistic eigenvalue spectrum and main eigenvector

The eigenvalue spectrum are not position dependent, hence no modification is really needed to the current network for a later use in generating CSM.

The issue with learning only one eigenvector, is that this method only allows for generating CSM for a single position.

Accuracy of the results: The data generated are accurate. When replacing real data by generated data, in the case of rank I beamforming or data augmentation for the eigenvalues, beamforming could be perfomed providing with sufficiently clear beamforming map.

Limitations of the results: only generating data for a single postion. The current model is not learning yet how to generate CSM for position that it has not seen. The data generated currently is not a 100% realistic since none of the noise eigenvectors have been generated.

from thesis task "A discussion regarding the computational demands and the accuracy as well as the validity of the sampled results should be given at the end of the thesis."

Chapter 5

Future Works

Investigating the noise eigenvector, e.g. using TransGAN Creating a conditional WGAN for the main eigenvector. This could be an easy first step for generating CSM (multichannel array data) for a single source.

Chapter 6

Conclusion

Appendix A

Architectures of the Generator and Critic Networks

TODO: to write: Tab.A.1, Tab.A.2, Tab.A.3, Tab.A.4, Tab.A.5, Tab.A.6, Tab.A.7, Tab.A.8, Tab.A.9 and Tab.A.10

Layer	Output Shape	Number of Parameters
InputLayer	128	0
Dense	256	32768
LeakyReLU	256	0
BatchNormalization	256	1024
Dense	512	131584
LeakyReLU	512	0
Dense	1024	525312
LeakyReLU	1024	0
Dense	64	65600

Table A.1: Architecture of the generator used in the WGAN-GP to generate eigenvalues. Total params: 756,288, Trainable params: 755,776, Non-trainable params: 512

Layer	Output Shape	Number of Parameters
InputLayer	64	0
Dense	512	33280
LeakyReLU	512	0
Dense	256	131328
LeakyReLU	256	0
Dense	1	257

Table A.2: Architecture of the critic used in the WGAN-GP to generate eigenvalues. Total params: 164,865, Trainable params: 164,865, Non-trainable params: 0

Layer	Output Shape	Number of Parameters
InputLayer	128	0
Dense	256	32768
LeakyReLU	256	0
BatchNormalization	256	1024
Dense	512	131584
LeakyReLU	512	0
Dense	1024	525312
LeakyReLU	1024	0
Dense	64	65600
ReLU	64	0
Multiply	64	0

Table A.3: Architecture of the generator used in the WGAN-GP to generate eigenvalues from their level values. Total params: 756,288, Trainable params: 755,776, Non-trainable params: 512

Layer	Output Shape	Number of Parameters
InputLayer	64	0
Multiply	64	0
Dense	512	33280
LeakyReLU	512	0
Dense	256	131328
LeakyReLU	256	0
Dense	1	257

Table A.4: Architecture of the critic used in the WGAN-GP to generate eigenvalues from their level values. Total params: 164,865, Trainable params: 164,865, Non-trainable params: 0

Layer	Output Shape	Number of Parameters
InputLayer	128	0
Dense	256	32768
LeakyReLU	256	0
BatchNormalization	256	1024
Dense	512	131584
LeakyReLU	512	0
Dense	1024	525312
LeakyReLU	1024	0
Dense	128	131200
Reshape	(1, 64, 2)	0

Table A.5: Architecture of the generator used in the WGAN-GP to generate the main eigenvector. Total params: 821,888, Trainable params: 821,376, Non-trainable params: 512

Layer	Output Shape	Number of Parameters
InputLayer	(1, 64, 2)	0
Flatten	128	0
Dense	512	66048
LeakyReLU	512	0
Dense	256	131328
LeakyReLU	256	0
Dense	1	257

Table A.6: Architecture of the critic used in the WGAN-GP to generate the main eigenvector. Total params: 197,633, Trainable params: 197,633, Non-trainable params: 0

Layer	Output Shape	Number of Parameters
InputLayer	128	0
Dense	256	32768
BatchNormalization	256	1024
LeakyReLU	256	0
Dense	1024	262144
LeakyReLU	1024	0
Dense	4096	4194304
LeakyReLU	4096	0
Dense	8064	33030144
LeakyReLU	8064	0
Reshape	(63, 64, 2)	0

Table A.7: Architecture of the generator used in the WGAN-GP to generate the noise eigenvectors. Total params: 37,520,384, Trainable params: 37,519,872, Non-trainable params: 512

Layer	Output Shape	Number of Parameters
InputLayer	(63, 64, 2)	0
Flatten	8064	0
Dense	4096	33034240
LeakyReLU	4096	0
Dense	512	2097664
LeakyReLU	512	0
Dense	256	131328
LeakyReLU	256	0
Dense	1	257

Table A.8: Architecture of the critic used in the WGAN-GP to generate the noise eigenvectors. Total params: 35,263,489, Trainable params: 35,263,489, Non-trainable params: 0

Layer	Output Shape	Number of Parameters
InputLayer	128	0
Dense	4096	524288
BatchNormalization	4096	16384
LeakyReLU	4096	0
Reshape	(4, 4, 256)	0
UpSampling2D	(8, 8, 256)	0
Conv2D	(8, 8, 128)	294912
BatchNormalization	(8, 8, 128)	512
LeakyReLU	(8, 8, 128)	0
UpSampling2D	(16, 16, 128)	0
Conv2D	(16, 16, 64)	73728
BatchNormalization	(16, 16, 64)	256
LeakyReLU	(16, 16, 64)	0
UpSampling2D	(32, 32, 64)	0
Conv2D	(32, 32, 128)	73728
BatchNormalization	(32, 32, 128)	512
LeakyReLU	(32, 32, 128)	0
UpSampling2D	(64, 64, 128)	0
Conv2D	(64, 64, 2)	2304
BatchNormalization	(64, 64, 2)	8
Activation	(64, 64, 2)	0

Table A.9: Architecture of the generator used in the WGAN-GP to generate snapshots used for creating an approximated CSM. Total params: 986,632, Trainable params: 977,796, Non-trainable params: 8,836

Layer	Output Shape	Number of Parameters
InputLayer	(64, 64, 2)	0
ZeroPadding2D	(68, 68, 2)	0
Conv2D	(34, 34, 64)	3264
LeakyReLU	(34, 34, 64)	0
Conv2D	(17, 17, 128)	204928
LeakyReLU	(17, 17, 128)	0
Dropout	(17, 17, 128)	0
Conv2D	(9, 9, 256)	819456
LeakyReLU	(9, 9, 256)	0
Dropout	(9, 9, 256)	0
Conv2D	(5, 5, 512)	3277312
LeakyReLU	(5, 5, 512)	0
Flatten	12800	0
Dropout	12800	0
Dense	1	12801

Table A.10: Architecture of the critic used in the WGAN-GP to generate snapshots used for creating an approximated CSM. Total params: 4,317,761, Trainable params: 4,317,761, Non-trainable params: 0

Bibliography

[1] P. Castellini, N. Giulietti, N. Falcionelli, A. F. Dragoni, and P. Chiariotti, "A neural network based microphone array approach to grid-less noise source localization," *Applied Acoustics*, vol. 177, p. 107947, 2021.

[2] A. Kujawski, G. Herold, and E. Sarradj, "A deep learning method for grid-free localization and quantification of sound sources," *The Journal of the Acoustical Society of America*, vol. 146, no. 3, pp. EL225–EL231, 2019.

[3] S. Y. Lee, J. Chang, and S. Lee, "Deep learning-based method for multiple sound source localization with high resolution and accuracy," *Mechanical Systems and Signal Processing*, vol. 161, p. 107959, 2021.

[4] W. Ma and X. Liu, "Phased microphone array for sound source localization with deep learning," *Aerospace Systems*, vol. 2, no. 2, pp. 71–81, 2019.

[5] W. G. Pinto, M. Bauerheim, and H. Parisot-Dupuis, "Deconvoluting acoustic beamforming maps with a deep neural network," 2021.

[6] P. Xu, E. J. Arcondoulis, and Y. Liu, "Deep neural network models for acoustic source localization," in *Berlin Beamforming Conference*, 2021.

[7] W. He, P. Motlicek, and J.-M. Odobez, "Deep neural networks for multiple speaker detection and localization," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 74–79.

[8] E. L. Ferguson, S. B. Williams, and C. T. Jin, "Sound source localization in a multipath environment using convolutional neural networks," in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2018, pp. 2386–2390.

[9] S. Chakrabarty and E. A. Habets, "Broadband doa estimation using convolutional neural networks trained with noise signals," in *2017 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA)*. IEEE, 2017, pp. 136–140.

[10] L. Perotin, R. Serizel, E. Vincent, and A. Guérin, "Crnn-based joint azimuth and elevation localization with the ambisonics intensity vector,"

in *2018 16th International Workshop on Acoustic Signal Enhancement (IWAENC)*. IEEE, 2018, pp. 241–245.

[11] S. Adavanne, A. Politis, and T. Virtanen, "Direction of arrival estimation for multiple sound sources using convolutional recurrent neural network," in *2018 26th European Signal Processing Conference (EUSIPCO)*. IEEE, 2018, pp. 1462–1466.

[12] R. Takeda and K. Komatani, "Sound source localization based on deep neural networks with directional activate function exploiting phase information," in *2016 IEEE international conference on acoustics, speech and signal processing (ICASSP)*. IEEE, 2016, pp. 405–409.

[13] P. Neekhara, C. Donahue, M. Puckette, S. Dubnov, and J. McAuley, "Expediting tts synthesis with adversarial vocoding," *arXiv preprint arXiv:1904.07944*, 2019.

[14] K. Kumar, R. Kumar, T. de Boissiere, L. Gestin, W. Z. Teoh, J. Sotelo, A. de Brébisson, Y. Bengio, and A. C. Courville, "Melgan: Generative adversarial networks for conditional waveform synthesis," in *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., vol. 32. Curran Associates, Inc., 2019. [Online]. Available: https://proceedings.neurips.cc/paper/2019/file/6804c9bca0a615bdb9374d00a9fcba59-Paper.pdf

[15] J. Engel, K. K. Agrawal, S. Chen, I. Gulrajani, C. Donahue, and A. Roberts, "Gansynth: Adversarial neural audio synthesis," *arXiv preprint arXiv:1902.08710*, 2019.

[16] E. Vargas, J. R. Hopgood, K. Brown, and K. Subr, "On improved training of cnn for acoustic source localisation," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 29, pp. 720–732, 2021.

[17] C. Papayiannis, C. Evers, and P. A. Naylor, "Data augmentation of room classifiers using generative adversarial networks," *arXiv preprint arXiv:1901.03257*, 2019.

[18] A. Ratnarajah, S.-X. Zhang, M. Yu, Z. Tang, D. Manocha, and D. Yu, "Fast-rir: Fast neural diffuse room impulse response generator. 10.48550," *arXiv preprint ARXIV.2110.04057*, 2021.

[19] M. J. Bianco, S. Gannot, and P. Gerstoft, "Semi-supervised source localization with deep generative modeling," in *2020 IEEE 30th International Workshop on Machine Learning for Signal Processing (MLSP)*. IEEE, 2020, pp. 1–6.

[20] P. Gerstoft, H. Groll, and C. F. Mecklenbräuker, "Parametric bootstrapping of array data with a generative adversarial network," in *2020 IEEE*

11th Sensor Array and Multichannel Signal Processing Workshop (SAM). IEEE, 2020, pp. 1–5.

[21] F. Hübner, W. Mack, and E. A. Habets, "Efficient training data generation for phase-based doa estimation," in *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2021, pp. 456–460.

[22] J. M. Vera-Diaz, D. Pizarro, and J. Macias-Guarasa, "Acoustic source localization with deep generalized cross correlations," *Signal Processing*, vol. 187, p. 108169, 2021.

[23] E. Sarradj, "A fast signal subspace approach for the determination of absolute levels from phased microphone array measurements," *Journal of Sound and Vibration*, vol. 329, no. 9, pp. 1553–1569, 2010.

[24] ——, "Three-dimensional acoustic source mapping with different beamforming steering vector formulations," *Advances in Acoustics and Vibration*, vol. 2012, 2012.

[25] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial networks," *Communications of the ACM*, vol. 63, no. 11, pp. 139–144, 2020.

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

[27] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein generative adversarial networks," in *International conference on machine learning*. PMLR, 2017, pp. 214–223.

[28] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C.Courville, "Improved training of wasserstein gans," *CoRR*, vol. abs/1704.00028, 2017. [Online]. Available: http://arxiv.org/abs/1704.00028

[29] P. Gerstoft, R. Menon, W. S. Hodgkiss, and C. F. Mecklenbräuker, "Eigenvalues of the sample covariance matrix for a towed array," *The Journal of the Acoustical Society of America*, vol. 132, no. 4, pp. 2388–2396, 2012.

[30] A. Nain, "Wgan-gp overriding model train step," May 2020. [Online]. Available: https://keras.io/examples/generative/wgan{_}gp/

List of Figures

List of Tables

Here you must include the signed declaration of originality. You find the form on the lab's website. Print it out, sign it and scan the signed form.